UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA

HARDVERSKA AKCELERACIJA KONVOLUCIONIH NEURONSKIH MREŽA U EMBEDED SISTEMIMA

DOKTORSKA DISERTACIJA

Mentor:
prof. dr Rastislav Struharik

Kandidat:
Damjan Rakanović

Novi sad, 2022 godine

УНИВЕРЗИТЕТ У НОВОМ САДУ **ОБРАЗАЦ – 5а**
НАВЕСТИ НАЗИВ ФАКУЛТЕТА ИЛИ ЦЕНТРА

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Дамјан Ракановић
Ментор (титула, име, презиме, звање, институција)	Др Растислав Струхарик, редовни професор, Факултет техничких наука
Наслов рада:	Хардверска акцелерација конволуционих неуронских мрежа у ембедед системима
Језик публикације (писмо):	Српски (латиница)
Физички опис рада:	Унети број: Страница 179 Поглавља 9 Референци 58 Табела 16 Слика 68 Графикона 2 Прилога 0
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Електроника
Кључне речи / предметна одредница:	Конволуционе неуронске мреже, хардверска акцелерација, орезивање конволуционих неуронских мрежа, ФПГА
Резиме на језику рада:	Протекла деценија је донела изузетан развој вештачке интелигенције и њених подобласти, посебно машинског учења. Значајно побољшање резултата довело је до раста интересовања не само у оквиру научне заједнице већ и електронске индустрије. Шта више, алгоритми машинског учења, су нашли широку примену и изван наведених оквира. Иако је целокупна подобласт доживела снажан развој и примену у разним сферама, једна од најзначајнијих тиче се обраде слика (класификација слика по разним критеријумима, локализација објеката, сегментација и слично). У оваквим задацима највише су се истакле конволуционе неуронске мреже (класа вештачких неуронских мрежа). Једна од мана овог типа вештачких неуронских мрежа огледа се у количини операција које је потребно извршити да би се слика процесирала. Првобитно је ова комплексност ограничавала њихову употребу на снажне рачунарске системе. Раст интересовања за примену достигнућа у разним апликацијама, повукла је развој и специјализованих акцелератора за компактне електронске системе. Како би се превазишла ограничења ресурса каква владају у оваквим системима, потребно је развити архитектуру која ефикасно користи расположиве ресурсе. Тема ове докторске дисертације је развој баш оваквог акцелератора, намењеног процесирању конволуционих неуронских мрежа. Пошто је реч о

[1] Аутор докторске дисертације потписао је и приложио следеће Обрасце:
5б – Изјава о ауторству;
5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;
5г – Изјава о коришћењу.
Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

	компактном акцелератору, конволуционе неуронске мреже нису процесиране у изворном облику већ су орезане како би се полазна комплексност редуковала, што је уобичајен приступ у оваквим системима. Анализом претходних решења може се закључити да је развој алгоритма за орезивање често био независан од развоја архитектуре што може ограничити крајње резултате акцелератора. Овакав исход је најчешће последица чињенице да алгоритам за орезивање није развијен имајући на уму расположиве хардверске ресурсе који ће бити коришћени приликом имплементације акцелератора. Да би се наведено превазишло, поступак развоја алгоритма за орезивање је у многоме укључио карактеристике крајњег циљаног система, а то је у случају ове дисертације програмабилни хардвер (ФПГА, енгл. *Field-programmable-gate-array*). Остварени резултати иду у прилог хипотези да је паралелни развој специјализованог алгоритма за орезивање и акцелератора од велике важности за постизање високе ефикасности процесирања конволуционих неуронских мрежа у компактним ФПГА системима.
Датум прихватања теме од стране надлежног већа:	28.10.2021.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	Председник: др Иван Мезеи, ванредни професор, Факултет техничких наука Члан: др Татјана Николић, редовни професор, Електронски факултет Ниш Члан: др Дејан Вукобратовић, редовни професор, Факултет техничких наука Члан: др Вук Врањковић, ванредни професор, Факултет техничких наука
Напомена:	

UNIVERSITY OF NOVI SAD
FACULTY OR CENTER

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Damjan Rakanović
Supervisor (title, first name, last name, position, institution)	Prof. Struharik Rastislav, PhD, Faculty of Technical Sciences
Thesis title:	Hardware acceleration of convolutional neural networks in embedded systems
Language of text (script):	Serbian language (latin script)
Physical description:	Number of: Pages 179 Chapters 9 References 58 Tables 16 Illustrations 68 Graphs 2 Appendices 0
Scientific field:	Electrical and Computer Engineering
Scientific subfield (scientific discipline):	Electronics
Subject, Key words:	Convolutional neural networks, hardware acceleration, CNN pruning, FPGA
Abstract in English language:	The past decade has shown an extraordinary development of artificial intelligence and its subfields, especially machine learning. Significant improvement in results has led to an increase in interest not only within the scientific community but also in the electronics industry. Moreover, machine learning algorithms have found wide application beyond these limits. Although the entire subfield has experienced strong development and wide usage in various applications, one of the most used application concerns computer vision (image classification, object localization, segmentation, etc.). Convolutional neural networks (a class of artificial neural networks) has shown the best results among all other algorithms in such tasks. Disadvantage of this type of artificial neural network is reflected in the amount of operations that need to be performed in order to process the image. Initially, this complexity limited their use only to powerful computer systems. However, the growth of interest has led the development of specialized accelerators targeting also compact embedded systems. In order to overcome the resource constraints of such systems, it is necessary to develop an architecture that efficiently uses available resources. The topic of this doctoral thesis is the development of compact accelerator that can efficiently process convolutional neural networks. Since the developed acceelrator is intended for edge applications, convolutional neural networks are not processed in their original form. They are pruned in order to reduce the initial complexity, which is a common approach when targeting embedded systems. An analysis of previous solutions can conclude that the development

[2] The author of doctoral dissertation has signed the following Statements:
5б – Statement on the authority,
5в – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,
5г – Statement on copyright licenses.
The paper and e-versions of Statements are held at he faculty and are not included into the printed thesis.

	of the pruning algorithm was often done independent of the development of the architecture which may limit the end results of the accelerator. This outcome is most often due to the fact that the pruning algorithm was not developed keeping in mind the available hardware. To overcome this, the created pruning algorithm takes into account the characteristics of the target system, which in the case is field-programmable-gate-array system on chip (FPGA SoC). Therefore, pruning algorithm is carefully tailored for FPGAs considering their resource characteristics. The achieved results support the hypothesis that the parallel development of a specialized pruning algorithm and accelerator is of great importance for achieving high processing efficiency of convolutional neural networks which are going to be deployed to the FPGA based systems.
Accepted on Scientific Board on:	28.10.2021.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	President: Assoc Prof. Ivan Mezei PhD, Faculty of Technical Sciences Member: Prof. Tatjana Nikolić PhD, Faculty of Electronic Engineering Member: Prof. Dejan Vukobratović PhD, Faculty of Technical Sciences Member: Assoc Prof. Vuk Vranjković PhD, Faculty of Technical Sciences
Note:	

Rezime

Protekla decenija je donela izuzetan razvoj veštačke inteligencije i njenih podoblasti, posebno mašinskog učenja. Značajno poboljšanje rezultata dovelo je do rasta interesovanja ne samo u okviru naučne zajednice već i elektronske industrije. Šta više, algoritmi mašinskog učenja, su našli široku primenu i izvan navedenih okvira. Iako je celokupna podoblast doživela snažan razvoj i primenu u raznim sferama, jedna od najznačajnijih tiče se obrade slika (klasifikacija slika po raznim kriterijumima, lokalizacija objekata, segmentacija i slično). U ovakvim zadacima najviše su se istakle konvolucione neuronske mreže (klasa veštačkih neuronskih mreža). Jedna od mana ovog tipa veštačkih neuronskih mreža ogleda se u količini operacija koje je potrebno izvršiti da bi se slika procesirala. Prvobitno je ova kompleksnost ograničavala njihovu upotrebu na snažne računarske sisteme. Rast interesovanja za primenu dostignuća u raznim aplikacijama, povukla je razvoj i specijalizovanih akceleratora za kompaktne elektronske sisteme. Kako bi se prevazišla ograničenja resursa kakva vladaju u ovakvim sistemima, potrebno je razviti arhitekturu koja efikasno koristi raspoložive resurse. Tema ove doktorske disertacije je razvoj baš ovakvog akceleratora, namenjenog procesiranju konvolucionih neuronskih mreža. Pošto je reč o kompaktnom akceleratoru, konvolucione neuronske mreže nisu procesirane u izvornom obliku već su orezane kako bi se polazna kompleksnost redukovala, što je uobičajen pristup u ovakvim sistemima. Analizom prethodnih rešenja može se zaključiti da je razvoj algoritma za orezivanje često bio nezavisan od razvoja arhitekture što može ograničiti krajnje rezultate akceleratora. Ovakav ishod je najčešće posledica činjenice da algoritam za orezivanje nije razvijen imajući na umu raspoložive hardverske resurse koji će biti korišćeni prilikom implementacije akceleratora. Da bi se navedeno prevazišlo, postupak razvoja algoritma za orezivanje je u mnogome uključio karakteristike krajnjeg ciljanog sistema, a to je u slučaju ove disertacije programabilni hardver (FPGA, engl. *Field-programmable-gate-array*). Ostvareni rezultati idu u prilog hipotezi da je paralelni razvoj specijalizovanog algoritma za orezivanje i akceleratora od velike važnosti za postizanje visoke efikasnosti procesiranja konvolucionih neuronskih mreža u kompaktnim FPGA sistemima.

Ključne reči: Konvolucione neuronske mreže, hardverska akceleracija, orezivanje konvolucionih neuronskih mreža, FPGA

Abstract

The past decade has shown an extraordinary development of artificial intelligence and its subfields, especially machine learning. Significant improvement in results has led to an increase in interest not only within the scientific community but also in the electronics industry. Moreover, machine learning algorithms have found wide application beyond these limits. Although the entire subfield has experienced strong development and wide usage in various applications, one of the most used application concerns computer vision (image classification, object localization, segmentation, etc.). Convolutional neural networks (a class of artificial neural networks) has shown the best results among all other algorithms in such tasks. Disadvantage of this type of artificial neural network is reflected in the amount of operations that need to be performed in order to process the image. Initially, this complexity limited their use only to powerful computer systems. However, the growth of interest has led the development of specialized accelerators targeting also compact embedded systems. In order to overcome the resource constraints of such systems, it is necessary to develop an architecture that efficiently uses available resources. The topic of this doctoral thesis is the development of compact accelerator that can efficiently process convolutional neural networks. Since the developed accelerator is intended for edge applications, convolutional neural networks are not processed in their original form. They are pruned in order to reduce the initial complexity, which is a common approach when targeting embedded systems. An analysis of previous solutions can conclude that the development of the pruning algorithm was often done independent of the development of the architecture which may limit the end results of the accelerator. This outcome is most often due to the fact that the pruning algorithm was not developed keeping in mind the available hardware. To overcome this, the created pruning algorithm takes into account the characteristics of the target system, which in the case is field-programmable-gate-array system on chip (FPGA SoC). Therefore, pruning algorithm is carefully tailored for FPGAs considering their resource characteristics. The achieved results support the hypothesis that the parallel development of a specialized pruning algorithm and accelerator is of great importance for achieving high processing efficiency of convolutional neural networks which are going to be deployed to the FPGA based systems.

Key words: Convolutional neural networks, hardware acceleration, CNN pruning, FPGA

Sadržaj

Lista slika

Lista tabela

Skraćenice

CNN	Convolutional Neural Network
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuits
AI	Artificial Intelligence
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
ANN	Artificial Neural Network
MAC	Multiply-Accumulate
ReLU	Rectified Linear
ELU	Exponential Linear
LUT	Look-up-table
DSP	Digital Signal Processor
IFM	Input Feature Map
OFM	Output Feature Map
HLS	High Level Synthesis
RTL	Register Transfer Level
RAM	Random Access Memory
BRAM	Block RAM
RBG	Red Blue Green
CLR	Cyclic Learning Rate
OS	Output Stream
IS	Input Stream
CC	Convolutional Core
DLP	Dense Layer Processor
NZV	Non-zero values
NZI	Non-zero index
RR	Round Robin
DM	Data Mover
IB	Input Buffer
OB	Output Buffer
KH	Kernel Height
KW	Kernel Width
IP	Intellectual Property
FSM	Finite State Machine
MB	Mega Byte

Uvod

U poslednjoj deceniji svedoci smo intenzivne aktivnost razvoja veštačke inteligencije, a time i njenih podoblasti, od kojih je jedna i mašinsko učenje. Snažan razvoj oblasti prvobitno je bio podstaknut razvojem hardvera koji je u stanju da u razumnom vremenu obrađuje zahtevane količine podataka. Vremenom je hardver postao ,,snažniji“ što je dalje omogućilo i razvoj na algoritamskoj osnovi. Zbog kvaliteta rezultata i mogućnosti široke primene u raznim oblastima, postalo je jasno da je od izuzetne važnosti omogućiti izvršavanje zahtevnih algoritama mašinskog učenja i na pristupačnim, ugrađenim elektronskim sistemima, u daljem tekstu: embeded sistemima (engl. *embedded systems*). Ovaj tip elektronskih sistema je dostupan velikom broju krajnjih korisnika u okviru *edge-computing* aplikacija što je samo još jedan motiv koji je dodatno pospešio razvoj kako hardverskih akceleratora tako i algoritama smanjene kompleksnosti. U ovakvim sistemima je od krucijalne važnosti kreirati specijalizovan hardverski modul koji izuzetno efikasno koristi raspoložive resurse da bi se iskoristio pun potencijal dostupnog hardvera. Upravo je razvoj takvog, specijalizovanog hardvera tema ove doktorske disertacije. Preciznije, hardverski akcelerator razvijen u okviru disertacije je izuzetno efikasan u pogledu izvršavanja jednog od algoritama mašinskog učenja, konvolucionih neuronskih mreža (CNN). Iako oblast beleži intenzivan razvoj duže vreme još uvek ne možemo reći da konvergira u nekom pravcu, konvolucione neuronske mreže su jedan od najšire korišćenih algoritama posebno u aplikacijama mašinske vizije.

Disertacija je tako koncipirana da je pored uvodnog poglavlja čini još 8 poglavlja koja svojim redosledom približavaju čitaocu kompletan proces razvoja akceleratora korak po korak. U nastavku ove glave predočeni su motivi za razvoj teme kao i osnovni doprinosi doktorske disertacije.

U glavi 2 je dat uvod u mašinsko učenje sa akcentom na konvolucione neuronske mreže. Pored opisa generičkog algoritma i jednostavnog modela CNN-a, detaljno su opisani slojevi koji su zastupljeni u CNN-ovima sa fokusom na način izračunavanja rezultata. Razumevanje predočenih slojeva je od suštinske važnosti za proces razvoja hardvera sposobnog da iste izvršava. Uz navedeno, drugi deo glave 2 daje uvid u aktuelno stanje u oblasti kada je reč o specijalizovanim hardverskim akceleratorima za CNN-ove.

Glava 3 donosi opis algoritma za klasterovanje kernela unutar konvolucionih i *fully-connected* (FC) slojeva. Pored algoritma, navedena je i motivacija za ovakav pristup pripremi mreže za orezivanje kao i postupak klasterovanja sa primerima.

Glava 4 predstavlja poboljšanje postojećeg algoritma za orezivanje CNN-ova sa akcentom na FPGA platforme. Takođe, jasno je analiziran uticaj na redukciju potrebnog hardvera u slučaju da se ovakav pristup koristi u odnosu na postojeći algoritam.

U glavi 5 predstavljena je sinergija algoritma za klasterovanje kernela i orezivanje konvolucionih i FC slojeva, odnosno dva algoritma prikazana u glavama 3 i 4. Uz algoritam, navedeni su i uticaji ovakvog načina orezivanja na potrebe za hardverskim resursima, ali i uticaju na performanse potencijalnih arhitektura.

Razvoj arhitekture koja koristi prednosti razvijenog algoritma je prikazana u glavi 6. Ova glava je ujedno i centralni deo disertacije u kojoj se može prepoznati uobičajen tok razvoja akceleratora. Početak je baziran na algoritamu za izračunavanje odziva konvolucionih slojeva i potencijalna mesta koja se mogu efikasno implementirati na hardveru zarad bržeg procesiranja. Potom je na visokom nivou dat primer uobičajenog toka podataka prilikom procesiranja slojeva CNN-a. Posle uvodnog dela, detaljno je opisan svaki pojedinačni blok arhitekture sa ciljem da se ne izostave činjenice potrebne za potencijalnu reprodukciju rezultata.

Neizbežan proces u razvoju kompleksnog hardvera je i postupak verifikacije koji je predočen u glavi 7. Pored funkcionalne verifikacije, prikazan je i način testiranja na stvarnom *embedded* sistemu za jedan od mnoštva testiranih slojeva.

Glava 8 sadrži eksperimentalne rezultate ostvarenih performansi na FPGA platformi. Pored navedenog, izvršeno je i poređenje sa postojećim akceleratorima koristeći široko prihvaćene metrike.

Glava 9 predstavlja zaključak doktorske disertacije u kojoj su sumirani doprinosi i najznačajniji rezultati prikazani u svim prethodnim glavama.

Na kraju, treba napomenuti da su rezultati u glavama 3, 4, 5, 6, 7 i 8 originalni doprinos autora disertacije izuzimajući poglavlje 6.8 (DLP jezgro) koje predstavlja doprinos kolege, profesora doktora Vuka Vranjkovića.

1.1 Motivacija

Razvoj veštačke inteligencije (engl. *Artificial Intelligence, AI*),vezuje se za sredinu prošlog veka. Prvi predstavnici modela veštačke inteligencije pripadaju simboličkoj veštačkoj inteligenciji (engl. *Symbolic AI*). Ipak, ova oblast veštačke inteligencije je imala ograničene uspehe posebno prilikom rešavanja kompleksnih zadataka. Mašinsko učenje [1], je oblast veštačke inteligencije koja je omogućila dalji napredak, posebno od kraja 80-tih godina, a aktuelna je i danas. Posebnu ekspanziju u oblasti kompjuterske vizije [2] doživljava pojavom prve uspešne konvolucione neuronske mreže, AlexNet [3], 2012 godine. Od tada smo svedoci ubrzanog razvoja oblasti, ali i široke primene rezultata [4], [5].

Iako imaju izuzetne performanse, CNN-ovi predstavljaju modele mašinskog učenja koji zahtevaju veliku procesorsku snagu sistema na kome se izvršavaju. Na primer, za klasifikaciju jedne slike pomoću VGG-16 [6] CNN-a, hardver izvrši oko 31 milijardu operacija. Za obradu velike količine podataka pomoću CNN-ova najčešće se koriste generalna rešenja kao što su grafički procesori (engl. *Graphics Processing Unit,* GPU*)* i jedinice za procesiranje tenzora (*Tensor Processing Unit,* TPU). Nažalost, navedene hardverske platforme (GPU i TPU) nisu primenjive u sistemima u kojima su resursi ograničeni, pa tako i embeded sistemima. Da postoji potreba za izvršavanjem CNN-ova na ovakvim sistemima najbolje pokazuje činjenica da je u zadnjih nekoliko godina publikovano više od stotinu radova na temu hardverske akceleracije CNN-ova kako u ASIC tako i u FPGA tehnologijama.

Pored razvoja akceleratora specifične namene, ubrzanje izvršavanja CNN-ova se može postići na dva dodatna načina:

1. Smanjenje kompleksnosti modela.
2. Orezivanjem konvolucionih neuronskih mreža [7].

Kada se govori o smanjenju kompleksnosti modela, najčešće se misli na redukciju modela na makro nivou. Na primer, postoje CNN-ovi kao što je MobileNet v1 [8], koji su nastali sa ciljem značajnog smanjenja kompleksnosti mreže u pogledu broja operacija i slojeva, zadržavajući performanse nekih drastično složenijih modela. Ovo je jedan pristup približavanja ovih algoritama širem spektru aplikacija, ali nije jedini. Drugi najčešći je smanjenje bita koji se

koriste za reprezentaciju brojeva kao i prelazak sa sistema brojeva sa pokretnim zarezom na brojeve sa fiksnim zarezom. Aritmetičke jedinice koje obrađuju podatke u ovakvom formatu su značajno jednostavnije što ima značajan uticaj na veličinu krajnjeg akceleratora. Pored navedenih postoje i drugi načini redukcije kompleksnosti modela od kojih se orezivanje CNN-ova značajno istaklo, toliko da se može smatrati zasebnom granom.

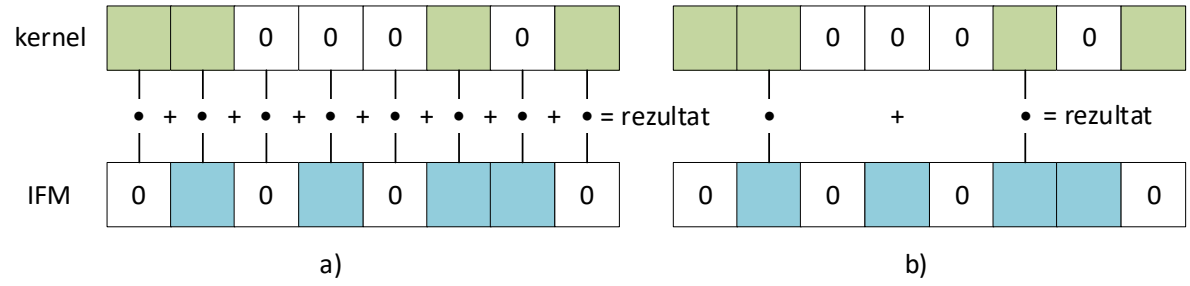
Orezivanje CNN-ova predstavlja proces u kome se veliki broj parametara kernela u konvolucionim slojevima, postavlja na vrednost nula. Drugim rečima, ovi parametri se uklanjaju. Ovim postupkom prekidaju se nepotrebne veze u CNN-ovima, a kao benefit dobijamo CNN koji ima manje parametara koji utiču na krajnji rezultat. Redukcija parametara se može iskoristiti na više načina. Najčešće se smanjenjem parametara CNN modela unapređuju performanse akceleratora, smanjuje potreba za memorijskim resursima za čuvanje parametara kernela i/ili smanjuje potrošnja energije prilikom procesiranja CNN-a.

Razvoj algoritama za orezivanje je najčešće razdvojen od razvoja arhitekture akceleratora. Ishod ovakvog pristupa optimizaciji je često arhitektura akceleratora koja ne može da iskoristi većinu prednosti koje algoritam za orezivanje pruža. Kako bi se iskoristile pogodnosti orezanih mreža, arhitekte pribegavaju usložnjavanju akceleratora te se može reći da se kompleksnost polaznog CNN-a, orezivanjem prenosi na akcelerator. To znači da bi akceleratori iskoristili sve prednosti orezanog CNN-a moraju da imaju značajno veću kompleksnost u odnosu na arhitekture koje izvršavaju neorezane CNN-ove. Da bi se umanjile mane ovakvog pristupa potrebno je uporedo razvijati arhitekturu i algoritam za orezivanje CNN-ova imajući na umu karakteristike raspoloživih hardverskih resursa na kojima će akcelerator biti implementiran.

Kao što je rečeno, osnovna prednost procesiranja orezanih mreža se ogleda u činjenici da je potrebno izvršiti značajno manji broj operacija u odnosu na obradu neorezanih CNN-ova. Jedan ilustratorni primer je prikazan na slici 1 na kojoj možemo videti dva vektora podataka nad kojima je potrebno izračunati skalarni proizvod. Skalarni proizvod se računa kao zbir svih proizvoda elemenata zelenog i plavog vektora koji se nalaze na istim pozicijama. Kvadratići u boji predstavljaju elemente različite od nula. Akcelerator koji nema mogućnost procesiranja orezanih mreža će izvršiti svih osam množenja i sedam sabiranja kako bi se izračunala konačna vrednost (Slika 1a)). Primetimo da samo dva proizvoda između elemenata vektora imaju vrednost različitu od nula. Ukoliko bi akcelerator imao mogućnost da preskoči nepotrebna množenja i sabiranja krajnji rezultat bi mogao biti značajno brže izračunat (samo dva

množenja i jedno sabiranje). Pored performansi preskakanje nepotrebnih množenja utiče i na smanjenu potrošnju energije.

Dva vektora prikazana na slici 1 mogu da predstavljaju kernel i deo ulazne mape atributa (engl. *Input Feature Map,* IFM) u konvolucioni sloj CNN-a nad kojima treba izračunati skalarni proizvod. U najvećem broju slučajeva nule u IFM-u potiču od aktivacionih funkcija prethodnih slojeva (najčešće ReLU) i njihov raspored je nepredvidiv. Većina *fine-grained* algoritama za orezivanje unosi nule u kernele, takođe, bez prethodno poznatog šablona u smislu pozicija preostalih vrednosti kernela. Navedeno znači da hardverska arhitektura koja procesira orezane CNN-ove, u opštem slučaju, ne sme da bude zavisna od rasporeda elemenata vektora koji su različiti od nula. Kompleksnost ovakvih arhitektura značajno prevazilazi arhitekture koje procesiraju elemente unapred poznatim redosledom (neorezani CNN-ovi). U opštem slučaju, arhitektura mora biti u stanju da preskoči proizvoljan broj parametara kernela i/ili IFM-a, što značajno usložnjava memorijski deo akceleratora te zahteva dodatnu logiku koja određuje koji je sledeći elemenat koji treba zatražiti. Uz navedeno, dodatni problem koji se javlja potiče od činjenice da se kerneli ne orezuju na isti način. Pošto se više kernela procesira paralelno, kako bi se postigle bolje performanse, to zahteva od akceleratora da balansira opterećenje između procesorskih elemenata. Ovakve potrebe nastaju iz nesavršenosti memorijskih podsistema koji nisu u stanju da isporuče bilo koji elemenat vektora u svakom trenutku u zavisnosti od toga da li je određenom kernelu (procesorskom elementu) potreban ili ne.

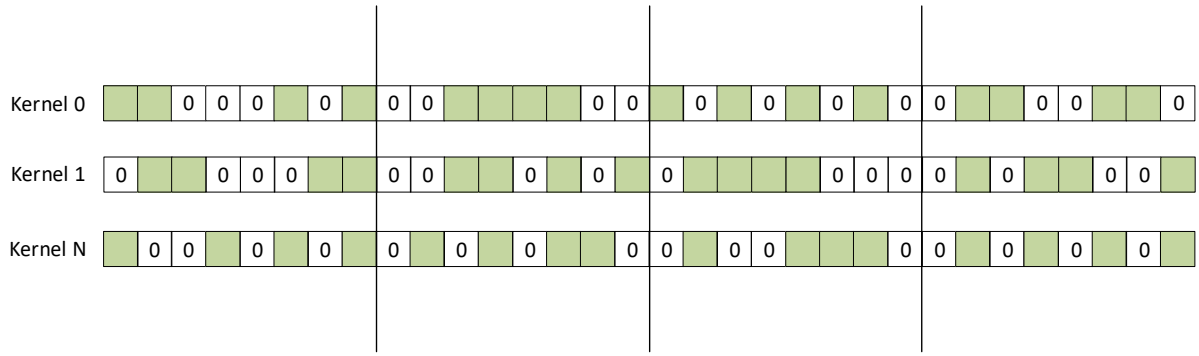


Slika 1 Procesiranje neorezane (a) i orezane mreže(b).

Navedene probleme je u velikoj meri moguće rešiti, ali na račun usložnjavanja akceleratora. Kompleksan akcelerator najčešće nije pogodan za embeded aplikacije. Pored veličine postoji i aspekt iskorišćenja raspoloživih hardverskih resursa u embeded sistemima, u kojem kompleksni akceleratori često ne ispoljavaju dobre karakteristike. Na primer, u slučajevima

FPGA baziranih akceleratora, često se dešava da logika za preskakanje nepotrebnih množenja, baferi za balansiranje opterećenja procesorskih elemenata i memorijski delovi akceleratora zauzmu skoro 100% raspoloživih LUT-ova (engl. *Look-up-table*), dok većina dostupnih DSP blokova (engl. *Digital signal processor*), ostane neiskorišćena. Nebalansirana potrošnja resursa najčešće dovodi do ograničenja prilikom skaliranja akceleratora što može biti limitirajući faktor u pogledu performansi.

Nepredvidiv raspored parametara različitih od nula se uzima kao osnovni uzrok navedenih problema koji rezultuju usložnjavanju arhitektura. Ono na šta trenutno nije poznato kako uticati je raspored nula unutar IFM-ova. Kako bi se uticaj IFM-a umanjio moguće je redukovati efikasnost arhitekture tako što ista neće biti u mogućnosti da preskače nule unutar IFM-a. Dalje, ukoliko se dodatno ograniči algoritam za orezivanje moguće je u potpunosti izbeći navedene pojave. Na algoritam je moguće uticati što implicira da je moguće uvesti šablone u pozicije parametara koji su posle orezivanja različiti od nula. Na slici 2 je prikazano jedno od ograničenja koje će biti korišćeno u nastavku kako bi se izbegla potpuna varijabilnost u pozicijama parametara od značaja.



Slika 2 Korišćeni način orezivanja kernela u nastavku.

Ukoliko se parametri kernela budu orezivali u grupama od, na primer, osam uzastopnih parametara tako da u svakoj grupi ostane unapred poznat broj parametara moguće je projektovati hardver koji je u stanju da efikasno procesira CNN uz smanjenu potrošnju resursa. Jezgro navedene ideje je prvi put prikazano u radu [9]. Ovakvim pristupom se prozor u kome se traže parametri različiti od nula ograničava na osam uzastopnih elemenata. Ovako mali prozori se mogu procesirati koristeći značajno jednostavniji hardver. Takođe, balansira se opterećenje procesorskih elemenata na nivou prozora od po osam parametara zato što svaki prozor zahteva identičan broj operacija. Sve navedeno će za rezultat imati kompaktnu

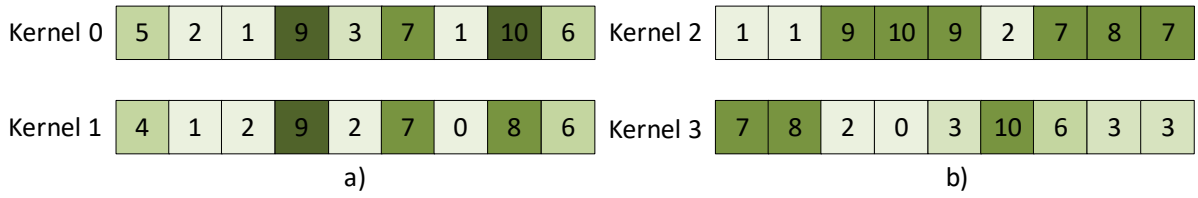
arhitekturu visokih performansi, posebno u pogledu dobijenih performansi po iskorišćenom hardverskom resursu što je od velike važnosti u embeded sistemima.

1.2 Glavni doprinosi doktorske disertacije

U doktorskoj disertaciji je predstavljen novi algoritam za orezivanje (engl. *Pruning*) konvolucionih neuronskih mreža (engl. *Convolutional Neural Networ*, CNN) kao i digitalna arhitektura, imena Argus, koja je u mogućnosti da iskoristi prednosti novog algoritma prilikom procesiranja orezanih CNN-ova. Pored orezanih, akcelerator ima mogućnost procesiranja i neorezanih, to jest, CNN-ova u originalnom obliku. Razvijeni algoritam za orezivanje posebno vodi računa o karakteristikama raspoloživih hardverskih resursa u *Field Programmable Gate Array* (FPGA) tehnologiji što znači da je arhitektura izuzetno pogodna za implementaciju na FPGA platformama. Iako je prilikom razvoja akcenat bio na FPGA tehnologiji, arhitektura je pogodana i za *Aplication Specific Integrated Circuits* (ASIC) sa nešto manjim benefitima u odnosu na poređenje sa FPGA baziranim arhitekturama. U nastavku podglave, detaljno su opisani svi navedeni doprinosi disertacije.

1.2.1 Razvoj algoritma za klasterovanja kernela unutar konvolucionih slojeva CNN-a.

Cilj algoritma za klasterovanje je da grupiše kernele unutar konvolucionih slojeva u grupe (klastere) veličine najmanje 2 ili veće. Polazna ideja je da se kerneli unutar jednog klastera orežu na isti način u pogledu pozicija parametara koje će biti orezane. Sličnost kernela unutar klastera se meri prema pozicijama parametara sa najvećim vrednostima. Slika 3a) ilustruje primer kada su kerneli slični, to jest, kada su dobri kandidati da se nađu unutar istog klastera i kada je to manje verovatan slučaj slika 3b). Primetimo da nijanse zelene boje odgovaraju amplitudi vrednosti parametara kernela (tamnije je veće).



Slika 3 Primer sličnih vektora (a) i manje sličnih u smislu pozicija velikih parametara (b).

Na slici 3 možemo primetiti da su vrednosti kernela 0 i 1 daleko sličnije po amplitudi od vrednosti kernela 2 i 3. Pošto se veći parametri kernela CNN-a smatraju važnijim za tačnost

mreže očekivano je da će manja degradacija tačnosti CNN-a biti uočljiva ukoliko kernele 0 i 1 orežemo na isti način nego ako to uradimo sa kernelima 2 i 3. Šta više, pitanje je koji kriterijum treba odabrati prilikom formiranja maske za orezivanje kernela 2 i 3 tako da se izbaci što manji broj bitnih parametara jednog i drugog kernela, a da budu orezani na isti način u smislu pozicija.

Rezultat rada algoritma za klasterovanje nije orezan CNN, već isključivo klasteri kernela koje je dalje moguće procesirati proizvoljnim algoritmom za orezivanje uz benefit da kerneli unutar svakog klastera imaju parametre sa velikim amplitudama na istim pozicijama. Izuzetni rezultati postignuti prilikom orezivanja CNN-ova su jedan od pokazatelja velike redundantnosti parametara CNN-a. Cilj klasterovanja i orezivanja kernela unutar klastera na isti način, je da se postigne balans između nivoa orezivanja i kompleksnosti procesiranja orezanih CNN-ova. Ukoliko je moguće zadržati prihvatljiv nivo orezivanja, a da se pri tome ne degradira tačnost CNN-a, ovakav pristup će doprineti smanjenoj kompleksnosti hardverskog akceleratora u pogledu potrebnih resursa. Razlog redukcije kompleksnosti akceleratora proističe iz činjenice da je potreban jedan hardverski blok koji selektuje parove parametar/ulazni atribut čiji proizvod rezultuje vrednošću različitom od nule, po klasteru kernela, a ne po kernelu. U nastavku ovaj modul će skraćeno biti referenciran kao BZPNP, to jest, blok za pronalazak nenula proizvoda. Nama je poznat jedan rad koji se bavi ovom problematikom pomoću grupisanja susednih kernela polaznog CNN modela [10]. Algoritam prikazan u [10] iziskuje analizu na nivou pojedinačnih CNN-ova kako bi se postigli što bolji rezultati orezivanja. Najčešće ova analiza uključuje i inženjera. U ovoj doktorskoj disertaciji osnovna ideja je kreiranje univerzalnog algoritama koji je nezavisan od modela CNN-a i problema koji se rešava.

1.2.2 Poboljšanje postojećeg algoritma za orezivanje CNN-ova [9] sa akcentom na FPGA platforme.

Predloženi algoritam predstavljen u radu [9] je jedan od retkih iz grupe *fine-grained* algoritama koji za izlaz ima CNN orezan tako da su pozicije parametara CNN-a različitih od nule na unapred poznatim mestima. Ovakav pristup uvodi neki vid regularnosti u orezane CNN-ove i značajno doprinosi smanjenju kompleksnosti hardverskog akceleratora koji procesira orezani CNN. Prednosti ovakvog pristupa orezivanju, u odnosu na većinu *fine-*

grained algoritama, proizilaze iz činjenice da uzastopne grupe od po, na primer osam parametara kernela, zahtevaju identičan broj MAC operacija (slika 2). Drugim rečima, ukoliko je procesorski elemenat sposoban da obradi jedanu ovakvu grupu u jednom taktu, a grupe imaju identičan broj MAC operacija, jasno je da će biti vrlo jednostavno ravnomerno uposliti procesorske elemenate bez potrebe za hardverom koji bi balansirao opterećenja. Drugi benefit je smanjena BZPNP logika što je rezultat relativno malih prozora IFM-a koji se procesiraju u svakom taktu od strane BZPNP logike. Prozori veličine, na primer, osam uzastopnih vrednosti IFM-a se u slučaju Argus arhitekture procesiraju isključivo pomoću četiri multipleksera.

Detaljnom analizom algoritma iz rada [9], ustanovljeno je da određene konfiguracije od značaja zahtevaju široke multipleksere koji nisu dostupni na modernim FPGA uređajima. Algoritam rešava problem regularnosti, to jest, pozicija parametara orezane CNN i time pojednostavljuje potreban hardver. Ipak, zahtevani široki multiplekseri koji se koriste u BZPNP blokovima poništavaju većinu ovih prednosti posebno kada se implementiraju na FPGA čipovima. Poboljšanje algoritma predstavljeno u ovoj doktorskoj disertaciji redukuje širine multipleksera sa 5-na-1 na 4-na-1 što kao rezultat ima značajno efikasnije mapiranje na veliku većinu modernih FPGA čipova. Multiplekser 5-na-1 zahteva oko 20% više logičkih kapija u ASIC implementaciji, što ne predstavlja toliko značajnu razliku kolika je prisutna prilikom FPGA implementacije. Multiplekser 5-na-1 je tačno dva puta veći od multipleksera 4-na-1 ukoliko se on mapira na 6-ulazni LUT, a ovakvi LUT-ovi su široko rasprostranjeni na modernim FPGA platformama. Kao zaključak, predstavljeno poboljšanje redukuje BZPNP dva puta. Naravno, dodatno ograničenje u procesu orezivanja će potencijalno da dovede do degradacije tačnosti orezanog CNN-a. U nastavku ove disertacije moguće je videti rezultate orezivanja u kojima je empirijski utvrđeno da za određene CNN modele predloženi način orezivanja ne narušava polaznu tačnost dok za neke je tačnost smanjena, ali u prihvatljivim granicama.

1.2.3 Sinergija poboljšanog algoritma za orezivanje i klasterovanja

Kako bi se kompletirao novi algoritam za orezivanje, potrebno je izvršiti uparivanje algoritma za klasterovanje i poboljšanog algoritma za orezivanje. Algoritam prvo klasteruje kernele unutar svih konvolucionih i FC slojeva, da bi potom inkrementalno orezao ciljani CNN. U konkretnom slučaju, inkrementalno orezivanje znači da će u svakom prolazu algoritma biti uklonjeno dodatnih 12.5% nepotrebnih parametara kernela, dok se ne ore̊že ukupno 50%

polaznih parametara. Između prolaza, orezani CNN će biti ponovo treniran kako bi se povratila prvobitna tačnost neorezanog modela.

Uspešno klasterovanje će rezultovati u smanjenju broja BZPNP blokova najmanje dva puta (klasteri veličine 2 kernela). Poboljšanje postojećeg algoritma za orezivanje CNN-ova [9], takođe, redukuje internu logiku BZPNP blokova dva puta.

1.2.4 Razvoj hardverske arhitekture koja je u mogućnosti da akcelerira kako neorezane tako i mreže orezane razvijenim algoritmom.

Kako bi se postigla visoka efikasnost, arhitektura je podeljena na više specijalizovanih blokova. Najkompleksniji blok predstavlja konvoluciono jezgro (engl. *Conv Core*, CC), koje je usko specijalizovano za procesiranje konvolucionih i FC slojeva. Pored CC modula postoji i blok za procesiranje ostalih slojeva kao što su *Polling* i *Adding*.

Zbog vremenske kompleksnosti konvolucionih slojeva akcenat disertacije je na optimizaciji CC modula. Svaki CC čine 32 procesorska elementa (engl. *Processing Element*, PE) koji mogu paralelno da procesiraju do 32 kernela istog konvolucionog sloja. Kako bi akcelerator bio kompletan CC blok ima mogućnost da pored CNN-ova orezanih razvijenim algoritmom procesira i neorezane mreže. Ukoliko su mreže orezane nekim drugim algoritmom, arhitektura neće biti u mogućnosti da iskoristi prednosti orezane mreže, ali će biti u stanju da je procesira ukoliko se CNN bude posmatrao kao neorezana mreža. Jedan CC predstavlja izuzetno kompaktno jezgro i kao takavog ga je moguće instancirati na gotovo svakom FPGA kolu. Sa druge strane, performanse jednog CC modula mogu biti nedovoljne u slučaju procesiranja kompleksnih CNN-ova. Ukoliko su potrebne bolje performanse, više CC modula može da se poveže preko projektovane konekcije (engl. *Link*) za umrežavanje CC-ova. *Link* modul doprinosi povećanju faktora paralelizacije, ali i smanjenju potreba za propusnim kapacitetom DRAM kontrolera ukoliko sloj čine više od 32 kernela. Ovim se postiže visoka skalabilnost arhitekture. Pored skalabilnosti, CC blok odlikuje i visok stepen univerzalnosti u pogledu procesiranja konvolucionih slojeva. CC modul je projektovan tako da ima izuzetan stepen uposlenosti PE-ova neovisno od veličine IFM-a, kernela, *stride*-a i ostalih parametara konvolucionih slojeva.

Pored PE-ova CC modul sadrži i blok koji je odgovoran za učitavanje parametara mreže i keširanje IFM-a, generišući ulazni tok podataka (engl. *Input Stream*, Istream) za PE-ove.

Prilikom testiranja na FPGA platformama, pokazalo se da je kvalitet keširanja značajan gotovo isto kao i arhitektura PE-ova i njihovog povezivanja. Keširanjem je, gotovo u potpunosti, moguće maskirati jednu od najvećih mana embeded sistema, a to je smanjena propusna moć DRAM kontrolera u odnosu na kompleksnije sisteme. Osnovna ideja korišćena u ovoj doktorskoj disertaciji je publikovana u radu [11], ali je ona nadograđena poboljšanjem prezentovanim u radu [12].

2 Mašinsko učenje

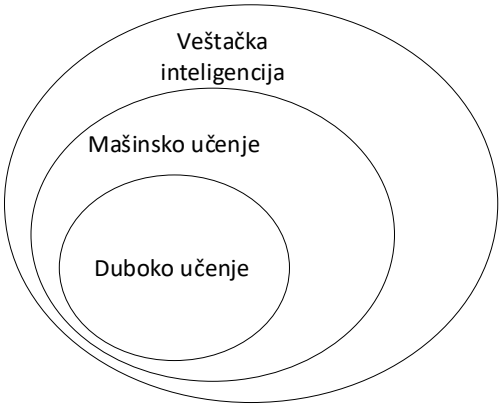
Mašinsko učenje je oblast veštačke inteligencije koja beleži snažan razvoj od 90-tih godina prošlog veka. Postoji mnoštvo definicija mašinskog učenja te su u nastavku pobrojane neke od njih:

1. Mašinsko učenje je proces u kome kompjuter modifikuje svoju akciju tako da akcije postaju sve tačnije, ukoliko tačnost predstavlja odstupanje preduzete od željene akcije [1].
2. Mašinsko učenje je grana veštačke inteligencije koja sistematično primenjuje algoritme kako bi se izvukle informacije iz raspoloživih podataka [13].
3. ,,Mašinsko učenje je skup metoda za konstrukciju matematičkih modela koji mogu automatski da prepoznaju šablone u podacima“ [14].

Prema [1], algoritme mašinskog učenja možemo podeliti i prema načinu na koji kreiraju model:

1. Nadgledano učenje (engl. *Supervised learning*) – Algoritam pokušava da generalizuje odziv na osnovu trening seta koji sadrži primere ulaznih podataka zajedno sa tačnim odzivima.
2. Nenadgledano učenje (engl. *Unsupervised learning*) – Algoritam nema na raspolaganju tačne odzive već samo primere. U procesu učenja pokušava da nađe sličnost između ulaznih podataka i kreira model koji može da kategoriše ulazne podatke prema pripadnosti nekoj grupi.
3. Pojačano učenje (engl. *Reinforcement learning*) – Algoritam dobija povratnu informaciju o tome da li je odgovor pogrešan ili tačan, ali mu se ne govori na koji način da ispravi predviđanje. Algoritam mora samostalno da istraži različite mogućnosti dokle god ne pronađe adekvatan način da daje tačne odgovore.
4. Evolutivno učenje (engl. *Evolutionary learning*) – Biološka evolucija se može posmatrati kao proces učenja. Ogranizmi se prilagođavaju kako bi popravili svoju šansu za preživljavanjem i time šansu za ostavljanjem potomaka u određenom okruženju.

Najzastupljeniji tip učenja je nadgledano učenje koje je u fokusu ostatka doktorske disertacije. Tačnije, podoblast mašinskog učenja, takozvano, duboko učenje (engl. *Deep Learning* [15]). Slika 4 prikazuje odnos veštačke inteligencije prema mašinskom i dubokom učenju.



Slika 4 Veštačka inteligencija i odnos prema mašinskom, odnosno, dubokom učenju.

Umesto da se modeli formiraju kao skup ručno kreiranih pravila (*Symbolic AI*), to jest, da se pravila kreiraju kao u klasičnom programiranju, algoritmi mašinskog učenja kao ulaz koriste podatke i odgovore da bi kreirali skup pravila. Po završetku učenja, modelima se na ulaz dovode novi podaci koje oni transformišu u odgovarajuće reprezentacije odgovora koje možemo lako dekodovati.

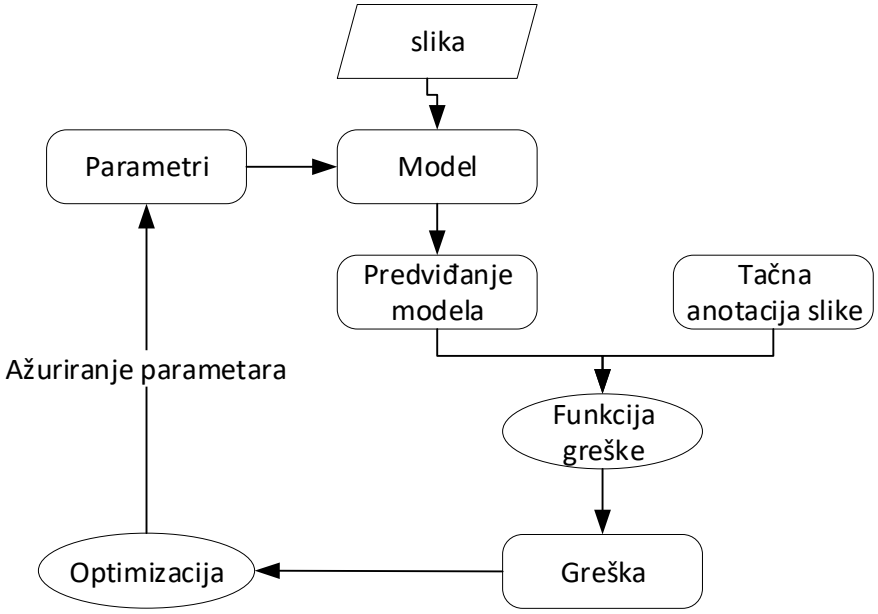
Zamislimo da želimo kreirati sistem koji uspešno razvrstava slike u dve grupe u zavisnosti od toga da li se na slikama nalaze psi ili mačke.

Da bismo kreirali skup pravila pomoću nekog od algoritama mašinskog učenja potrebni su:

- Ulazni podaci (slike na kojima su psi odnosno mačke).
- Anotacije određenog broja slika (slike iz skupa nad kojima će model učiti).
- Funkcija greške koja pokazuje koliko dobro (loše) model klasifikuje slike.

Proces učenja je prikazan na slici 5. Za rešavanje određenog problema potrebno je odrediti koji model mašinskog učenja ćemo koristiti i postaviti početne vrednosti parametara modela. Neka to u ovom primeru bude neuronska mreža. Parametri ove mreže se najčešće inicijalizuju na vrednosti koje se generišu na slučajan način i pri tome nisu velike. U procesu učenja, model transformise ulazne podatke (sliku) u odgovarajuću reprezentaciju koja predstavlja predviđanje modela. Predviđanje se poredi sa tačnom anotacijom slike i propušta kroz funkciju koja vraća grešku za trenutni primer. Greška se prosleđuje procesu koji vrši

optimizaciju parametara mreže koristeći algoritam propagacije greške kroz slojeve neuronske mreže unazad. Kao rezultat optimizacije dobijamo izmenjene parametre modela. Ovako izmenjeni parametri najčešće rezultuju kvalitetnijim modelom u pogledu predviđanja.



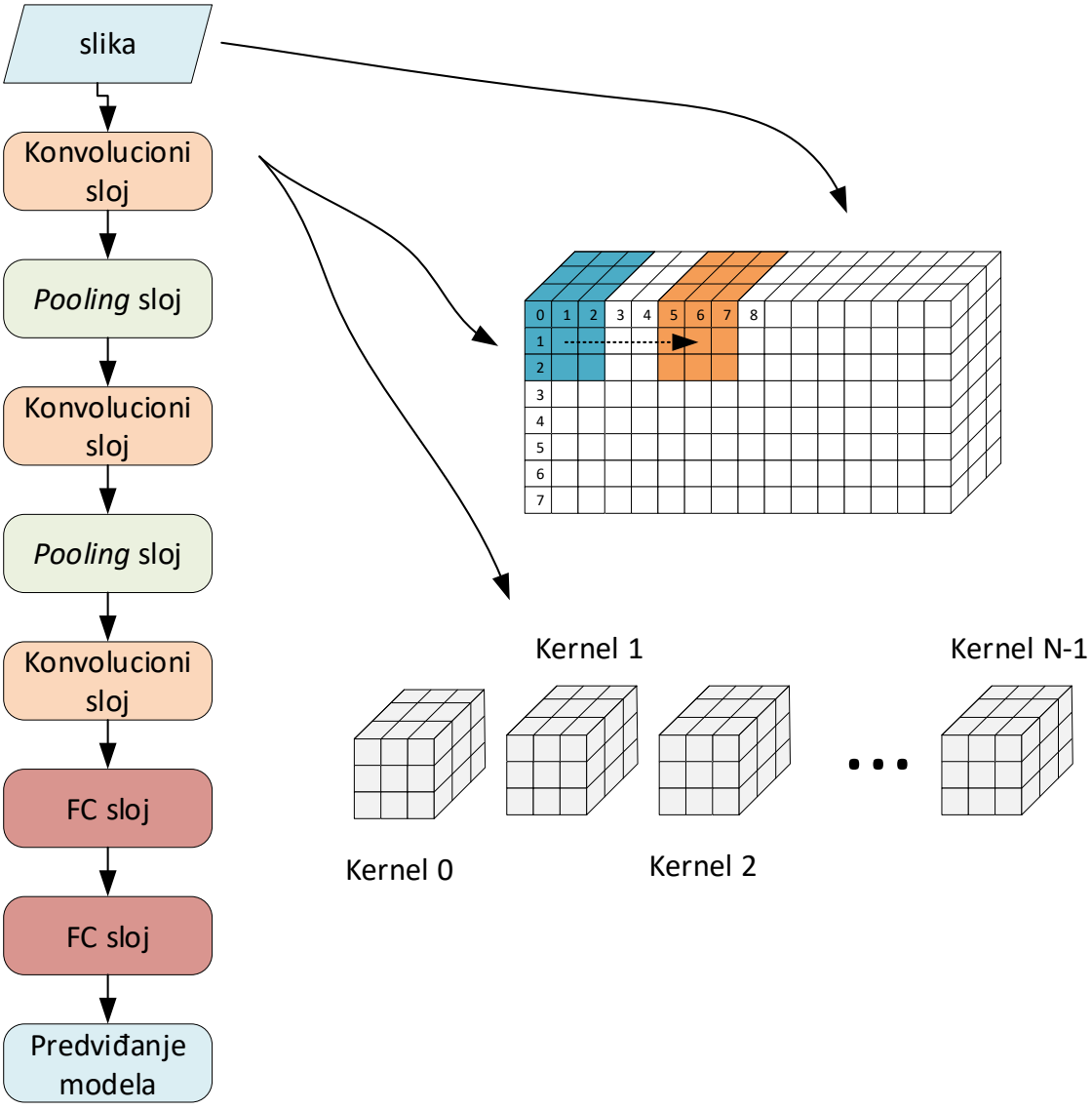
Slika 5 Proces treninga modela mašinskog učenja.

2.1.1 Konvolucione neuronske mreže

Kao što je rečeno, početak intenzivanog razvoja dubokog učenja, konkretno jednog od modela ovakvog učenja, konvolucionih neuronaskih mreža, se vezuje za otkriće AlexNet CNN-a 2012. godine. Iako su teorijski koreni ovakvog pristupa klasifikaciji slika postavljeni još krajem prošlog veka [16], nedostatak snažnih hardverskih resursa su sprečavali njegovu široku primenu. Od 2012, svaki naredni pobednik takmičenja u klasifikaciji slika [17] bio je model koji spada u CNN-ove. Slika 6 prikazuje primer konvolucione neuronske mreže koja može da se koristi za jednostavne zadatke, na primer, klasifikacija cifara pisanih rukom. Pored konvolucionih slojeva, CNN-ove čine i *Pooling* slojevi, potpuno povezani (engl. *Fully-connected*, FC), slojevi sabiranja (engl. *Adding layer*), *Batch Normalization* [18] itd.

Konvolucioni sloj čine kerneli koji su najčešće 3D oblika veličine 3x3xdubina gde je dubina najčešće jednaka dubini ulazne mape atributa. Prvi konvolucioni sloj vrši obradu nad slikom, i ukoliko je slika u RGB formatu, to jest, dubine tri, onda i kerneli prvog sloja imaju dubinu tri. Na slici 6 je prikazana konvolucija u prvom sloju CNN-a čija ulazna slika ima navedene karakteristike. Izlazna vrednost za svaki položaj kernela u odnosu na ulaznu mapu atributa

(IFM) se računa kao skalarni proizvod kernela i odgovarajućeg prozora IFM-a. Izračunavanje kompletne izlazne mape atributa (engl. *Output Feature Map*, OFM) se vrši tako što kerneli ,,klize“ po IFM-u. Na slici 6 su naznačena dva prozora IFM-a koja učestvuju u kreiranju tačaka OFM-a na dve pozicije. Prvi, označen plavom bojom i šesti prikazan narandžastom. Primetimo da je u ovom slučaju korak (engl. *stride*) jednak jedan. Ovo je najčešća vrednost koraka mada postoje slojevi koji imaju veći korak. Veći korak će rezultovati redukovanom veličinom OFM-om. Ipak, u većini slučajeva redukcija izlazne mape se postiže primenom *pooling* slojeva.



Slika 6 Primer jednostavne konvolucione neuronske mreže.

U nastavku su navedene dve ključne razlike konvolucionih slojeva u odnosu na FC slojeve.

1. Naučeni obrasci koje uče kerneli su invariantni na translaciju. Da bi FC slojevi bili u stanju da uoče određeni šablon (engl. *feature*) na svim delovima IFM-a, trening skup mora da ima takav šablon na svim delovima IFM-a. Sa druge strane, konvolucionim slojevima je dovoljno da u bilo kom delu ulazne mape uoče određeni šablon i to će biti dovoljno da ga mogu prepoznati na bilo kom delu IFM-a.
2. Uspešno uče veze između šablona na različitim nivoima hijerarhije. Početni slojevi CNN-ova obično uočavaju jednostavnije šablone kao što su ivice, dok viši slojevi CNN-a najčešće od jednostavnijih pokušavaju da formiraju kompleksnije obrasce.

Obrazac po kom se računa dvodimenzionalna konvolucija je prikazan izrazom 1.

$$(m * n)_{ij} = \sum_{k=0}^{c-1} \sum_{l=0}^{d-1} m_{i-k,i-l} g_{k,l}. \quad (1)$$

Obrazac 1 predstavlja 2D konvoluciju između matrica m i n, čije su dimenzije $a\ x\ b$ i $c\ x\ d$ respektivno.

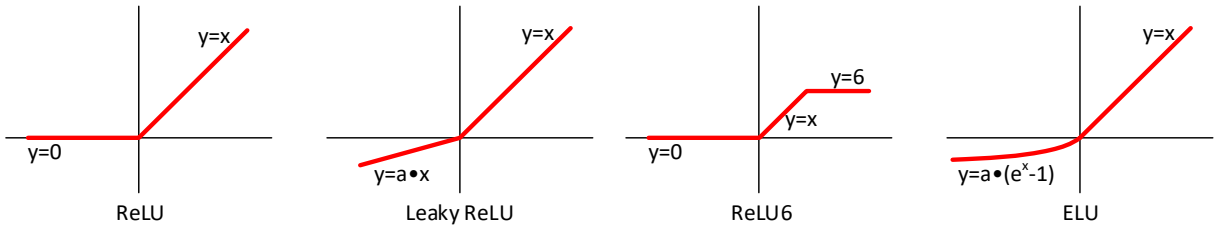
U pogledu broja operacija koje se izvode prilikom računanja odziva, konvolucioni slojevi predstavljaju najzahtevnije od svih slojeva unutar CNN-ova. Vreme provedeno u računanju odziva konvolucionih slojeva ima i do 90% udela u vremenu izračunavanja odziva kompletnog CNN-a [16], [19]. U tabeli 1 su izlistani brojevi *multiply-accumulate* (MAC) operacija po konvolucionim slojevima VGG16 CNN-a. Navedeni podaci jasno pokazuju da je prilikom razvoja arhitekture i algoritma za orezivanje od suštinskog značaja fokus staviti na optimizaciju izvršavanja konvolucionih slojeva. Pored navedenih slojeva u tabeli 1, VGG 16 sadrži i 5 *max pooling* slojeva čije je vreme izvršavanja, to jest, broj operacija zanemarljiv u poređenju sa konvolucionim slojevima. Takođe, postoji mogućnost da se napravi arhitektura koja je u mogućnosti da većinu *pooling* slojeva izračuna zajedno sa konvolucionim. Ovakvim rešenjem se u potpunosti eliminiše vreme potrebno za izračunavanje *pooling* slojeva.

VGG-16 sloj	**Broj MAC operacija po sloju:**
konvolucioni sloj 1	86.704.128
konvolucioni sloj 2	1.849.688.064
konvolucioni sloj 3	924.844.032
konvolucioni sloj 4	1.849.688.064
konvolucioni sloj 5	924.844.032
konvolucioni sloj 6	1.849.688.064
konvolucioni sloj 7	1.849.688.064

konvolucioni sloj 8	924.844.032
konvolucioni sloj 9	1.849.688.064
konvolucioni sloj 10	1.849.688.064
konvolucioni sloj 11	462.422.016
konvolucioni sloj 12	462.422.016
konvolucioni sloj 13	462.422.016
FC sloj 1	102.760.448
FC sloj 2	16.777.216
FC sloj 3	409.600
Ukupno	15.470.264.320

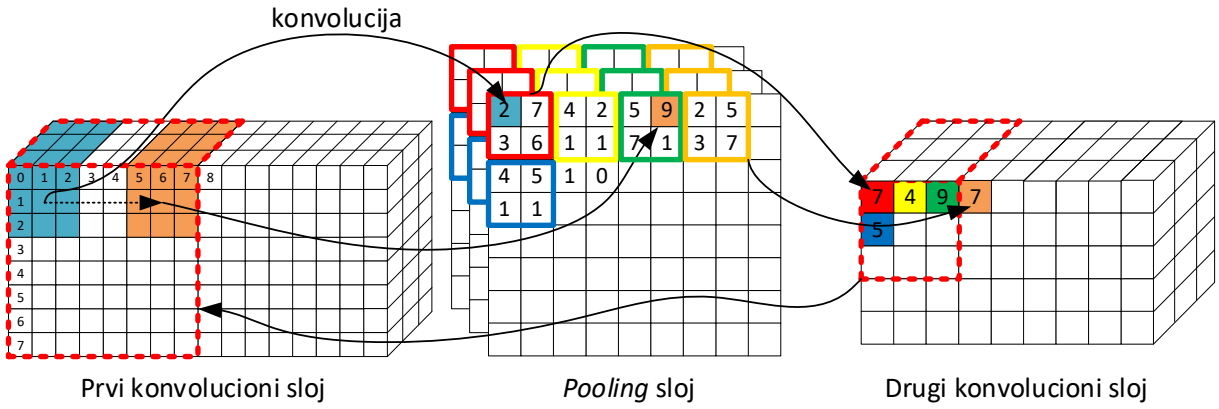
Tabela 1 Broj MAC operacija po konvolucionim i potpuno povezanim slojevima VGG16 CNN.

Prilikom izračunavanja OFM-a konvolucionih slojeva, rezultati pojedinačnih skalarnih proizvoda se najčešće prosleđuju aktivacionim funkcijama koje vrše nelinearni obradu nad njima. Najčešće je ta funkcija ReLU (engl. *rectified linear* – ReLU). Pored navedene u široj primeni se mogu naći: *Leaky* ReLU, *exponential linear* (ELU) [20] kao i ReLU6 (Slika 7).



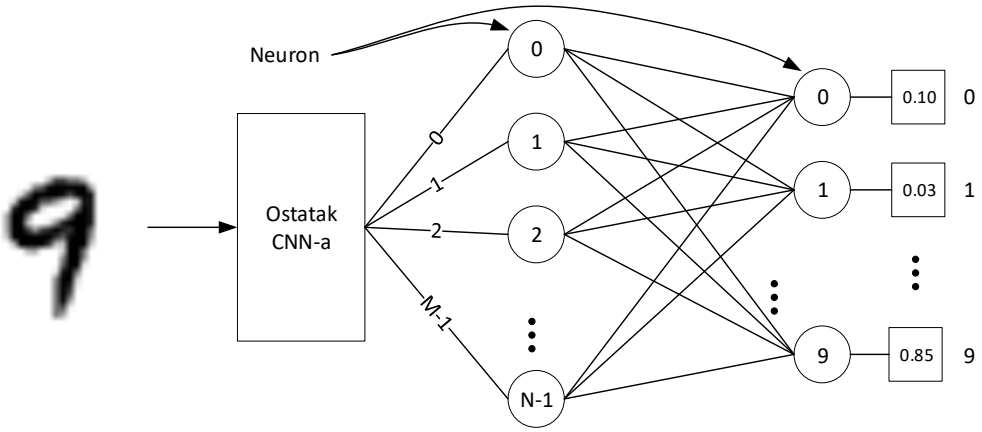
Slika 7 Prikaz najčešće korišćenih aktivacionih funkcija u konvolucionim slojevima.

Pooling sloj se najčešće nalazi između konvolucionih slojeva ili blokova konvolucionih slojeva. Cilj *pooling*-a je da agresivno smanji ulaznu mapu i to najčešće četiri puta. Za razliku od konvolucionih slojeva u kojima se obično koriste kerneli veličine 3x3, *pooling* slojevi procesiraju prozore veličine 2x2 sa *stride*-om 2. Najzastupljeniji *pooling* sloj je onaj koji izdvaja maksimalne vrednosti u prozorima veličine 2x2 (engl. *Maximum pooling*) i propušta ih ka sledećim slojevima CNN-a. Uloga *pooling* slojeva je da smanji broj atributa mapa koje treba da procesiraju naredni slojevi i da omogući narednim slojevima da obrađuju sve veće segmente ulaznih mapa predhodnih slojeva kao što je to prikazano na slici 8. Na slici su prikazana dva konvoluciona sloja sa *pooling* slojem između. Crveni isprekidani kvadrat na IFM-u prvog konvolucionog sloja označava deo IFM-a veličine 8x8 koji će u narednom konvolucionom sloju biti procesiran kernelom veličine 3x3. Ovakvo smanjenje IFM-a je postignuto *pooling* slojem koji se nalazi između dva konvoluciona sloja.



Slika 8 Primer smanjivanja izlaznih mapa konvolucionih slojeva pomoću pooling sloja.

Potpuno povezan sloj se obično nalazi na izlazu CNN-a i koristi se za određivanje krajnje pripadnosti nekog elementa određenoj klasi. Neuroni FC sloja za ulaze dobijaju sve izlaze prethodnog sloja i računaju skalarni proizvod ulaznog vektora i vektora parametara (težina, engl. *weights*) neurona. Vektor parametara neurona predstavlja naučene koeficijente neurona koji se u opštem slučaju razlikuju od parametara drugih neurona u istom sloju. Na slici 9 su prikazana poslednja dva sloja CNN-a sa slike 6.



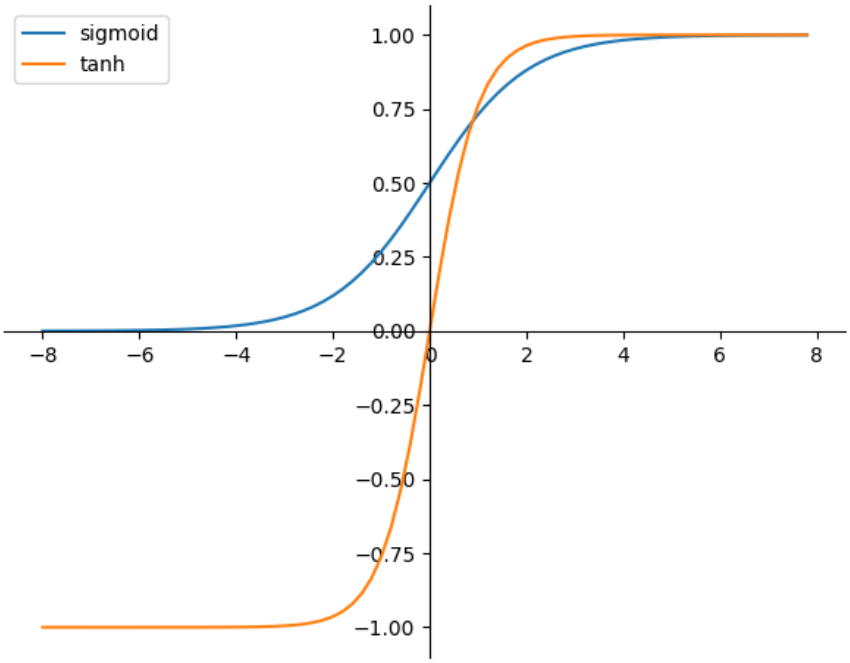
Slika 9 Detaljni prikaz potpuno povezanih slojeva CNN-a za prepoznavanje cifara.

Izlaz pojedinačnog neurona prvog FC sloja možemo računati pomoću izraza 2:

$$y_i = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_{M-1} x_{M-1}). \qquad (2)$$

Naučeni parametri neurona su predstavljeni vektorom w čija je veličina jednaka broju ulaznih podataka M. Funkcija σ se zove aktivaciona funkcija i spada u grupu nelinearnih funkcija. Za razliku od konvolucionih slojeva, u kojima dominira ReLU funkcija, FC slojevi koriste i druge,

kao što su normalizovana eksponencijalna funkcija (engl. *softmax*). *Softmax* funkcija se gotovo uvek koristi za poslednje slojeve čiji izlaz predstavlja verovatnoću da ulazna slika pripada određenoj klasi. Pored *softmax* funkcije za skrivene slojeve su se često koristile i sigmoidna funkcija i hiperbolički tangens. Svaka od navedenih aktivacionih funkcija ima svoje prednosti i mane. Ne može se reći koja je funkcija univerzalno najbolja zato što različiti problemi često zahtevaju različite aktivacione funkcije. Odabir aktivacione funkcije kod neuronskih mreža najviše utiče na brzinu učenja, a ne na potencijal mreže da nešto nauči. Na slici 10 su prikazane *sigmoid* i *tanh* funkcije.



Slika 10 Aktivacione funkcije FC slojeva (sigmoid i tanh).

Broj neurona u prvom FC sloju (N) je proizvoljan i najčešće se empirijski određuje. Jedan od načina je da se N povećava do trenutka kada mreža počinje lako da uči ,,napamet“ (engl. *overfitting*) primere iz trening skupa. Sa druge strane, broj neurona u poslednjem sloju je određen samim problemom. U slučaju prepoznavanja cifara, njihov broj je jednak 10. Svaki od 10 izlaza neurona predstavlja verovatnoću da ulazna slika pripada jednoj od 10 mogućih cifara (pod uslovom da se koristi *Softmax* aktivaciona funkcija).

U poređenju sa konvolucionim slojevima, FC slojevi zahtevaju značajno manji broj operacija prilikom procesiranja. Sa druge strane, FC slojevi su obično daleko zahtevniji u pogledu propusne moći memorijskog podsistema. Oba problema je moguće rešiti orezivanjem CNN-

ova, to jest, smanjenjem broja korisnih parametara CNN-a. U oba slučaju doćiće do smanjenja ukupnog broja potrebnih operacija da se procesira sloj, dok će se u slučaju FC slojeva značajno smanjiti i količina podataka koja se razmenjuje između memorijskih i procesorskih jedinica. Značajno redukovanje transfera podataka u FC slojevima proizilazi iz činjenice da jedan neuron daje samo jedan rezultat u OFM-u FC sloja. To znači da nema višestrukog korišćenja parametara neurona kao što je slučaj sa kernelima u konvolucionim slojevima. Na kraju, može se reći da nivo orezivanja parametara unutar FC slojeva direktno odgovara smanjenju razmene podataka između eksternih memorijskih jedinica i akceleratora. Primetimo da pređašnja analiza podrazumeva da se obrađuje jedna ulazna instanca (slika) u svakom trenutnku što znači da je veličina *batch*-a jednaka 1. Ovo je slučaj većine embeded akceleratora kod kojih je odziv (engl. *latency*) bitniji od propusne moću (engl. *throughput*).

2.2 Aktuelno stanje u oblasti

Od pojave AlexNet-a svedoci smo velikog razvoja dubokog učenja kako sa algoritamskog tako i sa stanovišta hardverskih akceleratora sposobnih da procesiraju sve kompleksnije CNN-ove. Iako se broj parametara CNN-a uglavnom smanjuje ili održava na istom nivou (VGG-16 [6] ima 138 miliona, Inception v3 [21] oko 23 miliona) novi tipovi slojeva dodaju novi nivo kompleksnosti CNN-ovima. Mreže čine sve raznovrsniji slojevi i konfiguracije istih kao što su različiti koraci prozora u konvolucionim mrežama ili oblici kernela koji više nisu dimenzija 3x3. Da bi se kompleksnost redukovala pristupilo se orezivanju CNN-ova. Orezivanje se pokazalo izuzetno uspešnim, do te mere da za određene mreže može da se ukloni i do 90% prvobitnih parametara. Uklanjanje se vrši anuliranjem nepotrebnih parametara što dovodi do toga da je moguće preskočiti MAC operaciju za određeni parametar unutar konvolucionog i FC sloja. Iako je stepen orezivanja prilično veliki, raspored preostalih parametara je takav da je gotovo nemoguće pronaći šablon u njihovim pozicijama. Raspored parametara različitih od nule predstavlja dodatnu komponentu koja otežava procesiranje do te mere da vreme izvršavanja orezanih mreža na hardverimu opšte namene, kao što su GPU i TPU, često prevazilazi vreme procesiranja neorezanih CNN. Pored pomenutih GPU i TPU, koji se najčešće koriste u velikim procesorskim centrima, za procesiranje CNN-ova postaju sve popularnije specijalizovane hardverske arhitekture implementirane kako na FPGA kolima tako i u ASIC tehnologijama. Zbog visokog nivoa tačnosti CNN-ova, pojavljuje se sve veća potreba za njihovim korišćenjem u embeded sistemima. Pored superiorinih performansi u odnosu na druge algoritme, razvoj

velikog broja embeded akceleratora specijalizovanih za CNN je podstaknut i razvojem izuzetno fleksibilnih FPGA baziranih sistema na čipu (engl. *System-on-Chip*, SoC). Potreba za akceleracijom CNN-ova se najbolje ogleda u činjenici da je u zadnjih nekoliko godina objavljeno više od 100 radova na ovu temu kako ASIC tako i FPGA baziranih arhitektura. Neke od uspešnih arhitektura kao što je Eyeris v1 [22], i v2 [23] su evoluirale zajedno sa novim arhitekturama CNN-ova kako bi ispratile pomenutu kompleksnost koju su uneli novi tipovi slojeva i algoritama za orezivanje CNN-ova. Obe verzije Eyeris akceleratora zajedno sa akceleratorima kao što su Cambricon-x [24], NullHop [25], DaDianNao [26], SparseNN [27], ENVISION [28], Thinker [29], UNPU [30] spadaju u arhitekture koje su realizovanje u ASIC tehnologiji. Pored ASIC rešenja postoji i veliki broj FPGA baziranih akceleratora kao što su Caffeine [31], Snowflake [32], NEURAghe [33], FpgaConvNet [34], CoNNa [35], NullHop (FPGA verzija) [25], i akceleratori predstavljeni u radovima [36], [37] i [38].

Od navedenih arhitektura NullHop i DaDianNao imaju mogućnost efikasnog procesiranja CNN-ova u smislu da mogu da preskoče operacije za argumente IFM-ova koji imaju vrednost nula. Zbog slučajne distribucije rasporeda korisnih argumenata i njihove zavisnosti od ulazne slike ovakve arhitekture imaju teško predvidivo vreme procesiranja svake ulazne instance. Količina argumenata različitih od nula je promenljiva od ulaza do ulaza. Sa druge strane, Sparse NN i Cambricon-x su takvi da mogu da preskoče nepotrebne operacije u slučaju da su parametri CNN-a jednaki nuli, to jest, u stanju su da procesiraju orezane CNN-ove. CoNNa je redak predstavnik koji ima mogućnost da preskoči nepotrebna izračunavanja kako u pogledu nula IFM-ova tako i u slučaju nula u parametrima CNN-a. Imajući u vidu navedeno, akcelerator koji se predlaže u ovoj doktorskoj disertaciji neće biti prvi koji koristi benefite orezanih CNN-ova. Ipak, nama nije poznat niti jedan akcelerator koji ima mogućnost procesiranja CNN-ova orezanih algoritmom koji je deo ove disertacije, a koji je posebno razvijen za FPGA platforme sa skromnim raspoloživim resursima. U nastavku sledi pregled navedenih arhitektura u kome su predstavljene njihove osnovne karakteristike.

DaDianNao [26] je arhitektura koja pripada ranoj fazi razvoja CNN akceleratora čiji je dizajn inspirisan prvim uspešnim CNN-ovima kao što su VGG i AlexNet. Akcelrator koristi velike memorijske banke veličine nekoliko desetina MB. Zbog velikih zahteva u pogledu količine memorije ovakav pristup nije pogodan za današnje embeded/FPGA sisteme.

Thinker [29] kao i DaDianNao pripada ASIC grupi CNN akceleratora. Izuzetno je fleksibilan sa podrškom različitoj preciznosti aritmetike, 8-bitnoj i 16-bitnoj. U 8-bitnoj konfiguraciji broji 1024 MAC jedinice. Iako ima mogućnost da procesira većinu poznatih slojeva CNN-ova, mana ove arhitekture se ogleda u nemogućnosti da procesira orezane mreže te tako njegova efikasnost nije na nivou arhitektura koje to mogu.

UNPU [30] je još jedan predstavnik grupe ASIC akceleratora sa nama najvećom poznatom fleksibilnošću u pogledu podrške različitim preciznostima aritmetičkih jedinica. Naime, arhitektura je u stanju da procesira CNN-ove čija je preciznost parametara u rasponu od 1 do 16 bita. Zbog svoje jedinstvene arhitekture, izuzetno je teško napraviti pravedno poređenje sa ostalim akceleratorima te je ovde samo pomenut zbog svoje specifičnosti.

NullHop [25] spada u grupu pionira akceleratora koji mogu da procesiraju kompresovane ulazne mape (nula vrednosti u IFM-u su izostavljene). Kao produkt procesiranja NullHop isporučuje takođe kompresovanu izlaznu mapu argumenata svakog sloja. Ovako kompresovana mapa se smešta na memorijske module izvan čipa. Pošto se smeštaju samo potrebni argumenti (bez argumenata čija je vrednost 0) postignuta je ušteda u potrošnji energije kao i smanjena potreba za velikim propusnim kapacitetima memorijskih magistrala. Ipak, širina magistrala je samo 32-bita što značajno degradira performanse akceleratora čak i ako procesira kompresovane tokove ulaznih odnosno izlaznih mapa. Autori su prijavili da je NullHop implementiran na FPGA platformi u stanju da obradi samo 0.441 slika u sekundi kada ih procesira pomoću VGG-16 CNN-a. Pored skromnih performansi, dodatnu prepreku u primeni NullHop arhitekture na pristupačnim FPGA platformama predstavlja činjenica da je za implementaciju potrebno 229k LUT-ova što prevazilazi veličine FPGA platformi, ulaznog nivoa veličine, i do 3 puta.

Cambricon-x [24] kao i akcelerator predložen u nastavku disertacije ima mogućnost preskakanja MAC operacija u slučaju da su neki parametri kernela CNN-a jednaki nuli. Svoju efikasnost duguje izuzetno velikim multiplekserima koji imaju 256 ulaza i kao takvi su izuzetno nepraktični za implementaciju na FPGA kolima. Ovako veliki multiplekseri bi zauzeli veliki broj LUT-ova na FPGA čipovima i time napravili izuzetno nebalansiranu potrošnju ograničenih resursa na čipu. Ovaj odnos bi bio izuzetno loš u pogledu potrošnje LUT-ova prema DSP blokovima koji su osnova za izračunavanje MAC operacija. Krajnji ishod bi bio takav da bi velika

količina DSP blokova ostala neiskorišćenja, odnosno, da bi pun potencijal raspoloživog FPGA čipa ostao neiskorišćen.

Eyeris v2 [23] predstavlja naprednu verziju prvobitnog Eyeris-a sa akcentom na podršci procesiranja orezanih i kompaktnih CNN-ova kao što je MobileNet [8]. Implementirani akcelerator spada u grupu najsnažnijih arhitektura u pogledu performansi, ali sa manom da zahteva izuzetno veliku propusnu moć memorijskog sistema izvan čipa. Autori predviđaju do 24% lošije performanse od prijavljenih u slučaju da je dostupan memorijski interfejs propusne moći od 25GB/s. Ovako moćni memorijski kontroleri nisu dostupni na većini današnjih embeded sistemima.

ENVISION [28] i **Snowflake** [32] su RISC bazirane arhitekture implementirane kao ASIC odnosno, FPGA bazirani akcelerator. ENVISION sadrži SIMD MAC strukturu veličine 16x16 pojedinačnih MAC jedinica, ukupno 256. Aritmetička preciznost je konfigurabilna i može biti 4, 8 i 16-bita. Snowflake čine 4 klastera sa ukupno 256 MAC jedinica koje rade na frekvenciji od 250 MHz. Broj MAC jedinica ove dve arhitekture (256) predstavlja količinu MAC blokova koji su raspoloživi na većini pristupačnih FPGA platformi. Identičan broj MAC jedinica zauzima i arhitektura predložena u nastavku u konfiguraciji sa četiri konvoluciona jezgra što daje mogućnost vrlo pravednog poređenja sa ovim arhitekturama. Takođe, ovo poređenje bi moglo da odgovori na pitanje koliko procesiranje orezanih mreža doprinosi performansama akceleratora jer ENVISION i Snowflake nemaju mogućnost procesiranja orezanih mreža.

Caffeine [31] nije samo akcelerator, već automatizovani kompajler čiji je ulaz Caffe [31] model CNN-a, a izlaz arhitektura imlementirana pomoću jezika za opis hardvera na visokom nivou (engl. *High Level Synthesis*, HLS). Ovakav pristup pruža izuzetnu fleksibilnost krajnjem korisniku koji ne mora imati znanja iz oblasti hardverske implementacije kako bi ga koristio. Mana HLS pristupa se još uvek ogleda u neefikasnim implementacijama u pogledu maksimizacije iskorišćenja raspoloživih performansi FPGA čipova. Poređenje Argus arhitekture sa Caffeine akceleratorom bi moglo da pruži uvid o trenutnom stupnju razvoja HLS kompajlera. Ova komparacija može da indikuje kolika je stvarna razlika u efikasnosti implementacija pomoću HLS i RTL (engl. *Register Transfer Level*) pristupa, koji se smatra nižim u smislu nivoa apstrakcije.

CoNNa [35] spada u grupu arhitektura koje mogu efikasno da procesiraju kako nula argumente u ulaznim mapama tako i nula parametre u kernelima i neuronima FC slojeva. Još jedna specifičnost se ogleda u činjenici da CoNNa podržava procesiranje CNN-ova orezanih proizvoljnim algoritmom. Od svih navedenih arhitektura, CoNNa pokazuje najbolju efikasnost kada se performanse skaliraju prema broju iskorišćenih MAC jedinica. Ipak, visoku efikasnost i podršku procesiranju CNN-ova orezanih proizvoljnim algoritmima plaća nešto većim zahtevima za LUT elementima na FPGA čipovima. Ishod je donekle nebalansirano iskorišćenje dostupnog hardvera na FPGA čipovima. Poseban algoritam za orezivanje predstavljen u nastavku disertacije prevazilazi ove mane, bez prevelike degradacije u pogledu performansi između Argus akceleratora i CoNNa-e. Svakako, Argus arhitektura ima manju efikasnost kada se posmatraju performanse po jednoj MAC jedinici, ali ne proporcionalno manju nego što je povećanje iskorišćenosti raspoloživih MAC jedinica na FPGA čipovima. Takođe, smanjenje zahteva u broju potrebnih LUT-ova omogućava da Argus akcelerator bude implementiran na pristupačnijim FPGA čipovima zadržavajući veliku većinu procesorske snage CoNNa-e.

NEURAghe [33] je još jedan predstavnik FPGA akceleratora koji ima odličan odnos potrebnih LUT-ova po jednom DSP bloku (jednoj MAC jedinici). Balansirano korišćenje resursa je preduslov za implementaciju akceleratora na kompaktnim FPGA čipovima. Iako je korišćenje raspoloživih resursa balansirano, NEURAghe koristi jako veliki broj resursa (864 DSP bloka), što predstavlja prepreku njegovoj impelmentaciji na kompaktnim FPGA platformama. Pored navedenog, Argus akcelerator prevazali prijavljene performanse (5.5 fps za VGG 16) sa značajno manje iskorišćenih resursa.

Akceleratori predstavljeni u referencama [36], [37] i [38] spadaju u grupu poslednje publikovanih arhitektura baziranih na FPGA SoC-evima. Akcelerator razvijen u radu [38] se značajno oslanja na karakteristike procesorskog sistema koji se nalazi na SoC FPGA čipu, koristi NEON aritmetičko-logičke jedinice specifične za ARM familiju čipova. Neke od slojeva CNN-ova akcelerira na NEON jedinicama te se ne može smatrati nezavisnim akceleratorom. Zbog specifičnog načina povezivanja DSP blokova na FPGA platformama u lance i pakovanju više operacija u jednom taktu, arhitektura predstavljena u referenci [38] ostvaruje odlične performanse po utrošenom LUT-u, i nešto slabije po utrošenom DSP-u u poređenju sa najmoćnijim akceleratorima. Navedeno je posledica nemogućnosti da procesira orezane mreže. Sa druge strane arhitekture prezentovane u referencama [36], i [37] procesiraju

orezane mreže što akceleratoru prikazanom u radu [37] omogućava da ostvari izuzetne performanse po svim metrikama. Zbog visokih performansi smatramo da bi bilo korisno uporediti arhitekturu opisanu u radu [36] sa Argus akceleratorom kako bi se potvrdili potencijalni kvaliteti novog algoritma za orezivanje i Argus arhitekture.

3 Razvoj algoritma za klasterovanje kernela unutar konvolucionih slojeva

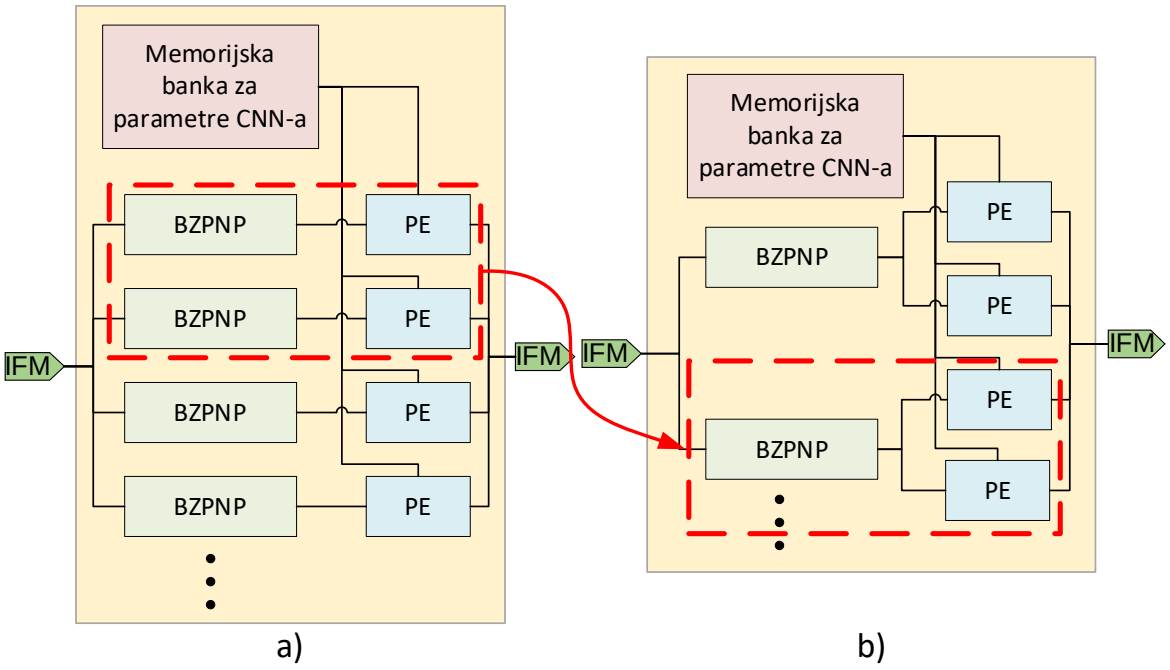
U ovom poglavlju predstavljen je algoritam za klasterovanje kernela unutar konvolucionih slojeva, ali i neurona u FC slojevima. Identičan algoritam je moguće promenjivati na oba tipa slojeva jer se FC sloj može posmatrati kao specijalni tip konvolucionog sloja. U prvom delu je predstavljena ideja klasterovanja zajedno sa pogodnostima koje donosi, da bi u nastavku bio predočen kompletan algoritam za klasterovanje i priprema modela CNN-a za dalje orezivanje.

3.1 Motivacija za razvoj algoritma za klasterovanje kernela

Prilikom implementacije više različitih arhitektura koje su prethodile Argus akceleratoru, ustanovljeno je da BZPNP modula zahteva značajanu količinu hardverskih resursa, najviše LUT-ova. Da bi se postigao visok stepen konfigurabilnosti, to jest, generalizacije hardvera bilo je potrebno da svaki PE ima pridružen BZPNP modul. Na ovaj način se dobija arhitektura koja je nezavisna od algoritma za orezivanje i odnosa u kom se nalaze orezani kerneli. Jednostavno govoreći, svaki BZPNP i PE modul čine nezavisnu celinu koja se snabdeva odgovarajućim delom IFM-a i može da procesira po jedan kernel orezane mreže, nezavisno od algoritma za orezivanje. Ovakav pristup daje visoku fleksibilnost arhitekturi, ali često nije optimalan u slučaju FPGA implementacije. Naime, velika većina PE-ova se, na FPGA kolima, mapira na DSP blokove, što znači da zahtevaju vrlo malo okružujućih resursa (LUT-ova). Pošto PE-ovi predstavljaju jezgra arhitektura, vrlo često je njihov broj korelisan sa performansama akceleratora što znači da je uglavno bolje imati što veći broj PE-ova kako bi se postigle što bolje performanse. Broj PE-ova na FPGA kolima je ograničen količinom raspoloživih DSP blokova, ukoliko izuzmemo vrlo neefikasne implementacije širokih množača koristeći LUT-ove za ovu namenu. Ukoliko je BZPNP kompleksan i zahteva veliku količinu LUT-ova, a akcelerator zahteva veliki broj PE-ova, postoji mogućnost da skaliranjem, BZPNP zauzmu sve LUT-ove pre nego svi DSP blokovi budu iskorišćeni. Zamislimo da BZPNP zahteva 500 LUT-ova, a da na FPGA kolu postoji 20k LUT-ova i 128 DSP blokova. Teoretski maksimum za broj BZPNP-ova na ovom FPGA kolu je 40 što implicira da bi od 128 raspoloživih DSP blokova ostalo 88 neiskorišćenih.

Kako bi se redukovao navedeni disbalans postoje dva pravca optimizacije. Jedan je smanjenje veličine (kompleksnosti) BZPNP bloka ili redukovanje broja BZPNP-ova po PE-u. U ovoj doktorskoj disertaciji su primenjena oba pristupa optimizaciji, konkretno klasterovanje ima za

cilj smanjenje BZPNP-ova po jednom PE-u. Na slici 11 su prikazani uobičajen način projektovanja univerzalne arhitekture (a) i akceleratora koji koristi jedan BZPNP za dva PE-a koja čine jedan klaster. U nastavku je detaljno izložena ideja kako je potrebno pripremiti CNN, to jest, kernele unutar konvolucionih slojeva kako bi bilo moguće koristiti optimizovanu arhitekturu.

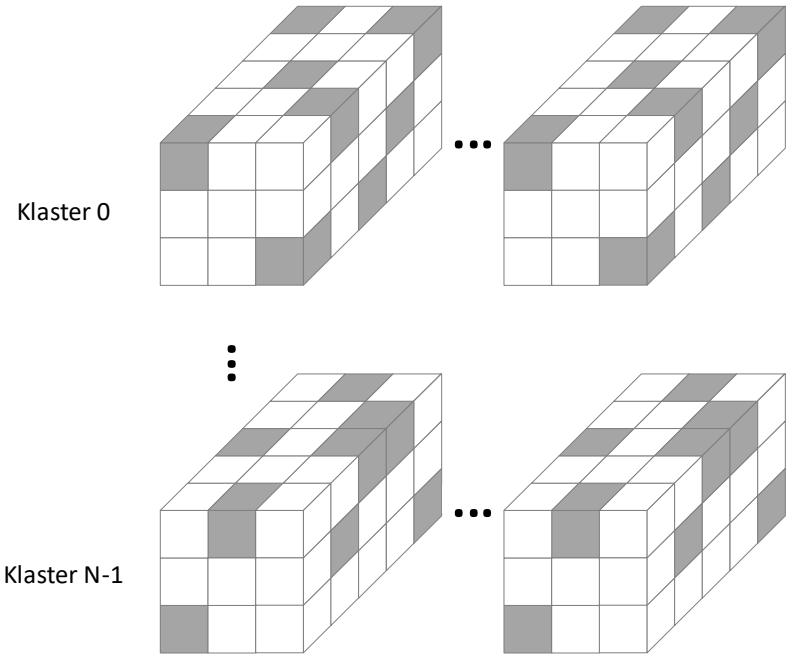


Slika 11 Uobičajen pristup projektovanju arhitektura sa jednim BZPNP po PE-u (a) i predloženi pristup (b).

Potreba za zasebnim BZPNP blokom po jednom PE-u je posledica algoritama za orezivanje koji uglavnom uklanjaju nepotrebne parametre CNN-a na nivou pojedinačnih kernela/neurona. U akademskoj zajednici pokazano je da je nivo redundantnosti parametara unutar kernela veliki i da se u nekim slojevima može ukloni i više od 90% parametara, ukoliko se svaki kernel orezuje nezavisno [39]. Napomenimo i to da je za kriterijum odabira nepotrebnih parametara odabrano da se parametri sa najmanjom apsolutnom vrednošću postave na nulu. Orezivanje je izvršeno inkrementalno, što znači da je prvo uklonjeno nekoliko procenata parametara sa najmanjom vrednošću, zatim je treniran CNN kako bi tačnost vratili na prvobitni nivo. Potom je orezano nekoliko narednih parametara iz svakog kernela i tako redom dokle god je bilo moguće povratiti prvobitnu tačnost.

Kako bi se smanjili zahtevi za hardverskim resursima, moguće je napraviti kompromis između nivoa orezivanja i slobode algoritma za orezivanje. Polazna pretpostavka je da se u svakom sloju mogu pronaći najmanje dva kernela čija se većina bitnih parametara (preostalih posle

orezivanja) nalaze na istim pozicijama. Ukoliko je ovo tačno, algoritam za orezivanje se može implementirati tako da za parove, ili grupe kernela, uklanja redundantne parametre na istim pozicijama. Ovako orezan CNN bi u svakom sloju imao grupe kernela čiji se parametri različiti od nula nalaze na istim pozicijama. Ove grupe kernela u nastavku zovemo klasterima. U pogledu redukovanja hardverskih zahteva, ovakav pristup orezivanju dovodi do toga da je dovoljno instancionirati jedan BZPNP blok po klasteru PE-ova kao na slici 11a). Primer ishoda orezivanja jednog sloja CNN-a, gde je orezivanje izvršeno nad klasterima kernela, je prikazano na slici 12. Uočimo da su pozicije preostalih parametara kernela (sive kockice) na istim pozicijama unutar klastera.



Slika 12 Preostali parametri (sive kockice) CNN-a posle kompresije korišćenjem principa klasterovanja.

Napomenimo i to da na slici 12 algoritam za orezivanje nije predmet razmatranja te se smatra da isti nema nikakva dodatna ograničenja. To znači da je algoritam za klasterovanje kernela nezavisan od algoritma za orezivanje.

3.2 Opis predloženog algoritma za klasterovanje

Detaljnije gledano, algoritam za klasterovanje kernela unutar konvolucionih i FC slojeva se može podeliti u tri koraka:

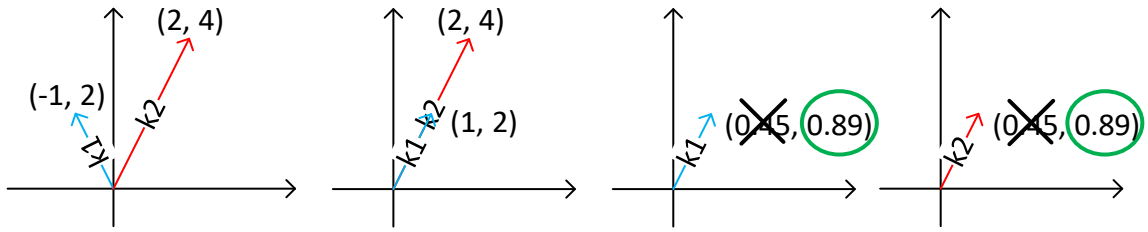
1. Za svaki konvolucioni i FC sloj potrebno je izračunati koeficijent međusobne sličnosti kernela/neurona i predstaviti ga u odgovarajućem obliku (npr. matrica sličnosti).

2. Izvršiti klasterovanje kernela nekim od algoritama za klasterovanje.
3. Preurediti pozicije kernela tako da se kerneli koji pripadaju istom klasteru nalaze jedan do drugog unutar sloja.

3.2.1 Računanje matrice sličnosti kernela/neurona

Za računanje sličnosti među kernelima/neuronima odabran je skalarni proizvod dva vektora kao najčešće korišćen pristup. Intuitivno gledano, sličnost kernela možemo računati na ovaj način zbog:

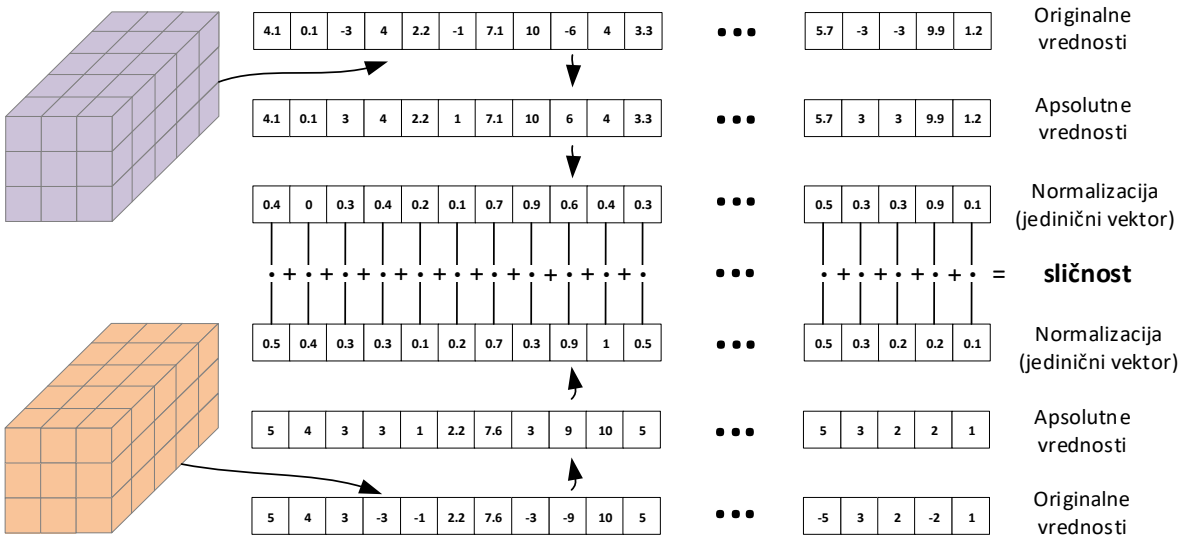
1. Orezivanje CNN-ova se vrši tako što se uklanjaju parametri koji imaju najmanju vrednost.
2. Zbog 1), kerneli su slični ako imaju mnoštvo parametara sa velikim apsolutnim vrednostima na istim pozicijama.
3. Zamislimo dva kernela koja imaju samo 2 parametra, što znači da ih je moguće predstaviti u ravni kao na slici 13. Zbog 2) nije bitno u kom se kvadrantu nalazi vektor, stoga ih je pre računanja sličnosti moguće prevesti u prvi kvadrant.
4. Identični kerneli će se smatrati oni čiji su vektori posle 3) kolinearni.
5. Iz 4) je jasno da dužina vektora nije bitna, te je moguće preračunati jedinične vektore za svaki kernel.
6. Skalarni proizvod, konačno, daje vrednost koja predstavlja sličnost dva kernela.



Slika 13 Intuitivna ideja iza upotrebe skalarnog proizvoda vektora za računanje sličnosti između kernela.

Prateći postupak za određivanje sličnosti na slici 13, jasno je da su vektori k1 i k2 slični u smislu pozicije parametara sa velikim apsolutnim vrednostima te je potrebno svrstati ih u isti klaster. Naravno, kerneli nisu vektori, ali ih je moguće predstaviti kao vektore bez narušavanja kvaliteta računanja sličnosti. Na slici 14 je prikazan primer dva kernela čije su dimenzije (osim dubine) vrlo čest primer kernela velikog broja različitih slojeva CNN modela. Prvi korak u računanju sličnosti je da se kerneli prevedu u jednodimenzionalni niz parametara. Svaki kernel

čine 9 nizova od 5 elemenata koje ćemo u nastavku zvati štapići. Dakle, štapić predstavlja jednodimenzionalni niz, deo kernela po dubini. Potrebno je u proizvoljnom redosledu posložiti štapiće jedan za drugim kako bi se dobio vektor dimenzije 45 elemenata. Naravno, bitno je da se i ostali kerneli unutar sloja serializuju na isti način. Posle serializacije, računaju se apsolutne vrednosti parametara da bi se potom preračunali jedinični vektori za svaki kernel. Poslednje u nizu je računanje skalarnog proizvoda jediničnih vektora što za rezultat daje sličnost dva kernela. Ukoliko je ova vrednost veća, kerneli su sličniji i obrnuto.



Slika 14 Detaljan prikaz računanja sličnosti dva kernela dimenzija 3x3x5.

Da bi se generisala matrica sličnosti za određeni sloj, potrebno je ponoviti navedeni postupak između svaka dva kernela. Na primer, u slučaju sloja sa 32 kernela potrebno je izračunati 480 sličnosti, odnosno, sve vrednosti iznad glavne dijagonale matrice. Vrednosti na glavnoj dijagonali treba da budu manje od 0 kako bi se izostavilo razmatranje sličnosti kernela samog sa sobom. Pošto rezultat sličnosti mora da bude veći ili jednak nuli dovoljno je da na glavnoj dijagonali budu negativni brojevi, na primer, -1. Sličnosti ispod glavne dijagonale su identične vrednostima iznad dijagonale.

U tabeli 2 je prikazan primer matrice sličnosti sloja koji ima 8 kernela i mogući raspored kernela po klasterima. Klasteri su naznačeni različitim bojama tako da kerneli 0 i 2 pripadaju jednom klasteru, kernel 1 i 5 drugom, kerneli 3 i 4 trećem klasteru, a kerneli 6 i 7 četvrtom.

Da bi se izvršilo klasterovanje, potrebno je matricu sličnosti procesirati odgovarajućim algoritmom. Zarad jednostavnije arhitekture akceleratora, potrebno je izvršiti klasterovanje tako da klasteri budu jednake veličine, vodeći računa da ta veličina bude unapred poznata.

Priroda algoritama za klasterovanje kao što su K-Means [40], DBSCAN [41] itd., je takva da ne garantuju da će klasteri sadržati jednak broj elemenata iz skupa podataka nad kojim se kreiraju klasteri. Posmatrano iz ugla nenadgledanog mašinskog učenja, kome algoritmi za klasterovanje pripadaju, ovo je očekivana pojava. Šta više, ishod rada algoritma u mnogome zavisi od inicijalnog klasterovanja na početku rada. Na primer, K-Means algoritam može da generiše klastere koji se svaki put značajno razlikuju po broju elemenata unutar klastera. Sa druge strane, za ovu namenu je moguće iskoristiti neki od algoritama za particionisanje. Potreban uslov je da algoritam deli polazni skup na balansirane particije u pogledu broja elemenata. Dalje, ove particije možemo posmatrati kao klastere sličnih kernela, ukoliko je ulaz u algoritam matrica sličnosti. U slučaju ove disertacije korišćen je Kernighan-Lin (KL) algoritam [42] koji ima sve potrebne osobine.

Tabela 2 Primer matrice sličnosti i klastera podeljenih po bojama.

	0	1	2	3	4	5	6	7
0	-1	0.5	15.4	14.4	0.4	5.5	2.1	0
1	0.5	-1	2.2	3.3	1.1	12.7	0.7	1.1
2	15.4	2.2	-1	7.1	9.4	8.2	3.8	6.4
3	14.4	3.3	7.1	-1	10.8	11.4	2.4	4
4	0.4	1.1	9.4	10.8	-1	8.7	6.2	3
5	5.5	12.7	8.2	11.4	8.7	-1	0	3.3
6	2.1	0.7	3.8	2.4	6.2	0	-1	17.5
7	0	1.1	6.4	4	3	3.3	17,5	-1

KL algoritam kao ulaz uzima matricu sličnosti i u prvoj iteraciji je deli na dve particije identične veličine. Zatim, algoritam premešta elemente dve početne particije tako da se dobiju dve particije u kojima će se nalaziti najsličniji kerneli. U primeru iz tabele 2, posle prvog prolaza KL algoritma rezultat su dve particije od po 4 kernela. U sledećoj iteraciji KL se pokreće nad ove dve particije pojedinačno generišući tako po dve dodatne particije, to jest, ukupno četiri particije od po dva kernela. Jasno je da se u svakoj particiji nalaze slični kerneli te ove particije možemo zvati klasterima.

Gore opisani postupak se koristi u slučajevima kada je broj kernela unutar sloja jednak 2^n. Ipak, postoje slojevi gde broj kernela nije jednak 2^n te je za ove slojeve potrebno primeniti drugi metod klasterovanja. Jedan od ovakvih slojeva je i poslednji FC sloj u CNN-ovima koji za zadatak imaju klasifikaciju *ImageNet* skupa slika koji ima tačno 1000 klasa, posledično i 1000 neurona u poslednjem FC sloju. U slučaju da broj kernela unutar sloja nije jednak 2^n, sloj će biti klasterovan *greedy* algoritmom. *Greedy* algoritam korišćen u ovoj doktorskoj disertaciji kreće redom od kernela sa indeksom 0. Zatim traži njemu najsličniji i grupiše ga sa kernelom 0 u isti klaster. Ova dva kernela se izbacuju iz daljeg razmatranja i algoritam nastavlja sa prvim sledećim preostalim kernelom. Na primer, neka je to kernel sa indeksom 1. Zatim njemu traži najsličniji, izuzimajući prethodna dva kernela koja formiraju prvi klaster. Kada nađe najsličniji kernelu 1, grupiše ih u drugi klaster i tako redom dok se svi kerneli ne grupišu.

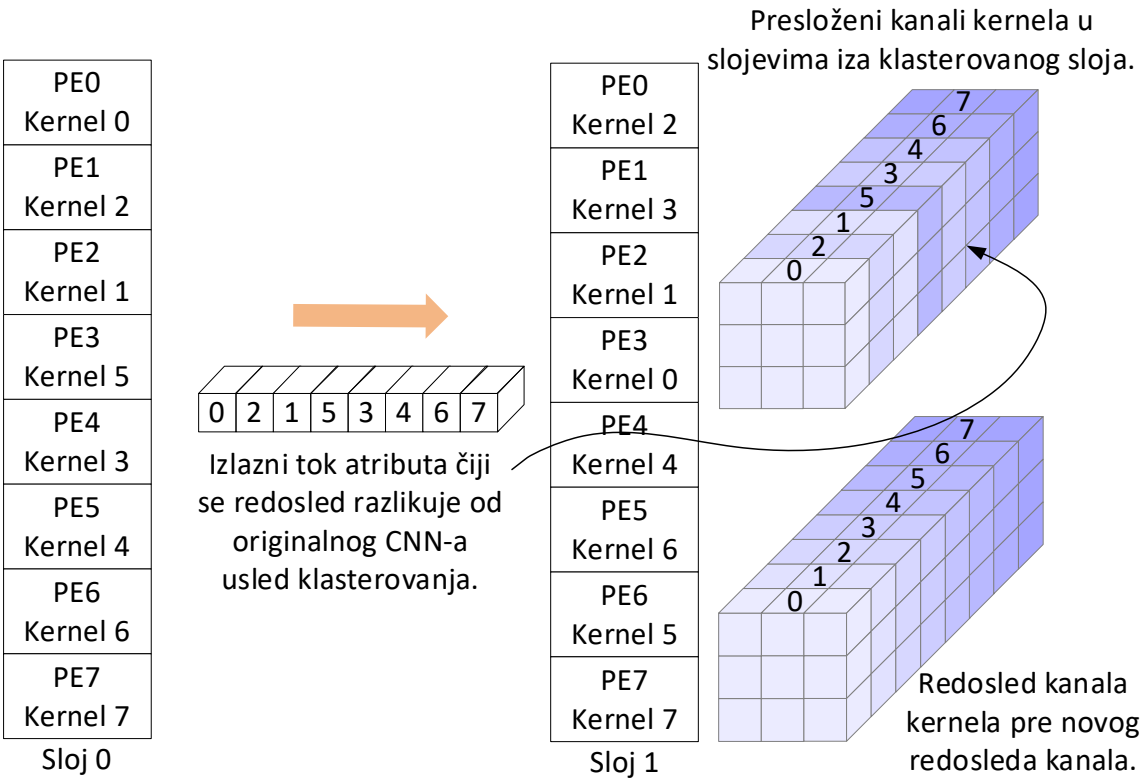
3.2.2 Algoritam za preslaganje kernela i kanala posle klasterovanja

Zarad jednostavnosti arhitekture, dobro je da se kerneli koji pripadaju istom klasteru procesiraju susednim PE-ovima. Na ovaj način će se unapred znati kako povezati BZPNP i PE-ove kao na slici 11. Pošto algoritam za klasterovanje ne vodi računa o tome da su kerneli unutar klastera i susedni u okviru originalnog CNN modela, jasno je da je posle klasterovanja potrebno presložiti kernele tako da su kerneli unutar jednog klastera susedni.

Posle opisanog preslaganja javlja se dodatni problem prilikom procesiranja narednog sloja. Na primer, neka prvi sloj čini ukupno 8 kernela tako da klastere čine kerneli (0,2), (1,5), (3,4) i (6,7). Ukoliko se kerneli preslože kao na slici 15, izlazni tok atributa će biti izmenjen u odnosu na originalni redosled. To znači da će izlazni atributi biti u redosledu 0, 2, 1, 5, 3, 4, 6 i 7, a ne u uzlaznom od 0 do 7. Da bi sledeći sloj pravilno prihvatio ove podatke potrebno je izmeniti

redosled kanala unutar kernela na isti način na koji su kerneli poređani u prethodnom konvolucionom sloju. Ovo je prikazano nijansiranjem kanala kernela na slici 15.

Dakle, radi smanjenja kompleksnosti arhitekture akceleratora potrebno je presložiti kernele unutar slojeva, a posledica toga je izmenjen redosled OFM tačaka što implicira potrebu za prilagođavanjem redosleda kanala kernela narednih slojeva. Napomenimo i to da opisani proces nije potreban zarad orezivanja CNN-a već isključivo da bi se arhitektura akceleratora pojednostavila.



Slika 15 Proces preraspoređivanja kanala unutar kernela u zavisnosti od klastera predhodnog konvolucionog sloja.

3.2.3 Pseudo kod algoritma za klasterovanje i preslaganje kernela CNN-a

Pseudo kod algoritma za klasterovanje i preslaganje, tačnije, kompletna priprema pre orezivanja je prikazana algoritmom 1. Prva *for* petlja (linija 1) prolazi kroz sve slojeve CNN-a i poziva funkciju *cluster_layer* koja vraća skup klasterovanih kernela. Funkcija *cluster_layer* uzima parametre sloja i smešta ih u promenljivu *weight_tensor*. Pre nego krenemo u izračunavanje sličnosti, potrebno je odrediti apsolutne vrednosti parametara tenzora i generisati jedinične vektore. Priprema *weight_tensor*-a je predstavljena linijama 12 i 13. Da bismo kreirali matricu sličnosti potrebno je da za svaki kernel (linija 15) odredimo njegovu

sličnost sa ostalima kernelima (*for* petlja uz liniji 16). Zbog simetričnosti, računamo samo gornju polovinu matrice što je u liniji 16 postignuto time da je početni indeks *for* petlje odabran kao i + 1. Efektivno, i + 1 predstavlja sledeći kernel u odnosu na kernel za koji tražimo sličnosti sa ostalima. Kada se odaberu dva kernela, izračuna se skalarni proizvod ovih vektora i rezultat se smešta na odgovarajuće mesto u matrici. Po završetku izračunavanja sličnosti poziva se algoritam za klasterovanje. Prema opisanom postupku, KL algoritam je pozivan iterativno u *top-down* maniru kako bi se kerneli razvrstali po klasterima.

Po završetku klasterovanja kernela unutar slojeva, funkcija *cluster_and_reorder_CNN* pristupa preslaganju kernela i kanala prema rezultatima klasterovanja. U drugoj *for* petlji ove funkcije (linija 3), opet prolazimo kroz kompletan CNN i za svaki sloj tražimo njegovog prethodnika (funkcija *find_pred*). Ukoliko je trenutni sloj iz grupe konvolucionih slojeva potrebno je presložiti njegove kernele u skladu sa formiranim klasterima za trenutni sloj. Za ovu operaciju je zadužena funkcija *reor_kernels*. Kanale u trenutnom konvolucionom sloju je potrebno presložiti prema redosledu kernela prethodnog konvolucionog sloja. Pored konvolucionih, potrebno je presložiti i kanale u slučaju da je trenutni sloj tipa *Batch-Normalization* [18]. Obe funkcije, *reor_kernels* i *reor_channels* prihvataju niz koji predstavlja redosled kernela u klasterima, to jest, novi redosled kernela (argument *clusters*). Pošto je potrebno presložiti kernele/kanale, na početku funkcija se formira matrica oblika matrice kernela tekućeg sloja. Vrednosti ove nove matrice (promenljiva *new_kernels*) su inicijalizovane na nulu. Funkcija *reor_kernels* prolazi kroz sve klastere i premešta kernele klastera na nove pozicije unutar *new_kernels*, što se postiže uvećavanjem iteratora *i*. Primetimo da pseudo kod odgovara korišćenom slučaju kada su klasteri veličine dva kernela. Kada se preslože svi kerneli, potrebno je zameniti originalni raspored kernela tekućeg sloja sa *new_kernels* što je učinjeno u liniji 26. Vrlo slično *reor_kernels* funkciji, i *reor_channels* prolazi kroz klastere, s' tim da promenljivu *clusters* tretira kao niz po kome treba sortirati kanale kernela koristeći funkciju *sort_channels*.

ALGORITAM 1: Klasterovanje kernela

```
0   func cluster_and_reorder_CNN(CNN_model)
1       for layer in CNN_model:
2           clusters[layer] = cluster_layer(CNN_model, layer.name)
3       for layer in CNN_model:
4           conv_pred = find_pred(CNN_model,layer)
5           if(layer.type == Conv):
```

```
6             reor_kernels(cluster[layer], layer)
7             reor_channels(cluster[conv_pred] , layer)
8           if(layer.type == BatchNorm):
9             reor_channels(cluster[conv_pred] , layer)

10  func cluster_layer(CNN_model,layer_name)
11    weight_tensor = get_tensor(CNN_model,layer_name)
12    weight_tensor = abs(weight_tensor)
13    weight_tensor = calc_unit_vector(weight_tensor)
14    kernel_num = length(weight_tensor)
15    for i in range(0, kernel_num):
16      for j in range(i+1, kernel_num):
17        sim[i][j] = dot_product(weight_tensor[i], weight_tensor[j])
18    clusters = Iter_Kerninghan_Lin(sim)
19    return clusters

20  func reor_kernels(clusters, layer)
21    new_kernels = zero_matrix(layer.kernels.shape)
22    i = 0
23    for cluster in clusters:
24      new_kernels[i, i+1] = layer.kernels[cluster:]
25      i = 2*i
26    layer.kernels = new_kernels

27  func reor_channels(clusters, layer)
28    new_kernels = zero_matrix(layer.kernels.shape)
29    i = 0
30    for cluster in clusters:
31      kernel0 = sort_channels(layer.kernels[cluster[0]], clusters)
32      kernel1 = sort_channels(layer.kernels[cluster[1]], clusters)
33      new_kernels[i, i+1] = [kernel0, kernel1]
34    layer.kernels = new_kernels
```

Do sada nam je poznat samo jedan rad [10] u kome je orezivanje mreže izvedeno nad grupama kernela, a ne nad pojedinačnim kernelima. Razlike između postupka prezentovanog u ovoj doktorskoj disertaciji i onog prikazanog u radu [10] su:

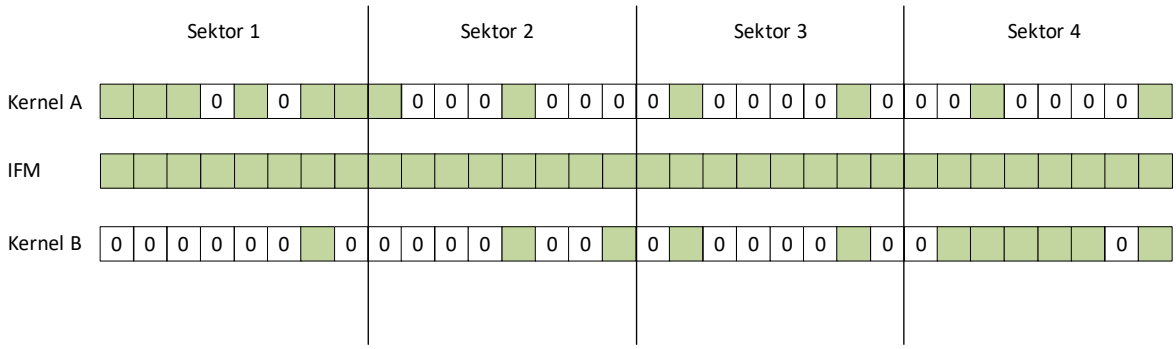
1. U objavljenom radu autori manuelno određuju veličine klastera za svaki sloj pojedinačno. Sa druge strane, u predloženom algoritmu veličina klastera je jednaka za sve slojeve što dodatno pojednostavljuje hardver.
2. U radu [10] autori koriste drugačiji algoritam za orezivanje nego algoritam koji je predstavljen u sledećem poglavlju. Algoritam za orezivanje korišćen u ovoj disertaciji će učiniti da broj operacija prilikom izračunavanja pojedinačnih konvolucija bude

identičan za sve klastere unutar sloja. U objavljenom radu postoji mogućnost da klasteri PE-ova budu neravnomerno opterećeni što iziskuje dodatne bafere koji bi balansirali ovakav raspored opterećenja. Ovi baferi značajno utiču na hardverske zahteve, najviše iz razloga što je potrebna velika propusna moć bafera kako bi PE jedinice ostale uposlene.

4 Poboljšanje postojećeg algoritma za orezivanje CNN-ova sa akcentom na FPGA platforme

U ovom poglavlju je detaljno izloženo poboljšanje postojećeg algoritma za orezivanje sa akcentom na probleme koje otklanja u odnosu na većinu drugih algoritama. Između ostalog, predstavljene su i druge potencijane mane potpuno slobodnog orezivanja kernela.

U opštem slučaju algoritmi za orezivanje koji uklanjaju parametre CNN-ova nad pojedinačnim kernelima mogu da uklone različit broj parametara u svakom od kernela jednog sloja. Različit broj parametara između kernela jednog sloja možu dovesti do smanjenja efikasnosti akceleratora koji obrađuje ovako orezane CNN-ove. Ovaj potencijalni problem za akcelerator, može se lako predupredити forsiranjem algoritma da svi kerneli jednog sloja imaju jednak broj preostalih parametara, bez značajnog uticaja na tačnost orezanog CNN-a. Drugi problem koji je daleko kompleksniji jeste raspored preostalih parametara unutar kernela. Prilikom orezivanja se dešava da se parametri jedanog kernela grupišu na različitim pozicijama u odnosu na druge kernele. Intuitivno gledano, neki delovi IFM-a više ekscituju određene kernele dok drugi kerneli akcentuju neke druge delove IFM-a. Ovakvo grupisanje preostalih parametara dovodi do toga da dok neki PE-ovi obrađuju delove IFM-a koji se trenutno dopremaju, drugi PE-ovi čekaju da se na ulazu pojave delovi IFM-a koji su od interesa za kernele koje oni procesiraju. Kao krajnji ishod, ovakav pristup orezivanju potencijalno može značajno da utiče na performanse akceleratora. Slika 16 ilustruje ovu situaciju.

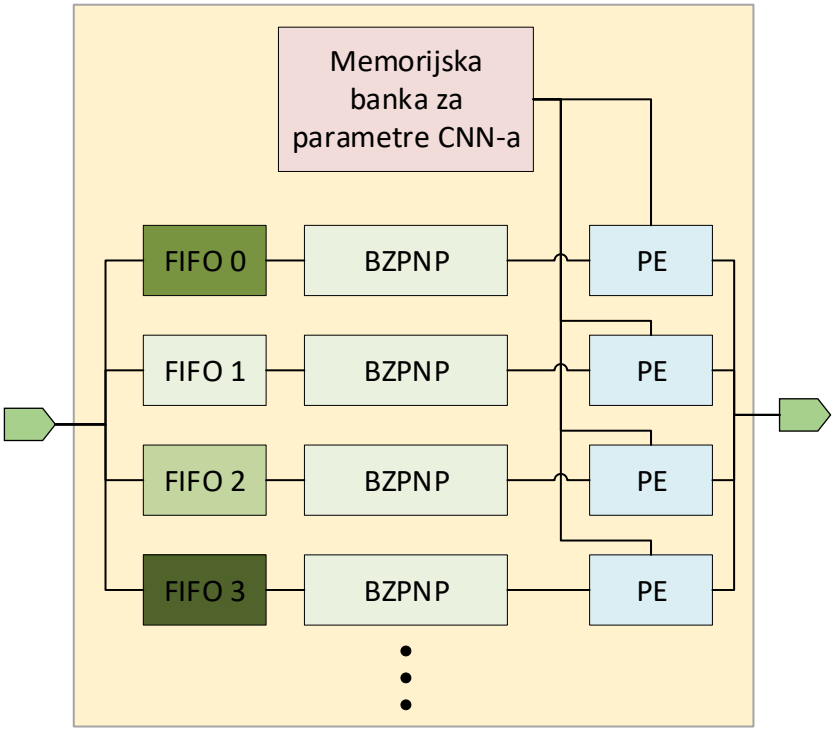


Slika 16 Primer neuravnoteženog rasporeda preostalih parametara posle orezivanja između kernela.

Kernel A ima grupisane parametre različite od nula na početku (sektor 1) dok kernel B ima veću gustinu preostalih parametara na kraju (sektor 4). Zamislimo da ulazni tok podataka (*Istream*) može da dopremi osam vrednosti IFM-a u jednom taktu, a PE da izvrši jednu MAC

operaciju u jednom taktu. Takođe, primetimo da *Istream* ne može da pređe na sledeću grupu od 8 vrednosti IFM-a dok svi PE-ovi ne završe obradu sektora 1. Za procesiranje sektora 1, dodeljeni PE za kernel A će potrošiti šest taktova dok će PE-u koji je dodeljen kernelu B biti potreban samo 1 takt. Ukupno za obradu sektora 1 je potrebno šest taktova. Sektor 2 i 3 su uravnoteženi i oba PE-a zahtevaju po dva takta za obradu. U sektoru 4 situacija je inverzna sektoru 1, sada PE dodeljen kernelu B zahteva šest taktova za procesiranje dok je za kernel A potrebno samo dva takta. U ovakvoj konstelaciji je neizbežno da u sektoru 1 PE kernela B ostaje neuposlen pet taktova, dok u sektoru 4 PE kernela A miruje četiri takta. Ovime se produžava trajanje konvolucije i umanjuje efikasnost arhitekture.

Navedeni disbalans je u velikoj meri moguće rešiti dodavanjem bafera za svaki PE kao što je to prikazano na slici 17. Ovi baferi najviše podsećaju na FIFO bafere sa razlikom na strani čitanja iz bafera. Pošto BZPNP zahteva element IFM-a koji odgovara parametru kernela koji je različit od nula, potrebno je da bafer ima mogućnost da preskoči neke elemente i da dostavi elemenat koji nije nužno sledeći po redu. Zbog sličnosti sa FIFO baferima zadržaćemo ovaj naziv u nastavku poglavlja. Primetimo da je zarad jednostavnosti izostavljeno klasterovanje te svaki PE ima pridružen BZPNP blok.



Slika 17 FIFO baferi kao rešenje za neravnomernu gustinu preostalih parametara u kernelima. Nijansa označava nivo popunjenosti bafera (tamnije je više).

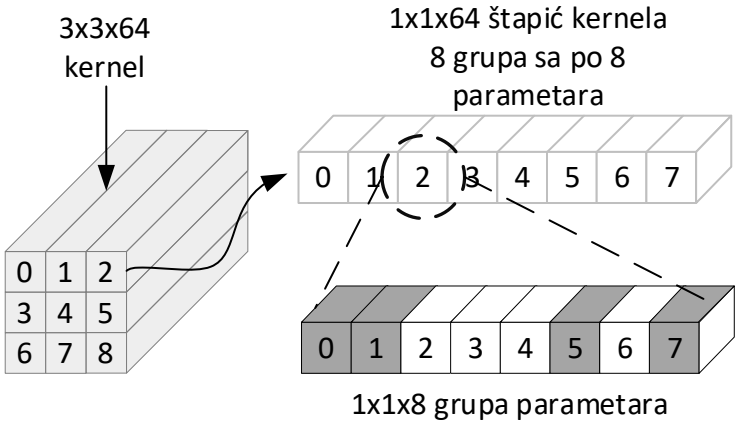
Na slici 17 je predstavljen primer u kome je FIFO 3 skroz pun što govori da odgovarajući BZPNP nije u mogućnosti da preskoči veći broj množenja jer je gustina preostalih parametara kernela koji obrađuje trenutno velika. Za razliku od FIFO bafera 3, FIFO 1 je skoro prazan jer kernel koji procesira odgovarajući BZPNP preskače više atributa IFM-a nego što ulazna magistrala uspeva da dopremi. U svetlu prethodnog primera možemo reći da PE pridružen FIFO baferu 3 procesira sektor 1 kernela A, dok PE dodeljen FIFO baferu 1 procesira sektor 1 kernela B.

Iako ovakav pristup rešava problem nebalansiranog rasporeda opterećenja PE-ova, implementacija je prilično zahtevna u pogledu hardverskih resursa, najviše LUT-ova na FPGA kolima. Jedan od razloga mapiranja FIFO bafera na LUT-ove je činjenica da ovi baferi nisu veliki da bi bilo isplativo implementirati ih na *Block RAM* (BRAM). Takođe, količina BRAM-ova je ograničena kao i broj DSP jedinica na svakom FPGA kolu i često se koristi za keširanje IFM-a, odnosno čuvanje parametara kernela. Uz to, ovi baferi moraju da imaju izuzetan odziv i da su ustanju da prihvate široke magistrale, to jest, velike količine podataka u svakom taktu od strane *Istream*-a.

Primena klasterovanja bi i ovde napravila značajnu uštedu u potrošnji hardverskih resursa umanjujući potreban broj bafera na identičan način kao i BZPNP blokova na slici 11b). Ipak, da bi se potpuno izbacili baferi neophodno je da sve PE jedinice u svakom taktu obrade isti segment IFM-a. Ukoliko u jednom taktu ulazni tok podataka doprema, na primer, osam ulaznih atributa, i PE jedinice moraju biti u stanju da obrade svih osam tačaka IFM-a u jednom taktu. Primetimo da ukoliko je orezivanje potpuno slobodno, u smislu određivanja koje parametre treba da izbaci, onda i PE-ovi moraju u najgorem slučaju da budu u stanju da izvrše osam MAC operacija u jednom taktu. Ovakav PE bi morao da ima osam MAC jedinica što bi značilo da u nekim drugim segmentima, u kojima nema osam parametara različitih od nula, neki MAC-ovi bi ostali neuposleni. Kao zaključak, može se izvesti da ukoliko su PE-ovi u stanju da u svakom taktu procesiraju sve podatke koji se dopremaju preko *Istream*-a, nema potrebe za FIFO baferima.

Algoritam za orezivanje koji rešava ovaj problem je predstavljen u radu [9]. Jedna od predloženih konfiguracija algoritma je da unutar svakog bloka od osam uzastopnih parametara kernela, algoritam ukloni (ореže) četiri. Ovakav način orezivanja se primenjuje nad svim kernelima u CNN-u. Na slici 18 je grafički prikaz jednog mogućeg ishoda orezivanja na ovakav način. Sivom bojom su obeleženi preostali parametri posle orezivanja.

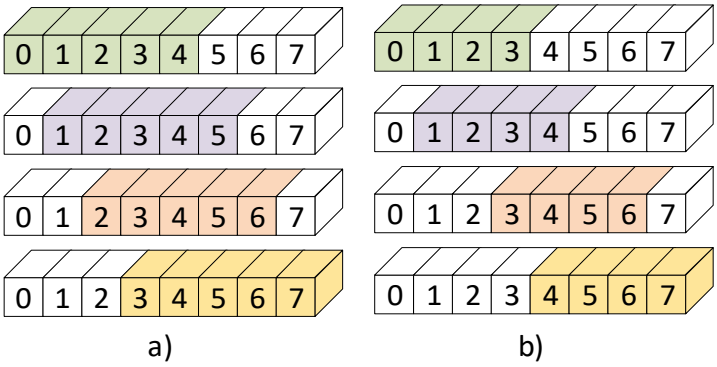
Ovakav pristup orezivanju CNN-ova ima za rezultat potpuno ravnomernu distribuciju parametara koje treba procesirati na nivou grupa, sa slobodom da se unutar grupa pozicije razlikuju. Iako je problem ravnomernog opterećenja PE jedinica rešen predloženim algoritmom [9], njegove konfiguracije (kao na primer, 4 od 8 preostalih parametara) nisu optimizovane za implementaciju na FPGA kolima. Autor nije uzeo u obzir karakteristike postojećih FPGA kola, konkretno, LUT-ova. U slučaju ovog algoritma LUT-ovi su jedini korišćeni resursi za implementaciju BZPNP blokova. Daljmo analizom, koja je predstavljena u poglavlju koje opisuje arhitekturu akceleratora, lako se utvrđuje da se BZPNP sastoji isključivo od četiri multipleksera, po jedan za svaki preostali parametar nakon orezivanja. Ovo znači da je za selektovanje svake tačke IFM-a koja odgovara preostalim parametrima kernela u grupi potreban po jedan multiplekser.



Slika 18 Primer preostalih parametara nakon orezivanja algoritmom [9]. Veličina grupe je osam, a stepen orezivanja 50%.

Za razliku od ASIC implementacija gde je granulacija potrošnje resursa na nivou jednog logičkog kola, u slučaju FPGA čipova imamo skokovit rast zauzeća resursa prilikom uvećanja implementiranih komponenti. Na primer, 6-ulazni LUT ima mogućnost da implementira bilo koju Bulovu funkciju koja ima šest promenljivih. Multiplekser 4-na-1 ima četiri ulaza podataka i dva selekciona bita, ukupno 6-ulazna funkcija. Manji multiplekser (3-na-1, 2-na-1) će takođe zauzeti jedan LUT. Naravno, ovo će biti slučaj pod uslovom da algoritam za sintezu ne uspe da optimizuje implementaciju manjih multipleksera kombinujući ih sa dodatnom logikom koristeći više-izlazne LUT-ove. Ukoliko se multiplekser poveća sa 4-na-1 na 5-na-1, tada će biti potrebna dva LUT-a za njegovu implementaciju. Ovo je skokoviti rast zauzeća resursa prisutan na svim FPGA čipovima. Za razliku od FPGA čipova, ASIC implementacija multipleksera 5-na-1 neće biti dvostruko veća od 4-na-1 već za nekoliko logičkih kola.

Na slici 19a) su prikazane moguće pozicije preostalih parametara posle orezivanja originalnim algoritmom. Kao što se može primetiti svaki od parametara se u opštem slučaju može naći na pet različitih pozicija što implicira multipleksere 5-na-1 unutar BZPNP blokova. Da bi se izbegao skokoviti rast zauzeća resursa moguće je uvesti dodatno ograničavanje dozvoljenih pozicija preostalih parametara kao na slici 19b). U toku orezivanja je moguće ograničiti algoritam da bira samo jednu od četiri pozicije, a da se ne naruši tačnost CNN-a. Ovime će se implementacija BZPNP bloka značajno pojednostaviti zbog mogućnosti korišćenja isključivo multipleksera 4-na-1. Kao primer, uzmimo da svaki BZPNP treba istovremeno da izdvoji četiri tačke IFM-a, što znači da ga čine četiri multipleksera 4-na-1. Ukoliko se koristi 16-bitna aritmetika to znači da će impelementacija jednog BZPNP-a zahtevati četiri 16-bitna multipleksera 4-na-1. Ovakvu strukturu je moguće implementirati pomoću 64 LUT-a. U slučaju da su multiplekseri kao u radu [9], zbog skokovitog porasta zahteva za resursima jedan BZPNP blok bi zahtevao dvostruko više LUT-ova, ukupno 128. Od ovakvog ograničenja benefita ima i ASIC implementacija, ali ne više od 20% redukcije u potrošnji resursa koji se koriste za multipleksere. Ovo je posledica finije granulacije u odnosu na FPGA tehnologiju.



Slika 19 Moguće pozicije preostalih parametara nakon orezivanja. Slika a) originalni algoritam [9]. Slika b) predloženo poboljšanje.

5 Sinergija poboljšanog algoritma za orezivanje i klasterovanja

Sledeći korak u sintezi novog algoritma za orezivanje je uparivanje prethodna dva, algoritma za klasterovanje i preslaganje kernela/neurona i poboljšanog algoritma za orezivanje. Klasterovanje je potrebno izvršiti kao vid preprocesiranja CNN-a, a onda nad klasterima treba pozvati algoritam za orezivanje iz prethodnog poglavlja. U nastavku je predstavljen algoritam i pseudo kodovi implementacije koja je izvršena pomoću *Python*-a [43] i *Keras* [44] biblioteke za mašinsko učenje. Takođe, izvršena je analiza uticaja algoritma za orezivanje na performanse i kompleksnost hardverskog akceleratora.

5.1 Opis predloženog algoritam za orezivanje sa klasterovanjem

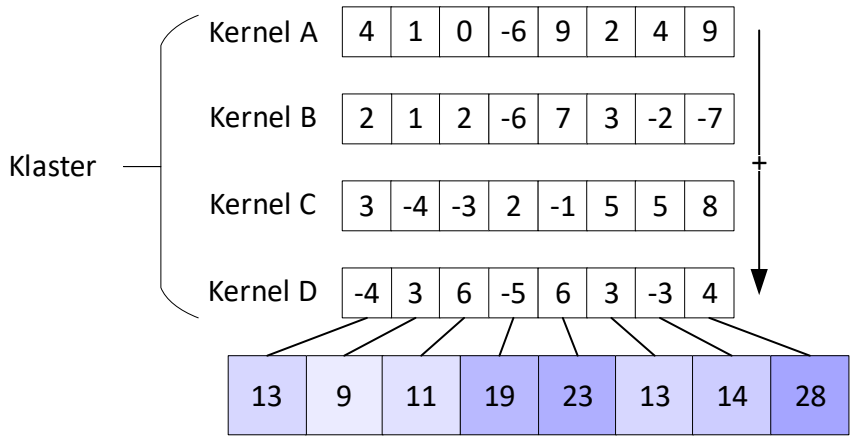
U ovom poglavlju je dat opis rada algoritma sa visokog nivoa apstrakcije kao i detaljan opis izvršavanje delova algoritma koji su bili najkompleksniji, kako u pogledu performansi CNN modela tako i u smislu reprodukcije prijavljenih rezultata od strane naučne zajednice. Pseudo kod, prikazan u algoritmu 2 (algoritam za orezivanje sa klasterovanjem), predstavlja pojednostavljenu verziju kompletnog algoritma za orezivanje sa klasterovanjem.

Da bi se odredila inicijalna tačnost neorezane mreže poziva se funkcija *evaluate_network* nad polaznim modelom CNN-a i tačnost se smešta u promenljivu *initial_accuracy*. Zatim se vrši klasterovanje i preslaganje elemenata mreže pozivom funkcije *cluster_and_reorder_CNN*. Ova funkcija odgovara prikazanoj funkciji istog imena u algoritmu za klasterovanje kernela. Kako bi se uštedelo na memorijskim zahtevima, ova funkcija ne generiše novu instancu modela već postojeću modifikuje. Sledeći korak je podela i serijalizacija kernela u grupe veličine osam parametara kao na slici 18 funkcijom *split_kernels_into_groups*. Ovim se preprocesiranje završava i algoritam ulazi u *for* petlju u kojoj se vrši inkrementalno orezivanje.

Orezivanje se sprovodi u četiri koraka što je prikazano spoljnom *for* petljom (linija 4). U prvom prolazu se uklanja jedan parametar iz svake grupe od po osam parametara (ukupno 12.5% parametara iz svakog konvolucionog sloja). Zatim se CNN trenira kako bi se postigla početna tačnost. U sledećem prolazu se uklanja još jedan parametar i tako redom dok se četiri od osam parametara ne uklone iz svake grupe.

Unutrašnja *for* petlja prolazi kroz sve grupe kreirajući listu koja predstavlja redosled kojim parametri treba da se uklone iz svake grupe pojedinačno (*create_pruning_order_list*). Slikovit prikaz rada *create_pruning_order_list* za jednu grupu jednog klastera

je prikazan na slici 20. Odabir parametara koje treba orezati po grupama se vrši nad svim kernelima unutar klastera za razliku od algoritma prikazanog u radu [9], u kojem se ovaj proces odvija na nivou pojedinačnog kernela. To znači da je potrebno napraviti odabir parametara tako da najviše odgovara celoj grupi kernela u klasteru, a ne pojedinačnom kernelu što je demonstrirano primerom na slici 20.



Slika 20 Prvi kandidati za orezivanje su predstavljeni svetlijom nijansom.

Na slici 20 je prikazan primer u slučaju klastera veličine četiri kernela sa akcentom na samo jednoj grupi od osam parametara. Prvi korak je normalizacija svih parametara unutar pojedinačnih kernela kako bi svaki parametar imao uticaj na klaster kakav ima u pojedinačnom kernelu. Neka su u ovom primeru parametri normalizovani na opseg od -10 do +10. Potom je potrebno sabrati apsolutne vrednosti parametara na istim pozicijama kao što je prikazano na slici 20. Tako sabrane vrednosti (slika 20 ljubičasti vektor) predstavljaju težinu koliko je svaka pojedinačna pozicija važna u klasteru. Manje vrednosti ovih težina predstavljaju pozicije parametara koji su prvi kandidati za orezivanje. U primeru na slici 20, prvi kandidati za orezivanje su predstavljeni svetlijim nijansama. Ukoliko neke pozicije imaju istu težinu, proizvoljna može biti odabrana za orezivanje.

Funcija *set_to_zero* postavlja na nulu parametre koje algoritam želi da ukloni. Pored navedenog, *set_to_zero* mora da vodi računa o dozvoljenim pozicijama preostalih parametara da bi se ispoštovala ograničenja sa slike 19b). Na primer, ukoliko u prva dva prolaza algoritam

izbaci parametre na pozicijama 0 i 1, a sledeći kandidat se nalazi na poziciji 2, *set_to_zero* će morati da odabere neki drugi parametar iz liste. Primetimo da na slici 19b) treći multiplekser nema mogućnost selektovanja pozicije 2, što nije slučaj u originalnom algoritmu [9].

ALGORITAM 2: Algoritam za orezivanje sa klasterovanjem

```
0   func prune_cnn(cnn_model, cluster_size):
1       Initial_accuracy = evaluate_network(cnn_model)
2       cluster_and_reorder_CNN(cnn_model)
3       kernel_groups = split_kernels_into_groups(cnn_model)
4       for i in range(4):
5           for group in kernel_groups:
6               pruning_list = create_pruning_order_list(group, clusters)
7               set_to_zero(group, pruning_list)
8           retrain pruned CNN(initial_accuracy)

9   func create_pruning_order_list(group, clusters):
10      for cluster in clusters:
11          cluster_group = group[cluster]
12          group_weight = [0, 0, 0, 0, 0, 0, 0, 0]
13          for kernel_group in cluster_group:
14              group_weight = group_weight + kernel_group
15          pruning_list.append(sort(group_weight))
16      return pruning_list
```

5.1.1 Rezultati orezivanja

Kako bi se proverio kvalitet i potencijalna šira primena algoritma, potrebno je odabrati CNN modele različitih karakteristika. Tri odabrane mreže su MobileNet v1, VGG-16 i ResNet50. Navedene mreže su iskorišćene kako bi se kvalitet algoritma proverio na modelima različite kompleksnosti. Tako MobileNet v1 spada u grupu vrlo kompaktnih CNN-ova. Kompaktnost MobileNet-a je jedan od razloga velike popularnosti ovog modela u embeded sistemima. Takođe, manji broj parametara originalne mreže u odnosu na, na primer, VGG-16 navodi na mišljenje da će biti teže orezati ovu mreže od VGG-16 što predstavlja dobar razlog zašto treba pokušati orezivanje nad MobileNet-om. Sa druge strane, VGG-16 je predstavnik izuzetno velikog modela CNN-a u smislu broja parametara. Prati ga izuzetna zastupljenost u evaluaciji performansi akceleratora. Jedna od prednosti VGG-16 u odnosu na druge modele je jasno uočljiva pravilnost koja se ispoljava između dimenzija kernela, to jest, slojeva koji ga čine. Upravo je navedena pravilnost karakteristika koja najverovatnije zadržava VGG-16 u eksperimentalnim sekcijama, najviše iz razloga što je manje zahtevno napraviti akcelerator koji podržava ovakve arhitekture modela. Zbog navedenog, VGG-16 je deo eksperimenta

evaluacije kvaliteta algoritma za orezivanje. Kao predstavnik modernih arhitektura odabran je ResNet50. Ovaj model odlikuje izuzetna dubina kao i slojevi netipični za prethodna dva modela.

Da bi se izvršila pravična procena kvaliteta algoritma, orezivanje je izvršeno nad modelima koji su prethodno trenirani nad ImageNet skupom slika. Već istrenirani modeli su preuzeti iz Keras biblioteke čije su originalne performanse prikazane u drugoj koloni tabele 3. Zadatak klasifikacije slika iz ImageNet skupa je široko prihvaćen u naučnoj zajednici kao dovoljno kompleksan da bi se nad njim vršili eksperimenti prilikom razvoja novih arhitektura CNN-ova.

U Tabeli 3 su prikazani rezultati rada algoritma za orezivanje nad tri različita CNN modela. Izvesna degradacija performansi se može uočiti prilikom orezivanja MobileNet-a i VGG-16 mreža dok to nije slučaj sa ResNet50. Da prijavljena degradacija nije značajna, najbolje govori činjenica da većina savremenih akceleratora procesira CNN-ove koristeći 8-bitnu aritmetiku, koja gotovo uvek dovodi do većih narušavanja performansi nego u, na primer, slučaju orezanog MobileNet v1 [45]. Uz navedeno, treba imati na umu da su arhitektura akceleratora i algoritam za orezivanje kreirani sa ciljem široke primene u embeded aplikacijama te će zadaci CNN-ova koji se budu izvršavali na akceleratoru biti značajno manje složenosti u poređenju sa klasifikacijom slika iz ImageNet skupa. Manja složenost zadataka CNN modela vodi ka lakšem orezivanju, to jest, smanjenju degradacije prikazane u tabeli 3.

CNN	**Neorezana Top-5 tačnost**	**Orezana razvijenim algoritmom**
ResNet50	92.1	92.1
VGG-16	90.1	89.8
MobileNet v1 224 1.0	89.5	89.0

Tabela 3 Rezultati orezivanja različitih CNN modela.

5.1.2 Detaljan opis bitnih delova algoritma za orezivanje i klasterovanje koji nisu prethodno opisani

U ovom poglavlju su predstavljeni neki od bitnih delova algoritma za orezivanje i klasterovanje, neophodnih za reprodukovanje prethodno objavljenih rezultata, ali i postizanje performansi orezanih modela navedenih u tabeli 3. Paragrafi su poređani hronološki prateći tok orezivanja, pa je tako na početku opisan postupak preprocesiranja slika. Zatim su navedeni razlozi zašto prvi sloj modela nije bio predmet orezivanja. Potom je definisana korišćena metoda za određivanje vrednosti *learning rate*-a kao jednog od najbitnijih parametara u

procesu učenja odnosno orezivanja. Dalje, dat je opis određivanja parametara koji se uklanjaju iz grupa unutar kernela, ali sada na nivou klastera. Na kraju, opisan je način kreiranja klastera i uticaj prirode koju ima korišćena heuristika (KL) na konačne rezultate.

Preprocesiranje slika predstavlja bitan faktor prilikom reprodukcije prijavljenih rezultata prethodno istreniranih modela iz Keras biblioteke. Slike u ImageNet skupu nisu identičnih dimenzija te ih je potrebno prilagoditi ulaznoj veličini modela. Čak ni modeli nemaju identične rezolucije ulaznih instanci, pa je tako ulazna slika u MobileNet veličine 224x224 piksela dok je u slučaju Inception v3 [21] ulaz veličine 299x299 piksela.

Priprema slika u slučaju MobileNet mreže se izvršava u dva koraka i to preskaliranje tako da jedna dimenzija ulazne slike bude 256 piksela, a potom se uzima centralni deo slike veličine 224x224. Preskaliranje se vrši tako što se manja od dve dimenzije (visina ili širina) postavlja na 256 dok se druga proporcionalno menja kao odnos originalnih dimenzija pomnožen sa 256. Ovo znači da će proporcije ostati identične kao i na početnoj slici sa tom razlikom što je manja dimenzija sada 256 piksela. Naravno, prilikom promena dimenzija potrebno je koristiti odgovarajuću interpolaciju. Drugi korak je centralno isecanje preskalirane slike na željenu dimenziju. U slučaju MobileNet-a to je 224x224 centralna piksela. Identičan postupak se koristi za, na primer, Inception v3 sa razlikom da je u prvom koraku potrebno sliku skalirati tako da manja dimenzija bude 299 piksela.

Prvi sloj CNN-ova se ostavlja u izvornom obliku, što znači da se ne vrši orezivanje nad njim iz dva razloga:

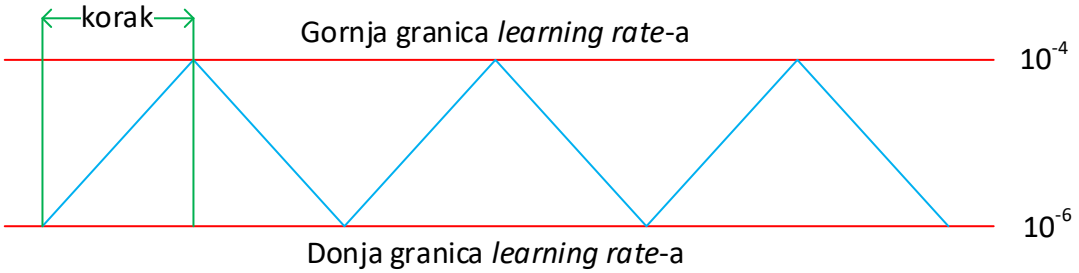
1. PE Argus akceleratora je u mogućnosti da u svakom taktu izvrši 4 MAC operacije u svakom štapiću što je više nego potrebno jer je dubina štapića ulazne slike u slučaju RGB formata jednaka tri. Dubina slike ograničava i dubinu kernela na tri po štapiću. Drugim rečima, štapići se dele na grupe od po osam parametara, a potom se orezuju četiri. Da bi se ispoštovala dubina štapića kernela od minimalno jedne grupe od osam parametara moguće je dopuniti svaki štapić kernela sa po pet parametara sa vrednostima 0. Jasno je da su ovako dopunjene grupe već orezane jer je više od četiri paramatera jednako nuli.
2. U slučaju većih instanci Argus akceleratora, usko grlo u prvom sloju postaje propusna moć memorijskog sistema, a ne procesorska snaga arhitekture te nije bilo potrebe

dodatno optimizovati orezivanje u prvom sloju. Osnovni razlog za povećanim zahtevima u pogledu performansi eksternih memorijskih jedinica se ogleda u činjenici da početni slojevi imaju mali broj kernela, najčešće 32, a neki čak i 16 kao što je slučaj sa MobileNet v1 0.5. Manji broj kernela na veći broj PE-ova ograničava korišćenje identičnih delova IFM-a na broj kernela. Drugim rečima, ako konvoluciono jezgro ima 32 PE-a, a sloj 16 kernela, jasno je da će biti moguće istim kernelom uposliti dva PE-a. Pošto nema smisla da PE-ovi sa istim kernelima obrađuju isti deo IFM-a, to znači da će svakom od njih biti dopremljen različit deo IFM-a. Kao posledica pojačava se pritisak na DRAM kontroler.

Primetimo da zbog ograničenja uvedenih na slici 19b) nije moguće procesirati tri uzastopna elementa IFM-a u prvom sloju svakog CNN-a. Ovaj problem je rešen na identičan način kao i omogućavanje akceleratora da procesira neorezane mreže. Rešenje je detaljno opisano u poglavlju 4.4.

Napomenimo i to da je u toku orezivanja pokušano sprečavanje promena vrednosti parametara kernela prvog sloja, međutim ovaj postupak nije doneo bolje rezultate nego u slučaju kada su se parametri menjali u toku orezivanja.

Learning rate predstavlja jedan od najbitnijih parametara prilikom treniranja CNN-ov. Pažljiv odabir ovog parametra utiče kako na krajnje performanse trenirane mreže tako i na brzinu treniranja. Prilikom orezivanja korišćena je *Cyclic Learning Rate* [46] (CLR) metoda. Ovaj pristup menja *learning rate* u toku treniranja modela, kao što je to prikazano na slici 21.



Slika 21 Cyclic Learning Rate, metoda određivanja learning rate parametra korišćen prilikom orezivanja.

Na slici 21 je moguće uočiti da vrednosti *learning rate*-a formiraju trouglove u toku treninga. Poželjno je za donju granicu odabrati neku malu vrednost ovog parametra, na primer, 10^{-6}. Jedan od načina odabira donje granice je i da ona bude identična vrednosti *learning rate*-a kojim su autori završili treniranje originalnih CNN modela. U slučaju orezanih mreža iz tabele

3, donja granica je postavljena na 10^{-6} dok je gornja varirala u eksperimentima u zavisnosti od modela u rasponu od 10^{-5} do 10^{-4}. Na početku treninga, *learning rate* će se povećavati do gornje granice, sa nagibom koji je definisan korakom na slici 21. Korak definiše vreme rasta odnosno opadanja *learning rate*-a izražen u epohama. Korak treba odabrati tako da bude između dve i osam epoha kako je navedeno u radu [46]. U slučaju orezivanja navedene tri mreže iz tabele 3, korak je bio postavljen na vrednost od četiri epohe. Prilikom korišćenja ovakvog pristupa za očekivati je da se pri vrhovima trouglova, to jest, na gornjoj granici *learning rate*-a, performanse modela donekle degradiraju. Ipak, u toku orezivanja je uočeno da mala degradacija najčešće vodi ka kvalitetnijem modelu kada *learning rate* ponovo dođe do donje granice nego u slučaju kada se gornja granica odabere tako da ne naruši trenutnu tačnost modela. Autor rada ovo opisuje kao izlazak iz lokalnog minimuma usled povećanja *learning rate*-a. Pored trougaonog oblika moguće je koristiti i trougaoni oblik kod koga je vrh svakog narednog trougla sredina između prethodne donje i gornje granice. Ovo opadanje može da ima i eksponencijalni oblik, međutim u toku orezivanja nismo primetili poboljšanja u odnosu na osnovni trougaoni oblik.

Grupisanje kernela unutar klastere se vrši samo jednom na početku prvog orezivanja, rezultati se čuvaju i identični se koriste u budućim eksperimentima nad istim CNN modelom. Tri su razloga za ovaj pristup. Jedan je srkaćenje procesa orezivanja svakog pojedinačnog eksperimenta za vreme potrebno za klasterovanje. Drugi razlog je formiranje identičnih uslova za svaki naredni pokušaj orezivanja. Treći razlog je utvrđen empirijski i tiče se krajnje tačnosti orezanog CNN modela koja je bila identična za različito klasterovane kernele usled klasterovanja KL heuristikom.

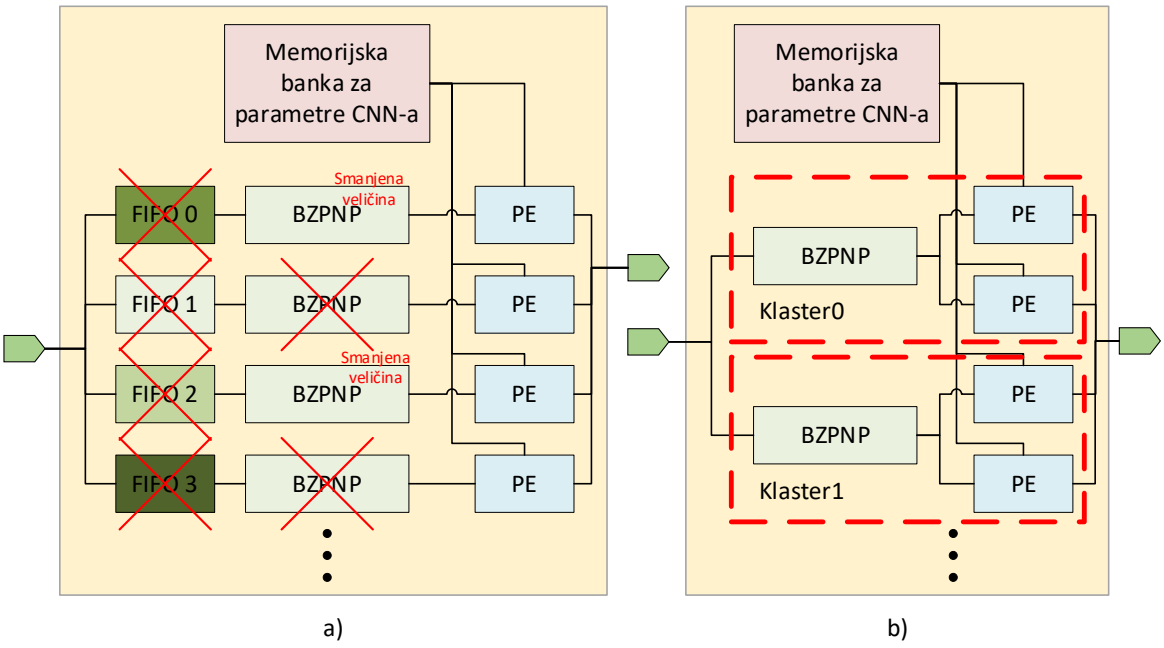
1. Pošto je problem klasterovanja NP težak, sledi da se ne mogu formirati optimalni klasteri u razumnom vremenu te je korišćena KL heuristika. Iako je vreme rada algoritma za klasterovanje značajno kraće od procesa orezivanja (nekoliko sati naspram nekoliko dana) ono ipak oduzima nekoliko sati na početku svakog eksperimenta. Ukoliko je orezivanje pokrenuto sa novim parametrima, na primer, za *learning rate*, vrlo je verovatno da će se u prvih nekoliko sati videti da li se eksperiment kreće u boljem pravcu nego prethodni eksperimenti. Ovim se štedi značajna količina vremena prilikom podešavanja algoritma za orezivanje.

2. Kako bi se utvrdili efekti menjanja parametara treninga CNN modela, potrebno je da ulazi između više eksperimenata sa različitim postavkama budu identični. Iako različiti klasteri, nastali usled više pokretanja KL algoritma ne utiče značajno na krajnji rezultat orezivanja, poželjno je da ova promenljiva bude isključena iz razmatranaj prilikom postavljanja parametara za treniranje mreža.
3. Kao što je rečeno, manja odstupanja između formiranih klastera nemaju uticaj na krajnji rezultat orezivanja. Ovo je bitna činjenica koja na neki način definiše stabilnost, to jest, predvidivost rada celokupnog algoritma za orezivanje.

5.2 Uticaj algoritma za orezivanje na potrebe za hardverskim resursima

Kao rekapitulacija, u ovom poglavlju su pobrojane sve korišćene optimizacije algoritma za orezivanje koje imaju uticaj na smanjenje zahteva za hardverskim resursima. Na slici 22 su naznačeni svi blokovi potencijalne hardverske arhitekture koji su primenom ovakvog načina orezivanja potpuno nepotrebni ili čija je veličina značajno smanjena u odnosu na prvobitno potrebnu.

1. Prva prednost razvijenog algoritma u odnosu na većinu prethodnih algoritama je eliminacija FIFO bafera. Pošto su PE-ovi ravnomerno opterećeni u svakom taktu, FIFO baferi za balansiranje opeterećenja nisu potrebni. Ovo je postignuto time što je svaka grupa od po osam uzastopnih parametara CNN-a podjednako orezana u smislu broja preostalih parametara. Primetimo da ovo nije poboljšanje koje donosi algoritam razvijen u ovoj doktorskoj disertaciji već je to nasleđena prednost polaznog algoritma predstavljenog u radu [9].
2. Ukoliko je veličina klastera jednaka dva, onda je potrebno dvostruko manje BZPNP-ova unutar akceleratora što predstavlja redukciju od dva puta. Ovo je direktno uticaj klasterovanja kernela.
3. Zbog dodatnog ograničenja algoritma za orezivanje prikazanog na slici 19b), BZPNP je dvostruko manji u odnosu na originalnu veličinu koja se dobija u slučaju primene rezultata objavljenih u radu [9]. Ovaj nivo redukcije je ostvariv ukoliko se arhitektura implementira na FPGA platformama dok se ušteda u slučaju ASIC implementacije kreće na nivou od 20%.



Slika 22 Rekapitulacija benefita korišćenja razvijenog algoritma za orezivanje (b) u odnosu na hipotetičku arhitekturu (a) koja ima podršku za proizvoljne algoritme za orezivanje.

5.3 Uticaj razvijenog algoritma za orezivanje na performanse

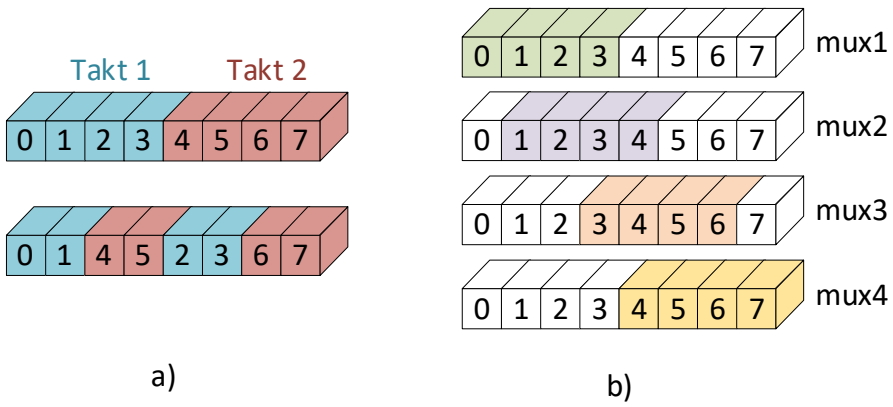
Direktan uticaj algoritma na performanse nije jednostavno predstaviti u ovom trenutku te je detaljno poređenje sa prethodno objavljenim arhitekturama dato u poglavlju koje sumira eksperimentalne rezultate. Ipak, ono što je jasno je da su PE-ovi ravnomerno opterećeni u svakom taktu što će imati značajnog uticaja na efikasnost arhitekture. Ravnomerno opetećenje u mnogome pojednostavljuje okružujuću logiku te je moguće instancionirati veći broj PE-ova, što uglavnom ima za posledicu poboljšanje performansi akceleratora.

5.4 Uticaj razvijenog algoritma za orezivanje na kompleksnost akceleratora

Kao što je i očekivano, u pogledu zahteva za hardverskim resursima, arhitektura koja je u stanju da procesira CNN-ove orezane predloženim algoritmom će biti kompaktnija od arhitektura koje podržavaju proizvoljne algoritme za orezivanje. Jedna hipotetička arhitektura koja podržava proizvoljni algoritam za orezivanje je prikazana na slici 22a) dok je arhitektura koja podržava predloženi algoritam uporedo prikazana na b) delu iste slike. Ipak, logička kompleksnost akceleratora se ne ogleda samo u količini iskorišćenih hardverskih resursa već i načinu na koji se isti koriste (međusobna interakcija, povezivanje itd.). Takođe, okružujući softver, koji vrši pripremu CNN-ova, postaje značajno složeniji. U nastavku glave su detaljno izloženi izvori navedenog usložnjavanja celokupnog razvoja akceleratora.

Klasterovanje značajno utiče na kompleksnost pripreme CNN-a za orezivanje, najviše zbog postupka preslaganja kernela kako bi se kerneli iz istog klastera našli jedan pored drugog. Ovo dalje povlači i preslaganje kanala u narednim konvolucionim i *Batch Normalization* slojevima.

Sa druge strane, optimizacija uvedena na slici 19b), stvara dodatne probleme prilikom procesiranja neorezanih CNN-ova. Zamislimo da je svaki PE realizovan tako da u svakom taktu može da izvrši četiri MAC operacije (što i jeste slučaj Argus akceleratora). Ovakav PE je u mogućnosti da procesirati svaku orezanu grupu u jednom, a neorezanu grupu od osam uzastopnih tačaka IFM-a, u dva takta kao na slici 23a). Ipak, obrada četiri uzastopne tačke jedne grupe IFM-a nije moguća zbog optimizacije prikazane na slici 23b). Razlog za ovaj nedostatak proizilazi iz činjenica da su BZPNP blokovi realizovani pomoću multipleksera 4-na-1. Na primer, treći multiplekser sa slike 23b) nema mogućnost odabira tačke IFM-a koja se nalazi na poziciji 2. To znači da u prvom taktu, kada se obrađuju prve četiri tačke jedne grupe IFM-a, nije moguće selektovati tačku sa pozicije 2. Kako bi se rešio navedeni problem moguće je presložiti grupe od po osam IFM tačaka na ulazu u akcelerator kao na slici 23a) - dole. Ovo je moguće učiniti pomoću četiri multipleksera 2-na-1 na jednom mestu u *Istream-u*. Navedena četiri multipleksera su veličine jednog BZPNP bloka, što predstavlja zanemarljivo povećanje akceleratora posebno ako se uzme u obzir koliki je doprinos u smislu fleksibilnosti arhitekture.



Slika 23 Povećanje kompleksnosti arhitekture zarad podrške procesiranja neorezanih modela uzrokovane ograničenjem uvedenim nad postojećim algoritmom za orezivanje [9].

Na slici 23b) je prikazan opseg IFM tačaka koje su dostupne multiplekserima. Multiplekser 1 može da selektuje jednu od prve četiri tačke, multiplekser 2 tačke od druge do pete i tako redom. Pošto smo uzeli da jedan PE može da izvrši četiri MAC operacije u jednom taktu, jasno je da je u prvom taktu potrebno procesirati prve četiri tačke IFM-a. U drugom taktu će biti

procesirane tačke na pozicijama od 4 do 7. Da bi u svakom taktu multiplekseri mogli da selektuju odgovarajuće tačke IFM-a potrebno je zameniti mesta tačkama 2 i 4 odnosno 3 i 5 kao na slici 23a) dole. Posle preslaganja, multiplekseri će selektovati odgovarajuće tačke IFM-a kao što je prikazano u tabeli 4.

	Takt 1		Takt 2	
	IFM tačka	Pozicija u grupi	IFM tačka	Pozicija u grupi
mux1	0	0	4	2
mux2	1	1	5	3
mux3	2	4	6	6
mux4	3	5	7	7

Tabela 4 Selektovanje odgovarajućih tačaka IFM-a u slučaju neorezanih CNN-ova po taktovima.

Podrške za neorezane CNN modele je potrebna kako bi se osigurala šira primena na buduće CNN-ove, koji će potencijalno sadržati nove tipove konvolucionih slojeva i/ili kombinacije više slojeva unutar blokova. Uz to, postoji mogućnost da je za neke specifične CNN-ove nemoguće orezati sve slojeve sa faktorom orezivanja od 50%, a da se pri tome ne degradira tačnost više nego što je prihvatljivo. Ovakvim pristupom se osigurava mogućnost akceleracije i ovakvih mreža.

6 Arhitektura akceleratora

U ovom poglavlju je opisan kompletan proces kreiranja akceleratora koji prati uobičajeni tok razvoja jednog hardverskog bloka. Polazna tačka ovog postupka je algoritam koji je potrebno hardverski akcelerirati. Zatim se nad algoritmom primenjuju tehnike optimizacije, kao što je na primer, razmotavanje petlji. Nakon optimizacije algoritma, pristupa se razvoju arhitekture, implementaciji i na kraju verifikaciji akceleratora.

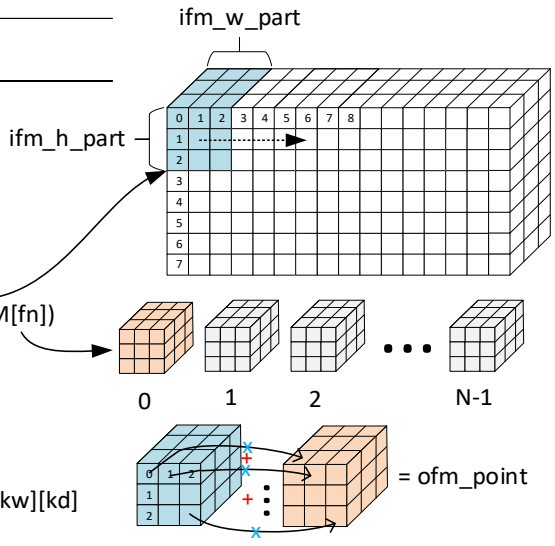
Arhitekturu akceleratora, koja je pogodna za procesiranje CNN-ova orezanih predloženim algoritmom ćemo nazvati Argus. Inspiracija za ovakav odabir imena se može pronaći u karakteristikama algoritma za orezivanje. Naime, Argus je stvorenje iz grčke mitologije koje je po celom telu imalo oči. Analogija između očiju, kernela, parametara i uopšte mašinske vizije je jasna. Ovo stvorenje je bilo stražar koje je čak i dok spava držalo otvoreno polovinu svojih očiju kako bi opažalo okolinu. Ova osobina stvorenja se na neki način može povezati sa činjenicom da predloženi algoritam za orezivanje orezuje tačno polovinu parametara slojeva, ali i dalje zadržava performanse kakve ima i bez orezivanja.

6.1 Optimizacija algoritma za procesiranje konvolucionih slojeva za efikasnu hardversku implementaciju

Najzahtevniji slojevi CNN mreža u pogledu računarskog vremena i prenosa podataka su svakako konvolucioni. Udeo vremena izvršavanja ovih slojeva u CNN-ovima se kreće i do 90% ukupnog vremena procesiranja [16], [19]. Iz navedenog je jasno da će krajnje performanse akceleratora biti u jakoj korelaciji sa kvalitetom arhitekture da efikasno izvršava konvolucione slojeve. U algoritmu 3 je prikazan pseudo kod koji predstavlja računanje odziva generičkog konvolucionog sloja. Na prvi pogled se jasno uočava veći broj *for* petlji koje se mogu razmotati (engl. *unroll*) na razne načine i time paralelizovati procesiranje određenih delova algoritma. Posledično i ubrzati obrada celog konvolucionog sloja. Prilikom uvođenja ovakvih optimizacija treba biti izuzetno pažljiv zato što baš odabir petlji i načina na koji će biti optimizovane može rezultovati potpuno različitim arhitekturama akceleratora. Pogrešan odabir petlje može značajno da utiče na složenost, ali i performanse krajnjeg hardvera.

ALGORITAM 3: Pseudo kod algoritma za računanje odziva konvolucionog sloja

```
     func calc_layer_ofm(IFM, KM):
L1     for(fn = 0; fn < Kernel_Num; fn++)
L2       for(y = 0; y < OFM_Height; y++)
L3         for(x = 0; x < OFM_Width; x++)
             ifm_h_part = y·Sv:(y·Sv + Kernel_Height)
             ifm_w_part = x·Sv:(x·Sv + Kernel_Width)
             ifm_bundle = IFM[ifm_h_part][ifm_w_part][:]
             OFM[x][y][fn] = calc_ofm_point(ifm_bundle, KM[fn])

     func calc_ofm_point(ifm_bundle, km):
L4     for(kh = 0; kh < Kernel_Height; kh++)
L5       for(kw = 0; kw < Kernel_Width; kw++)
L6         for(kd = 0; kd < Kernel_Depth; kd++)
             ofm_point += ifm_bundle[kh][kw][kd] · km[kh][kw][kd]
     return ofm_point
```



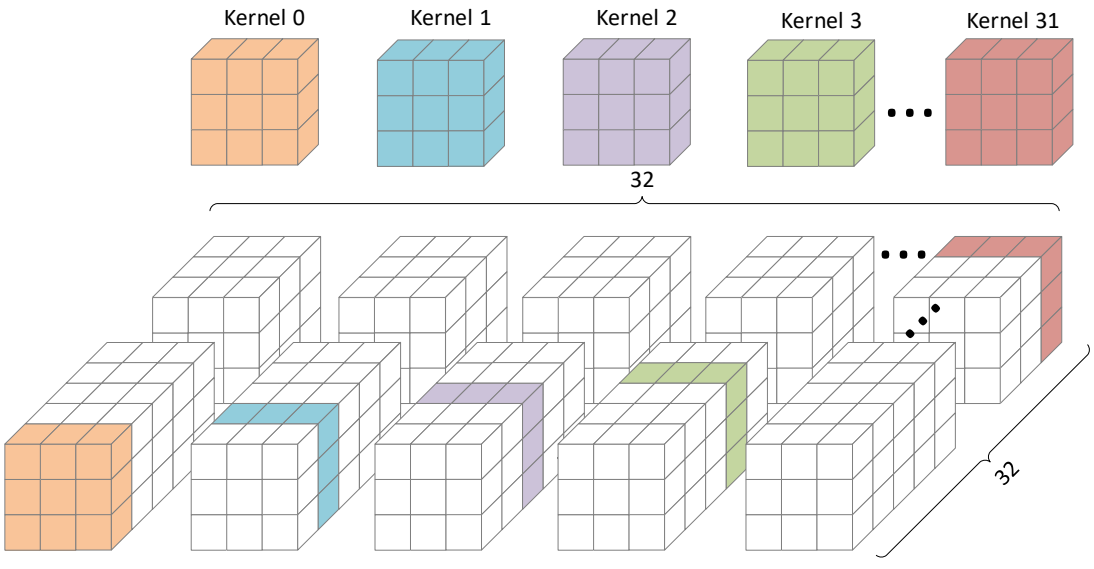
Zarad jednostavnosti i dalje podele algoritma 3 na hardverske blokove, računanje odziva konvolucionog sloja je podeljeno na dve funkcije od kojih *calc_ofm_point* računa pojedinačne skalarne proizvode kernela i odgovarajućeg snopa štapića, a *calc_layer_ofm* poziva prethodnu za svaku tačku OFM-a pripremajući joj ulazne argumente. Izostavljeni deo procesiranja u okviru konvolucionih slojeva je dodavanje *bias*-a i IFM *padding*. *Zero Padding* deo IFM-a neće uticati na vreme potrebno da se funkcija *calc_layer_ofm* izvrši iz razloga što je broj operacija u sloju određen veličinom OFM-a, a ne veličinom IFM-a. Ovo se može uočiti iz granica *for* petlji (L2 i L3) unutar funkcije *calc_layer_ofm*. Takođe, opciono dodavanje vrednosti *bias*-a na rezultat skalarnog proizvoda kernela i snopa štapića IFM-a može lako biti dodat na rezultat bez dodatne potrošnje vremena za ovu operaciju.

Funkcija *cala_layer_ofm* kao ulaz uzima IFM i sve kernele trenutnog sloja. IFM se procesira kao na slici u okviru algoritma 3, tako što kerneli klize po IFM-u sa leva na desno. Broj konvolucija po kanalu je definisan horizontalnom i vertikalnom dimenzijom OFM-a što je u algoritmu prikazano petljama L2 i L3. Dubina OFM-a je definisana brojem kernela u trenutnom sloju. Petlja L1 ide kroz sve kernele i time definiše navedenu dubinu OFM-a.

Druga funkcija, funkcija *calc_ofm_point*, za ulaz dobija trenutni snop štapića IFM-a (na slici u okviru algoritma plavim osenčeno) i jedan kernel. Izlaz funkcije je skalarni proizvod između kernela i snopa štapića. Petlje L4 i L5 prolaze kroz vertikalne, odnosno, horizontalne kordinate snopa štapića i kernela. Petlja L6 prolazi kroz sve kanale IFM-a za dati snop dokle god se ne

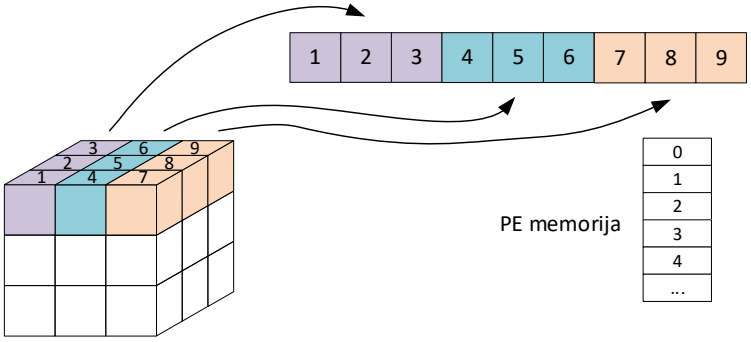
obrade sve tačke IFM-a po dubina. Dubina kernela je u velikoj većini slučajeva jednaka dubini IFM-a osim u slučaju *Depthwise* konvolucionih slojeva [8].

Specifičnost *Depthwise* konvolucionih slojeva se ogleda u obliku njihovog kernela, koji su dvodimenzionalni za razliku od kernela standardnih konvolucionih slojeva, koji su trodimenzionalni. Kako bi se omogućilo izvršavanje *Depthwise* konvolucionih slojeva na arhitekturama koje su prvenstveno optimizovane za standardne konvolucione slojeve, moguće je kernele dopuniti nulama i time ih učiniti trodimenzionalnim. Dopunjavanje nulama je prikazano na slici 24. Originalni, dvodimenzionalni kerneli su prikazani bojama, dok su njihove trodimenzionalne varijante dopunjene nulama, čiji parametri su prikazani belom bojom. Nule svakako neće uticati na krajnji rezultat izračunavanja te je sa algoritamske strane sve korektno. Naravno, ovakav pristup neće rezultovati u teoretskom maksimumu performansi pošto će PE blokovi izvršiti značajan broj nepotrebnih kalkulacija. Kako bi se nepotrebno procesiranje smanjilo koliko je moguće, nije potrebno dopuniti kernele do dubine IFM-a već to može biti neki značajno manji broj. Ovo najviše zavisi od karakteristika akcelerator, a u slučaju arhitekture prikazane u nastavku ove doktorske disertacije, kernele je potrebno dopuniti do dubine koja je jednaka broju procesorskih elemenata konvolucionog jezgra (32). Iako neefikasno, u eksperimentalnoj sekciji se može videti poređenje sa akceleratorom specijalizovanim za procesiranje CNN-ova sa *Depthwise* slojevima [47], koje dovodi u pitanje potrebu razvoja posebnog hardverskog modula za ovakav tip konvolucionih slojeva.



Slika 24 Priprema Depthwise konvolucionog sloja za procesiranje PE modulima prilagođenim standarnim kernelima.

Nasuprot velikom broju arhitektura čiji se procesorski blokovi zasnivaju na dvodimenzionalnim matricama PE-ova, Argus arhitektura koristi jedan PE za računanje svih odviza koje generiše jedan kernel konvoluiran sa IFM-om. Ovo znači da je svaki PE odgovoran za izračunavanje rezultata jednog kanala OFM-a. Govoreći u terminima algoritma 3, svaki PE je zadužen za izvršavanje kompletne funkcije *calc_ofm_point*. Da bi navedeno bilo moguće, potrebno je da svaki PE ima odgovarajuće memorijske elemente za čuvanje kernela. U narednim poglavljima će biti detaljno opisana mikroarhitektura PE-ova, ali je bitno napomenuti da će se parametri kernela čuvati u jednodimenzionalnom nizu, što znači da će se izgubiti informacije o obliku kernela. Smeštanje parametara kernela u memorijeske blokove PE-ova je prikazano na slici 25.



Slika 25 Čuvanje parametara kernela unutar memorijskih ćelija PE-a.

Na slici 25 se može videti da se štapići kernela nadovezuju čime se dobija nezavisnost oblika kernela od arhitekture PE-a. Ovo je još jedna od prednosti Argus akceleratora u odnosu na neke od postojećih arhitektura zato što Argus ima mogućnost procesiranja širokog spektra kernela bez obzira na njihov oblik. Na kraju, napomenimo i to da se u okviru memorijskih blokova PE-ova smeštaju samo parametri kernela različiti od nule koje ćemo u nastavku označavati sa NZV (engl. *Non-zero values*). Pored NZV-ova, PE-ovima su potrebni i podaci o pozicijama NZV-ova unutar orezanih grupa kernela koje ćemo skraćeno zvati NZI (engl. *Non-zero index*).

Za uvođenje paralelizacije u procesiranje konvolucionih slojeva moguće je primeniti razmotavanje petlji algoritma 3. Prva optimizacija je razmotavanje petlje L1 sa faktorom PE_Num. U praksi, ovo znači da je potrebno imati PE_Num instanciranih PE-ova koji će paralelno računati više odziva konvolucije. Svi PE-ovi će procesirati isti deo IFM-a, samo sa različitim kernelima što znači da će se PE_Num kanala OFM-a računati istovremeno. Kao što

je već rečeno, PE_Num je potrebno pažljivo odabrati i u slučaju Argus akceleratora je on jednak 32. Ovo je optimalan broj PE-ova po jednoj instanci kako bi PE-ovi bili uposleni za veliku većinu slojeva jer uglavnom, svi imaju 32 ili više kernela. Sa druge strane, faktor paralelizacije od 32 često nije dovoljan kako bi se postigle željene performanse**.**

Kako bi se omogućila skalabilnost, Argus akcelerator je projektovan tako da se vrlo jednostavno može povezati nekoliko instanci u značajno moćniju arhitekturu. Sa stanovišta jednog jezgra, dodatna paralelizacija izgleda kao smanjenje ukupnog broja kernela u datom sloju, idealno sa faktorom broja konvolucionih jezgara. Na primer, ukoliko sloj ima 64 kernela, a koristi se instanca Argusa sa dva konvoluciona jezgra od po 32 PE-a, moguće je svakom jezgru dodeliti po 32 kernela kako bi se dvostruko smanjio broj iteracija petlje L1. Sa druge strane, ako sloj ima manje kernela nego PE-ova, na primer 32 kernela koja obrađuje Argus akcelerator sa dva konvoluciona jezgra (64 PE-ova), tada je moguće podeliti petlju L2 na dva dela. Gornja polovina IFM bi se izvršavala na jednoj instanci konvolucionog jezgra, dok bi donja polovina bila procesirana drugim konvolucionim jezgrom. Zbog komleksnosti, detaljno pojašnjenje procesiranja CNN-ova pomoću više konvolucionih jezgara je dato u poglavlju 6.7.

Drugu optimizaciju je moguće implementirati unutar *calc_ofm_point* i to razmotavanjem petlje L6. Razmotavanje ove petlje sa faktorom četiri neće imati uticaj na svestranost PE-ova u pogledu različitih oblika kernela koje PE može da procesira zato što se razmotava procesiranje po dubini kernela. Ukoliko neki kernel nema dubinu koja je umnožak broja četiri, moguće je dopuniti kernel nulama i time neznatno smanjiti performanse akceleratora. Primetimo i da ukoliko je mreža orezana, spolja gledano, svaki PE će procesirati osam tačaka IFM-a u jednom taktu. Ovo je posledica prirode algoritma za orezivanje koji u svakoj grupi od po osam tačaka kernela odstrani četiri što implicira da će PE imati dovoljno procesorske snage da u svakom taktu obradi preostale četiri tačke, odnosno, celu grupu od osam tačaka IFM-a. Gledano iz ugla krajnjeg korisnika, akcelerator će imati ukupan faktor paralelizacije jednak PE_Num • 8, ukoliko je mreža orezana. Algoritam 4 predstavlja pseudo kod u kojem su nad algoritmom 3 primenjene navedene optimizacije. Primetimo da se funkcija *calc_layer_ofm* razlikuje za slučaj kada je PE_Num veći od Kernel_Num u sloju (na primer dva konvoluciona jezgra koja procesiraju sloj sa 32 kernela) i kada je PE_Num manji od broja kernela u sloju (na primer, dva konvoluciona jezgra koja procesiraju 128 kernela). Kako bi se istakle razlike između dve implementacije funkcije *calc_layer_ofm*, iste su označene crvenom bojom.

ALGORITAM 4: Pseudo kod algoritma za računanje odziva konvolucionog sloja sa navedenim optimizacijama

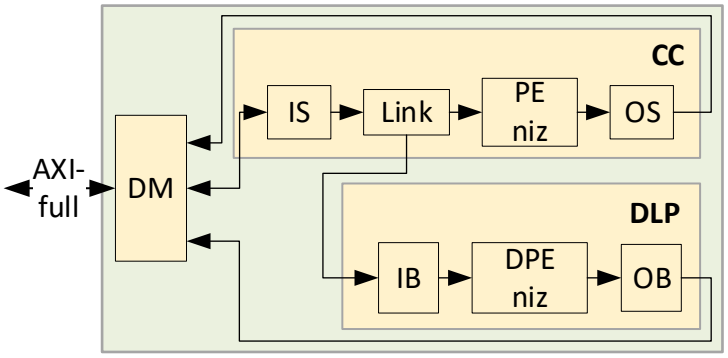
```
    func calc_layer_ofm(IFM, KM, nzi): - PE_Num > Kernel_Num
L1    for(fn = 0; fn < Kernel_Num; fn+=PE_Num)
L2      for(y = 0; y < OFM_Height; y = y+OFM_Height/conv_core_num)
L3        for(x = 0; x < OFM_Width; x++)
            ifm_h_part = y·Sv:(y·Sv + Kernel_Height)
            ifm_w_part = x·Sv:(x·Sv + Kernel_Width)
            ifm_bundle = IFM[ifm_h_part][ifm_w_part][:]
            OFM[x][y][fn] = calc_ofm_point(ifm_bundle, KM[fn], nzi)
            OFM[x][y][fn+1] = calc_ofm_point(ifm_bundle, KM[fn], nzi)
            ....
            OFM[x][y][fn+PE_Num-1]
                  = calc_ofm_point(ifm_bundle, KM[fn+PE_Num-1], nzi)

    func calc_layer_ofm(IFM, KM, nzi): - PE_Num < Kernel_Num
      kernel_groups = Kernel_Num / PE_Num
      kernel_num_per_group = Kernel_Num / kernel_groups
      for(i = 0; i < kernel_groups: i++)
L1      for(fn = 0; fn < kernel_num_per_group; fn+=PE_Num)
L2        for(y = 0; y < OFM_Height; y++)
L3          for(x = 0; x < OFM_Width; x++)
              ifm_h_part = y·Sv:(y·Sv + Kernel_Height)
              ifm_w_part = x·Sv:(x·Sv + Kernel_Width)
              ifm_bundle = IFM[ifm_h_part][ifm_w_part][:]
              OFM[x][y][fn] = calc_ofm_point(ifm_bundle, KM[fn], nzi)
              OFM[x][y][fn+1] = calc_ofm_point(ifm_bundle, KM[fn], nzi)
              ....
              OFM[x][y][fn+PE_Num-1]
                    = calc_ofm_point(ifm_bundle, KM[fn+PE_Num-1], nzi)

    func calc_ofm_point(ifm_bundle, km, nzi):
L4    for(kh = 0; kh < Kernel_Height; kh++)
L5      for(kw = 0; kw < Kernel_Width; kw++)
L6        while(kd < Kernel_Depth):
            if(network_pruned):
              nzv_terms =
                take_nzv_terms(ifm_bundle[kh][kw][kd:kd+8, nzi_pos)
              incr = 8
            else:
              nzv_terms = ifm_bundle[kh][kw][kd:kd+4]
              incr = 4
            ofm_point += nzv_terms[0]·km[kh][kw][kd]
            ......
            ofm_point += nzv_terms[3]·km[kh][kw][kd+3]
            kd += incr
    return ofm_point
```

Za razliku od algoritma 3, u algoritmu 4 se može uočiti paralelno računanje PE_Num OFM tačaka unutar petlje L3. Pored IFM-a i kernela, ulaz u *calc_layer_ofm* je proširen za NZI vrednosti u slučaju da je mreža orezana. Takođe, NZI parametri se prosleđuju i *calc_ofm_point* funkciji koja inkorporira drugu optimizaciju. Pošto se procesiranje unutar PE-a razlikuje u slučaju kada je mreža orezana i kada nije, *for* petlja L6 je zamenjena *while* petljom koja takođe prolazi kroz štapiće kernela po dubini. Ukoliko je mreža orezana korak u *while* petlji (*incr*) je osam dok je u slučaju neorezane mreže korak jednak četiri. Pored inkrementa *while* petlje, i odabir IFM tačaka koje ulaze u MAC jedinice je drugačiji u ova dva slučaja. U slučaju orezane mreže poziva se funkcija *take_nzv_terms* koja na osnovu argumenta *nzi_pos* odabira četiri tačke iz jedne grupe IFM-a. U slučaju neorezanog CNN-a (*else* grana) u *nzv_terms* se smeštaju četiri uzastopne vrednosti trenutnog štapića IFM-a.

Primetimo da Argus akcelerator sa 32 PE-a u jednom prolazu može da izračuna samo 32 kanala OFM-a, što znači da će za procesiranje sloja sa, na primer 128 kernela, biti potrebna četiri ciklusa petlje L1 u algoritmu 4 ukoliko se sloj obrađuje samo jednom instancom konvolucionog jezgra. U slučaju da postoji više konvolucionih jezgara, na primer dva, biće potrebno samo dva prolaza što se može uočiti analizom *calc_layer_ofm* funkcije kada je PE_Num manji od Kernel_Num.

6.2 Prikaz arhitekture na najvišem nivou apstrakcije

U ovom poglavlju je prikazana mikroarhitektura Argus CNN akceleratora na najvišem nivou apstrakcije. Poglavlje je podeljeno prema hardverskim celinama prateći tok podataka, što znači da su prvo opisani blokovi za učitavanje podataka iz DRAM-a, zatim procesorski elementi i na kraju blokovi zaduženi za smeštanje podataka u eksternoj memoriji. Na slici 26 se mogu uočiti tri osnovna bloka, *Data Mover* (DM) koji je spona između DRAM kontrolera i ostatka akceleratora, konvoluciono jezgro (engl. *Convolutional Core*, CC) i DLP jezgro (engl. *Dense Layer Processor*). Centralna procesorska jedinica je CC koji čine ulazni tok podataka (engl. *Input Stream*, IS), niz PE-ova i izlazni tok podataka (engl. *Output Stream*, OS). U nastavku uvodnog dela poglavlja će biti reči o osnovnim funkcionalnostima blokova unutar CC modula kao najbitnije jedinice akcelerator. Pored toga biće uvedene i skraćenice koje će se intenzivno koristiti u daljem opisu arhitekture. Slika 26 predstavlja blok dijagram arhitekture Argusa na najvišem nivou apstrakcije.



Slika 26 Arhitektura Argus akceleratora sa jednim CC modulom.

Osnovna funkcionalnost IS modula je distribucija podataka od eksternog DRAM-a, preko DM modula, do PE-ova. Ovaj proces se može podeliti na nekoliko manjih operacija i to na: potraživanje odgovarajućih podataka od DRAM kontrolera, keširanje delova IFM-a, i slanje štapića IFM-a ka PE-ovima za računanje konvolucionih rezultata. Keširanje je izuzetno bitno kako bi se postigla visoka efikasnost PE-ova, a tako i kompletnog akceleratora. Pored navedenog, IS je zadužen za učitavanje parametara kernela i pozicija elemenata različitih od nula koje koristi BZPNP kako bi selektovao odgovarajuće tačke IFM-a.

Centralna jedinica CC bloka je niz PE-ova koga čine memorijske banke za NZV, NZI, 32 PE-a i blok za prikupljanje rezultata koje generišu PE blokovi. Na početku procesiranja konvolucionog sloja, PE niz dobija NZV i NZI od IS modula da bi potom, IS počeo da učitava odgovarajuće delove IFM-a. Po završetku izračunavanja prvih rezultata konvolucije, izlazni blok PE niza postavlja rezultate PE-ova na izlaz u *Round Robin* (RR) maniru, od PE-a na poziciji 0 do PE-a na poziciji 31 ili do poslednjeg uposlenog PE-a u zavisnosti od broja kernela u sloju. Kako bi se širina rezultata redukovala na 16-bita, na izlazu PE niza postoji i blok koji vrši zaokruživanje rezultata u željenom formatu.

Na kraju ovog lanca nalazi se OS blok koji generiše odgovarajuće adrese u okviru DRAM-a na koje treba smestiti rezultate koje generiše PE niz. Uz to, OS sadrži i modul koji primenjuje aktivacione funkcije na rezultate u zavisnosti od konfiguracije sloja.

Modul koji služi za povezivanje više CC blokova ili CC sa DLP modulom se zove *Link* i o njemu će više reči biti u zasebnom delu poglavlja.

DLP jezgro akcelerira ne konvolucione slojeve CNN-ova, sa akcentom na *max pooling*, *average pooling* i sloj sabiranja. DLP modul je visoko optimizovan za navedene slojeve koji, iako

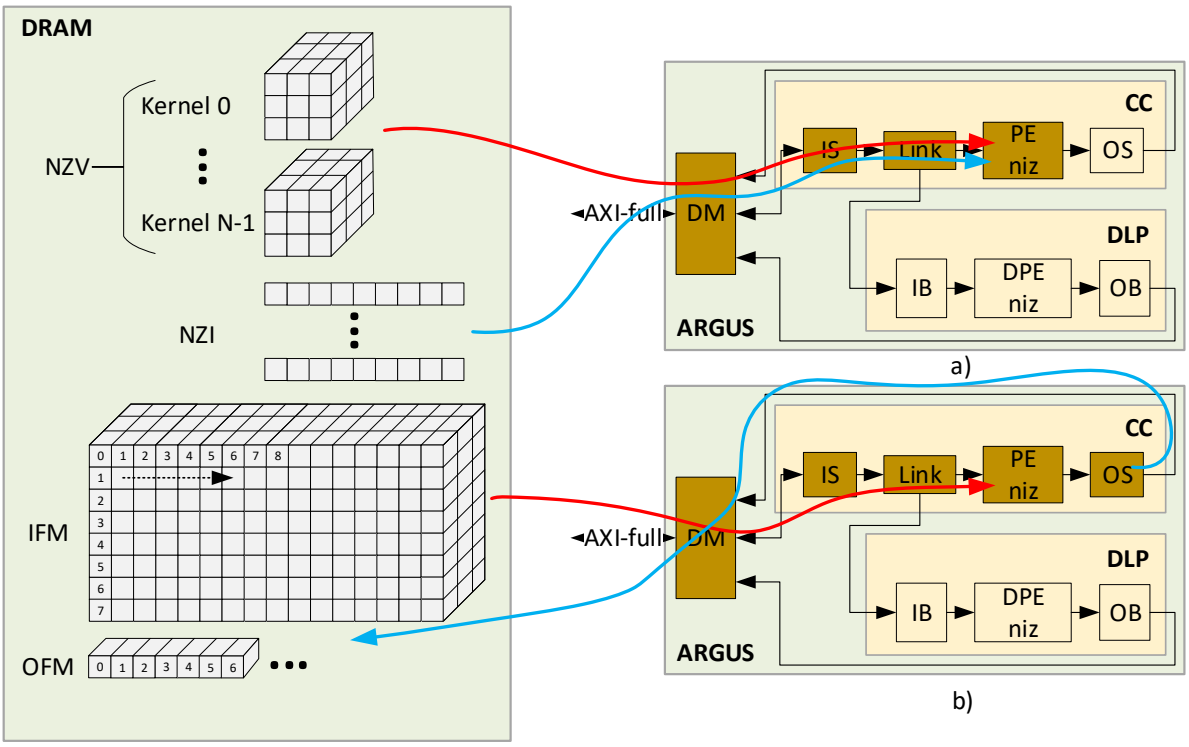
zauzimaju značajno manji procenat vremena procesiranja slika pomoću CNN-ova, ipak mogu da imaju značajan uticaj ukoliko se konvolucioni slojevi izvršavaju izuzetno efikasno. U ekstremnom slučaju, mogu da postanu dominantni, ako je akcelerator visoko optimizovan za obradu konvolucionih slojeva. Svakako, softversko izvršavanje pomenutih slojeva bi zahtevalo više vremena nego što je Argus akceleratoru potrebno da obradi konvolucione slojeve te bi performanse bile značajno degradirane. Na visokom nivou apstrakcije, DLP jezgro se može podeliti na ulazni bafer (IB), DPE niz (engl. *Dense Processor Element*) i izlazni bafer (OB). Ulazni i izlazni bafer čine registri i standardne memorijske ćelije dok je DPE izgrađen oko sabirača, komparatora i množača potrebnih posebno u slučaju *average pooling* slojeva.

6.3 Uobičajeni tok podataka prilikom procesiranja konvolucionog sloja

U nastavku je prikazan uobičajeni tok podataka prilikom procesiranja standardnih konvolucionih slojeva, kako bi se jasnije uočila uloga svih blokova na visokom nivou.

Na slici 27a) je prikazan prvi korak u procesiranju, to jest, učitavanje NZV i NZI podataka u PE niz. Aktivni blokovi akceleratora su naznačeni tamnom nijansom. Transakcije preko DM-a inicira IS tako što zatraži NZV podatke kernela. Potom DRAM kontroler odgovara i isporučuje zahtevane podatke preko DM-a kroz IS i *Link* module ka PE nizu. Ovaj tok podataka je naznačen crvenom strelicom. Naravno, u slučaju orezanog sloja samo parametri kernela različiti od nula se šalju iz DRAM-a do PE niza. Po završetku učitavanja NZV-ova IS detektuje da je memorijski port slobodan i šalje zahtev za NZI koji se istom putanjom učitavaju do PE niza (plava strelica).

Sledeći korak je učitavanje IFM-a i početak procesiranja konvolucionog sloja. Aktivni delovi akceleratora kao i smer toka podataka su prikazani na slici 27b). U ovom procesu, IS zahteva podatke IFM-a preko DM-a. Učitani delovi IFM-a se smeštaju unutar keš memorije IS modula koji ih potom distribuira do PE niza. Zbog ponovnog korišćenja učitanih delova IFM-a, IS neće zahtevati nove štapiće iz DRAM-a dokle god se ne oslobodi mesto u keš memoriji. Paralelno sa opisanim, PE niz računa odzive i inicira njihovo skladištenje u DRAM-u. Ovo se, takođe, odvija preko DM-a čiji AXI-full protokol ima mogućnost istovremenog čitanje i pisanja u DRAM preko istog porta.



Slika 27 Učitavanje NZV, NZI podataka iz DRAM-a na početku procesiranja konvolucionog sloja (a) i računanje rezultata konvolucije (b).

U slučaju obrade drugih slojeva, podaci će se preko *Link* modula proslediti DLP jezgru čiji će OB (engl. *Output buffer*) smeštati podatke u DRAM, slično kao što to radi OS u CC modulu.

6.4 Ulazni tok podataka (IS)

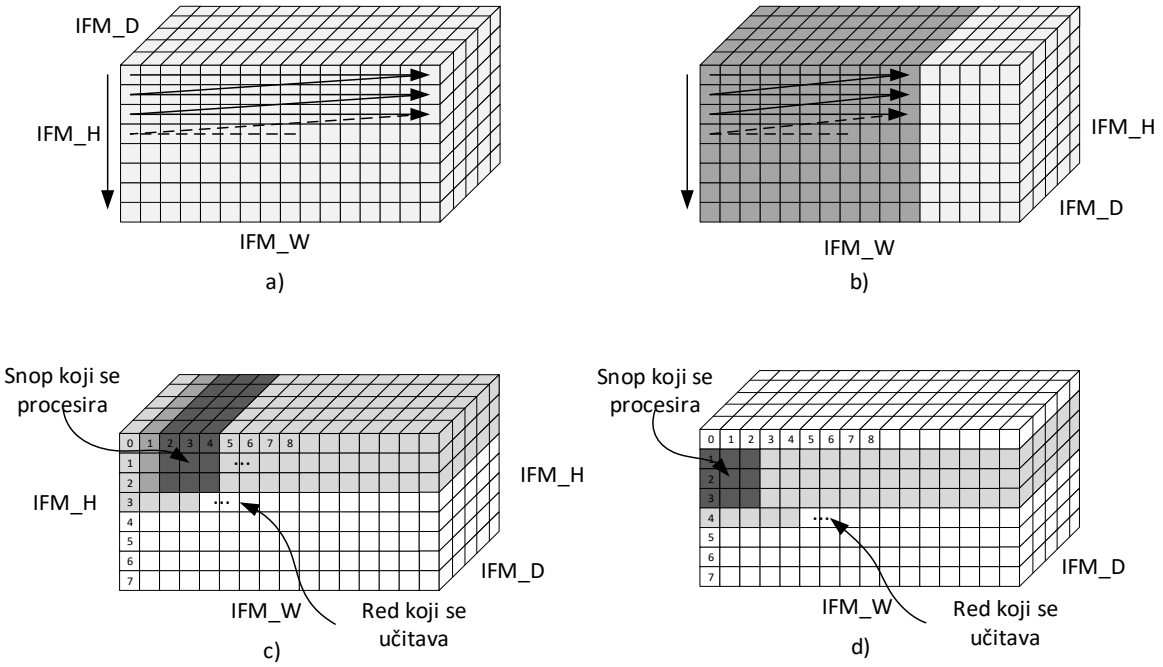
6.4.1 Opis funkcionalnosti – mehanizam keširanja IFM-a

Uz PE-ove, IS predstavlja ključnu komponentu za postizanje visoke efikasnosti akceleratora. Navedeno proizilazi iz činjenice da su konvolucioni slojevi izuzetno zahtevni kako u pogledu procesorske snage tako i u pogledu propusne moći eksternih memorijskih sistema. Procesorska snaga se uvećava instancioniranjem većeg broja PE-ova, što neretko rezultuje uvećanju zahteva u pogledu propusne moći DRAM kontrolera. Da bi se smanjio uticaj karakteristika memorijskog sistema na performanse akceleratora, moguće je razviti lokalni keš koji će služiti za privremeno čuvanje dela IFM-a koji se trenutno procesira u internoj memoriji akceleratora.

Prethodno publikovane ideje u radovima [11] i [12] su korišćenje kao osnova za kreiranje efikasnog, ali jednostavnog mehanizma keširanja IFM-a. Ukoliko je raspoloživa keš memorija dovoljne veličine mehanizam opisan u nastavku osigurava da se svaki štapić IFM-a učitava

samo jednom iz DRAM-a ma koliko puta on bio korišćen prilikom procesiranja. U slučaju izuzetno limitiranih memorijskih resursa, ovo pravilo će biti narušeno, ali bez prevelikog uticaja na krajnje performanse keš sistema.

Za opis rada algoritma keširanja uzmimo da su visina kernela (engl. *Kernel Height*, KH) i širina kernela (engl. *Kernel Width*, KW) jednake i iznose tri. Takođe, uzmimo da je vertikalni i horizontalni korak (engl. *Stride*) takođe jedan. Na početku obrade IFM-a, IS generiše zahtev za učitavanje IFM štapića i njihovo smeštanje unutar keša. Učitavanje počinje iz gornjeg levog ugla IFM-a i nastavlja kroz ceo prvi red kao što je prikazano na slici 28a). Nakon što je učitano onoliko redova IFM-a kolika je i visina kernela (KH), počinje procesiranje tako što IS šalje potrebne snopove štapića ka PE-ovima. Početak procesiranja, odnosno, treći snop koji se procesira je prikazan na slici 28c). Za navedenu konfiguraciju jasno je da se na slici 28c) vrednosti štapića u koloni dva IFM-a, koriste već treći put. Prvi put su bili korišćeni kada je procesiran prvi snop koji obuhvata kolone 0, 1 i 2. Drugi put prilikom računanja odziva na drugi snop, to jest, snop koji obuhvata kolone 1, 2 i 3 i na kraju, treći put, na naznačenom snopu.



Slika 28 Učitavanje IFM-a (a), učitavanje kada keš memorija nije dovoljna (b), procesiranje (c) i oslobađanje keš memorije (d).

Po završetku procesiranja prvog reda IS počinje da šalje snopove iz drugog reda kako je to prikazano na slici 28d). Primetimo da se sada, za procesiranje prvog snopa u drugom redu, ponovo koriste kolone označene indeksima 0, 1 i 2 i njihovi štapići u redovima 1 i 2. U slučaju štapića u koloni 0, oni se sada koriste drugi put, štapići u koloni 1 treći put, a u koloni 2 četvrti.

Daljom analizom bismo došli do zaključka da se samo štapići na rubu IFM-a koriste manji broj puta, dok idući ka centru IFM-a ponovno korišćenje je veće i kreće se do 9 puta u slučaju kernela veličine 3x3 i koraka konvolucije 1. Prilikom prelaska u drugi red, IS oslobađa memoriju keša zauzetu štapićima prvog reda i na njihovo mesto nastavlja učitavanje štapića iz reda označenog indeksom 4 na slici 28d). Kako bi se izbegli zastoji u procesiranju prilikom prelaska na naredne redove snopova IFM-a, potrebno je unapred učitati štapiće iz narednih redova. Broj redova koji se mora učitati pre nego se trenutni završi odgovara *Stride*-u i u primeru na slici 28 iznosi jedan. Kada je reč o dimenzijama keš memorije potrebno je da keš bude visine KH + Stride, a širine IFM_W. Da bi se izračunala potrebna količina memorije unutar keša potrebno je još znati i kolika je dubina IFM-a. Izraz 3 predstavlja formulu po kojoj se može izračunati veličina keš memorije izražena u broju tačaka IFM-a:

$$Veličina\ keša = IFM\_W \bullet (\mathrm{KH} + \mathrm{Stride}) \bullet \mathrm{IFM\_D} \tag{3}$$

Prema podacima objavljenim u radu [11] ovakav pristup keširanju redukuje potrebnu propusnu moć DRAM kontrolera i do 9 puta u odnosu na akceleratore koji nemaju mehanizam keširanja za slučaj kada je kernel veličine 3x3, a korak konvolucije 1.

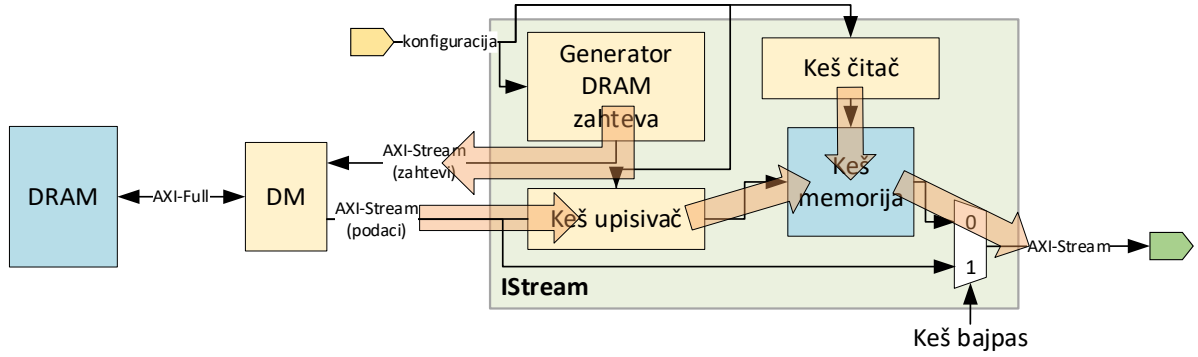
Ipak, da bi se isključila zavisnost veličine keš memorije od CNN model koji se akcelerira, potrebno je jedan od parametara iz izraza (3) učiniti konfigurabilnim. Kako sva četiri zavise isključivo od parametara sloja CNN-a i nije moguće uticati na njih direktno. Sa druge strane, moguće je procesirati IFM po uzdužnim trakama (engl. *Stripe*) kako je to prikazano na slici 28 (b). Tamno sivom bojom je označen deo koji se trenutno procesira dok će neosenčeni deo IFM-a biti obrađen u drugom prolazu. Ovakav pristup zadržava najveći deo benefita prethodno opisanog keširanja. Jedina mana je ponovno učitavanje štapića na liniji razdvajanja ovih vertikalnih delova IFM-a. Štapići će biti ponovo učitani iz DRAM-a prilikom obrade neosenčenog dela IFM-a na slici 28b). Na kraju, ovo ne predstavlja značajnu degradaciju jer je reč o nekoliko štapića IFM-a naspram širine IFM-a koja je reda do nekoliko stotina štapića. Navedena mana je svakako prihvatljiva kako zbog svog malog uticaja na performance tako i zbog velikog benefita koji proizilazi iz ovakvog načina procesiranja, a to je, nezavisnost određivanja veličine keš memorije od CNN modela koji će biti akceleriran. Sada se izraz 3 može napisati kao:

$$Veličina\ keša = Stripe\_W \bullet (\mathrm{KH} + \mathrm{Stride}) \bullet \mathrm{IFM\_D} \tag{4}$$

Drugim rečima, promenljiva vrednost parametra Stripe_W i nezavisnost istog od širine IFM-a daje mogućnost da se iskoriste pogodnosti keširanja i na FPGA sistemima sa izuzetno skromnim memorijskim resursima.

6.4.2 Mikroarhitektura

Ulazni tok podataka čine četiri osnovne komponente: generator DRAM zahteva, keš upisivač, keš čitač i keš memorija. Pored navedenih komponenti, na slici 29 se može primetiti da postoje i četiri interfejsa koja se koriste za komunikaciju sa okružujućim komponentama. Uz konfiguracioni, postoje još dva interfejsa koja služe za dopremanje podataka iz DRAM-a, i to: jedan preko koga se šalju zahtevi DM-u, i drugi, koji prihvata tražene podatke iz DRAM-a. Interfejsi koji prihvataju, odnosno, distribuiraju podatke su širine 128 bita. Izlazni interfejs šalje podatke ka *Link* modulu, koji dalje distribuira NZI, NZV ili IFM ka PE nizu i/ili drugom CC, odnosno, DLP modulu. Primetimo da je transparentnim strelicama prikazan uobičajeni tok podataka unutar IS modula u slučaju kada je keširanje aktivno. Generator DRAM zahteva potražuje podatke od DRAM-a, koji odgovara keš upisivaču, koji dalje upisuje podatke u keš memoriju. Upisane podatke iz keš memorije isčitava keš čitač te se izlaznom strelicom oni dalje prosleđuju narednim blokovima.



Slika 29 Blok dijagram ulaznog toka podataka.

Konfiguracioni interfejs doprema parametre tipične za trenutni sloj koji se procesira. Lista podataka potrebnih za konfiguraciju kao i opis svakog porta je prikazana u tabeli 5.

Naziv konfiguracionog porta	Opis
start	Puls koji IS dobija od centralne mašine stanja da treba da započne procesiranje. Puls označava da se na svim konfiguracionim portovima nalaze odgovarajući podaci potrebni za obradu trenutnog sloja.

stick_depth	Označava dubinu štapića, to jest, dubinu kernela koja se trenutno procesira. Pošto se nekad neće procesirati kompletna dubina IFM-a u jednom prolazu, Stick_depth nije isto što i Ifm_depth u opštem slučaju.
ifm_height	Visina IFM-a.
ifm_width	Širina IFM-a.
ifm_depth	Dubina IFM-a.
pad_top	Broj štapića koji označava *zero padding* na vrhu IFM-a.
pad_bot	Broj štapića koji označava *zero padding* na dnu IFM-a.
pad_left	Broj štapića koji označava *zero padding* na levoj strani IFM-a.
pad_right	Broj štapića koji označava *zero padding* na desnoj strani IFM-a.
kernel_h	Visina kernela.
kernel_w	Širina kernela.
stride_horizontal	Horizontalni korak konvolucije.
stride_vertical	Vertikalni korak konvolucije.
bypass_cache	U nekim slojevima se zaobilazi mehanizam keširanja. Ukoliko je ovo slučaj baypass_cache ima vrednost 1.
load_ifm_at_once	Po potrebni, kompletan IFM može biti zatražen samo jednim zahtevom kako bi se iskoristila maksimalna propusna moć DRAM kontrolera. Ovo je moguće samo u odgovarajućim slojevima, kao što je *pointwise* tip konvolucije.
stripe_ff	Broj bajtova koje je potrebno preskočiti u DRAM-u u slučaju da se ne procesira kompletna širina IFM-a usled nedostatka memorije.
cache_height	Visina keš memorije (visina u broju štapića IFM-a).
cache_size	Veličina keš memorije.
stripe_width	Označava širinu dela IFM koji se trenutno procesira ukoliko nema mogućnosti za smeštanje cele širine u keš.
ifm_stick_to_load	Ukupan broj štapića IFM-a koje treba učitati za dati sloj.
bias_addr	Početna adresa bias-a u DRAM-u.
nzi_addr	Početna adresa NZI u DRAM-u.
nzv_addr	Početna adresa NZV u DRAM-u.
ifm_addr	Početna adresa IFM u DRAM-u.
total_bias	Ukupan broj bias-a.
total_nzi	Ukupan broj NZI-eva.
total_nzv	Ukupan broj NZV-ova.
total_ifm	Ukupan broj IFM tačaka koje treba učitatio.
use_input_link	U slučaju da se koristi Link modul, nije potrebno da IS modul bude aktivan u toku učitavanja IFM-a što se indikuje jedinicom na ovom portu.

Tabela 5 Opis konfiguracionih portova ulaznog toka podataka.

Uz navedene konfiguracione portove, IS ima i parametar CACHE_MEM_LOC kojim je, pre implementacije, moguće konfigurisati veličinu keš memorije u zavisnosti od dostupnih resursa ili nekih drugih ograničenja.

6.4.2.1 Generator DRAM zahteva

Po konfigurisanju akceleratora, generator DRAM zahteva počinje da šalje komande ka DM-u potražujući prvo, *bias*-e, zatim NZI i NZV vrednosti i na kraju IFM po potrebi. Pre opisa formata komande, treba napomenuti da se ista ka DM-u šalje preko AXI-Stream [48] interfejsa. Potom

se komanda konvertuje u AXI-Full zahtev ka DRAM kontroleru koji preko istog AXI-Full interfejsa odgovara slanjem traženih podataka iz DRAM-a ka DM-u. Dalje, DM prosleđuje akceleratoru prihvaćene podatke, ali preko AXI-Stream interfejsa. Dakle, DM konvertuje protokol iz AXI-Full od DRAM-a do DM-a, u AXI-Stream od DM-a ka akceleratoru. Manipulacija interfejsima je naznačena na slici 29. Format komande, prikazan u tabeli 6, je uslovljen oblikom koji je definisan od strane kompanije Xilinx čiji je DM IP (engl. *Intellectual Property*) [49].

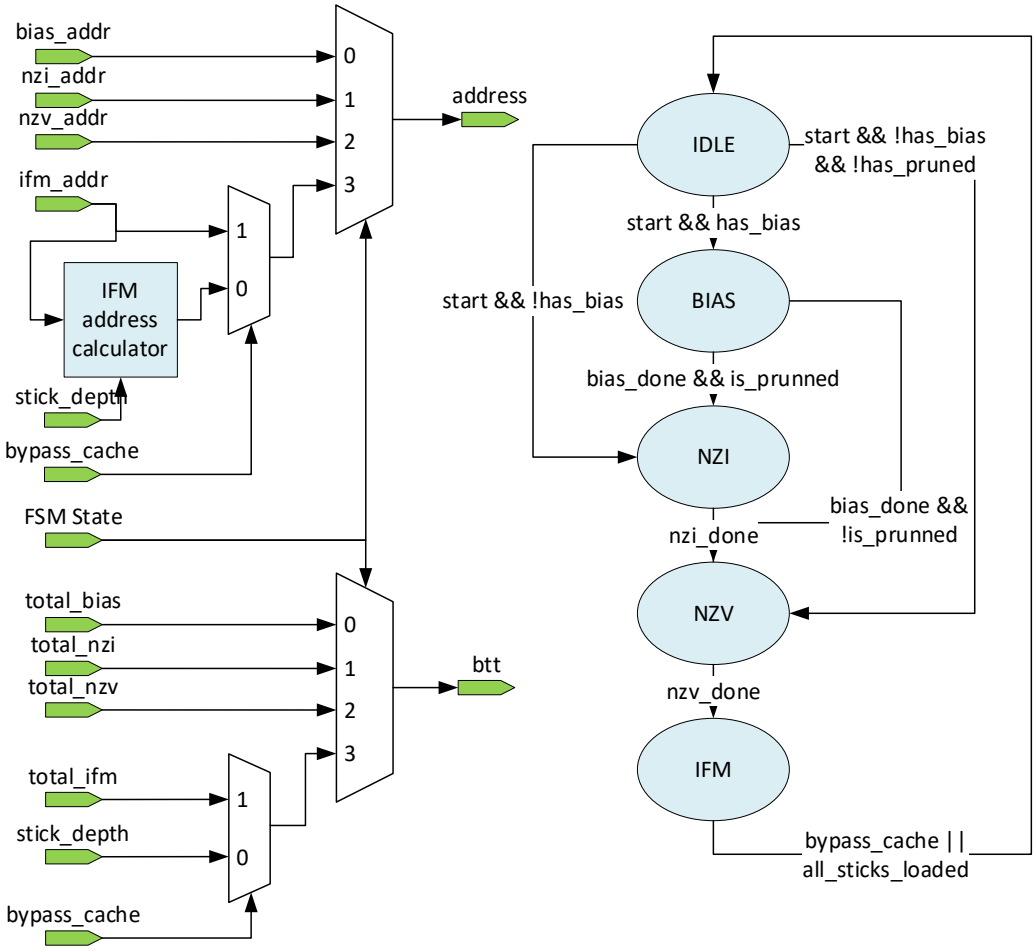
xCACHE	xUSER	RSVD	TAG	SADDR	DRR	EOF	DSA	Type	BTT
0	0	0	0	adresa	0	1	0	1	veličina

Tabela 6 Format komande koju prihvata DM.

U navedenoj komandi polja od interesa su SADDR, EOF, TYPE i BTT. Ostala polja je potrebno konfigurisati kao nula prilikom svake transakcije. SADDR predstavlja 32-bitnu vrednost adrese u DRAM-u počevši od koje se traže podaci. EOF postavljen na 1 će omogućiti da prilikom isčitavanja poslednjeg zatraženog podatka bude asertovan TLAST signal AXI interfejsa. TLAST informacija će dalje biti korišćena od strane akceleratora, najčešće kao indikacija poslednjeg podatka u traženom štapiću IFM-a. Takođe, koristi se i kao indikator da su odgovarajući NZV/NZI podaci učitani. Polje Type postavljeno na 1 označava da je traženi zahtev inkrementalnog tipa u smislu adresa. Na primer, ukoliko se zatraži 8 bajta počevši od adrese 10, to znači da će se prilikom transakcije svakog bajta adresa uvećavati, to jest, biće poslati podaci sa adresa 10-17. U slučaju da je Type postavljen na vrednost 0, adresa čitanja se ne bi inkrementirala. Na kraju, polje BTT označava veličinu bloka podataka koji se traži izraženu u bajtima. Ukoliko su podaci poravnati u DRAM-u tako da svaki niz počinje na adresi koja je umnožak broja 16, svi navedeni parametri osim SADDR i BTT treba da budu fiksirani kao što je to navedeno u tabeli 6. SADDR i BTT će biti generisani različito za svaki zahtev.

Sa stanovišta hardverske implementacije, blok za generisanje DRAM zahteva čini jedna mašina stanja (engl. *Finite State Machine*, FSM) koja upravlja jedinicama za generisanje adresa i BTT polja. Na slici 30 je predstavljena mašina stanja i blok za generisanje adresa, odnosno BTT polja. FSM čine pet stanja počevši od IDLE stanja u kome se FSM nalazi posle reseta ili po završetku procesiranja sloja. Po aktiviranju *start* signala, FSM prelazi iz IDLE stanja u BIAS, NZI ili NZV stanje. Ovaj prelaz je definisan time da li sloj ima *bias*-e (*has_bias*), odnosno, da li je mreža orezana (*is_pruned*). Po završetku učitavanja *bias*-a, FSM prelazi u NZI ili NZV stanje u zavisnosti od toga da li je CNN orezan ili ne. U NZI stanju se zahtevaju NZI podaci iz DRAM-a.

Nezavisno od *has_bias* i *is_pruned*, FSM će na kraju stići do stanja NZV u kome se učitavaju parametri kernela. Po završetku učitavanja parametara konvolucionog sloja, FSM prelazi u IFM stanje. U ovom stanju najčešće se traže štapić po štapić IFM-a iz DRAM-a. Po završetku učitavanja svih štapića, FSM prelazi u IDLE stanje. Sa druge strane, ukoliko je potrebno bajpasovati keš (na primer, u slučaju procesiranja *pointwise* konvolucionog sloja), FSM će zatražiti kompletan IFM samo jednim zahtevom i preći u IDLE stanje. Ukoliko se pak koristi *Link* modul za dopremanje IFM-a, to znači da Argus akcelerator ima više CC modula i da trenutni CC koristi podatke koje učitava IS nekog drugog CC bloka te će FSM po ulasku u IFM stanje, odmah, bez traženja dodatnih zahteva preći u IDLE stanje.



Slika 30 Mikroarhitektura bloka za generisanje DRAM zahteva.

Uprošćena logika za generisanje adresa i BTT-a, prikazana na slici 30, ima dva multipleksera 4-na-1 koja su kontrolisana stanjem FSM-a (*FSM State*). U slučaju da se adrese IFM-a zahtevaju štapić po štapić, potrebno je uključiti *IFM address calculator* blok koji će svaku narednu adresu zahteva uvećavati za dubinu štapića (*stick_depth*). Za generisanje BTT-a se koristi drugi 4-na-

1 multiplekser, takođe, kontrolisan trenutnim stanjem FSM-a. U slučaju da se u jednom zahtevu potražuje kompletan IFM, dodatni multiplekser 2-na-1 će umesto *stick_depth* proslediti *total_ifm* kao BTT i time zatražiti ceo IFM.

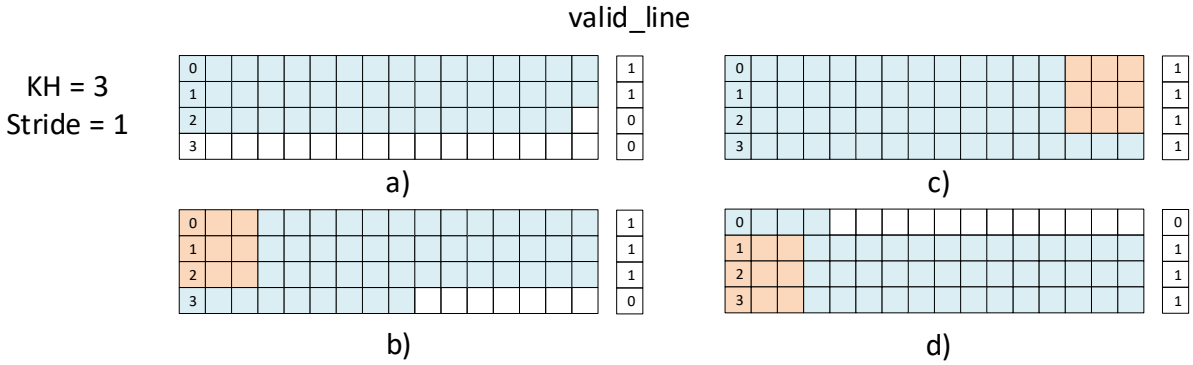
Jednostavna arhitektura generatora DRAM zahteva je postignuta njegovim razdvajanjem od toka podataka koji pristižu iz DRAM-a. Ovo je moguće zato što DM ima bafer komandi koje prima i signalizira *ready* signalom u okviru AXI-Stream protokola kada može da primi novu komandu. Komandni bafer DM-a se prazni po realizovanju svakog zahteva, ali je kompletan mehanizam zatvoren u DM-u što znači da nije potrebna dodatna interakcija Argus akceleratora. Napomenimo i to da je redosled učitavanja svih podataka u sloju unapred poznat što implicira da je i generisanje komandi unapred poznato.

6.4.2.2 Keš memorija

Keš memorija je realizovana kao klasična dvo-portna memorija konfigurabilne dubine i širine. Kako bi se iskoristile maksimalne performanse interfejsa koji dopremaju podatke ka i od keša, memorija mora da ima pristupe širine koji odgovaraju navedenim interfejsima. U slučaju Argus arhitekture širine su 128-bita, što znači da su i lokacije u memoriji 128-bita. Uparivanjem širina se omogućava upis/čitanje 128-bita podataka u jednom taktu. Pošto se radi sa 16-bitnom aritmetikom, to suštinski znači da je svaka lokacija dovoljna za smeštanje osam podataka. Kontrolne signale keš memorije, kao što su, adresa, *wr_en*, *rd_en* i ostale, kontrolišu keš upisivač/čitač koji su detaljno opisani u nastavku.

Uz klasičnu memoriju za čuvanje podataka (delova IFM-a), keš memorija sadrži i jednostavnu statusnu logiku koja radi po principu opisanom na slici 31 (*valid_line*). U ovom primeru je uzeto da je visina kernela (KH) jednaka tri dok je korak (*Stride*) jedan. Posle reseta, memorija se puni štapićima kao na slici 31a). Plavom bojom su naznačeni već upisani štapići. Svaki red IFM-a ima pridružen po jedan statusni registar, koji je indikator da li je red pun ili prazan. Na slici 31a) može se uočiti da su prva dva reda upisana u potpunosti te je *valid_line* indikator za ove redove postavljen na 1. Redovi 2 i 3 nisu upisani te su njihovi indikatori validnosti jednaki 0. U terminima standardnih keš memorija, *cache hit* se dešava ukoliko čitač čita bilo koji štapić iz reda čiji *valid_line* bit ima vrednost 1, a *cache miss* ukoliko je vrednost ovog bita jednaka 0. Po upisu poslednjeg štapića u redu 2 uključuje se keš čitač koji isčitava prvi snop štapića označen narandžastom bojom na slici 31b) i šalje ga ka nizu PE-ova. Primetimo da se red 3 učitava dok se prva tri procesiraju. Po završetku upisa reda 3, svi *valid_line* indikatori su

postavljeni na 1 i dalji upis se stopira kao na slici 31c). Po završetku procesiranja prva tri reda štapića, keš čitač prelazi na redove 1, 2 i 3, slika 31d). U trenutku prelaska u sledeći red, oslobađa se red 0 i nastavlja se dalji upis štapića u ovaj red kao na slici 31d). Postavljanje i brisanje indikatora vrše keš upisivač i keš čitač respektivno.



Slika 31 Upisivanje u keš memoriju po snopovima, validacija i oslobađanje redova keša.

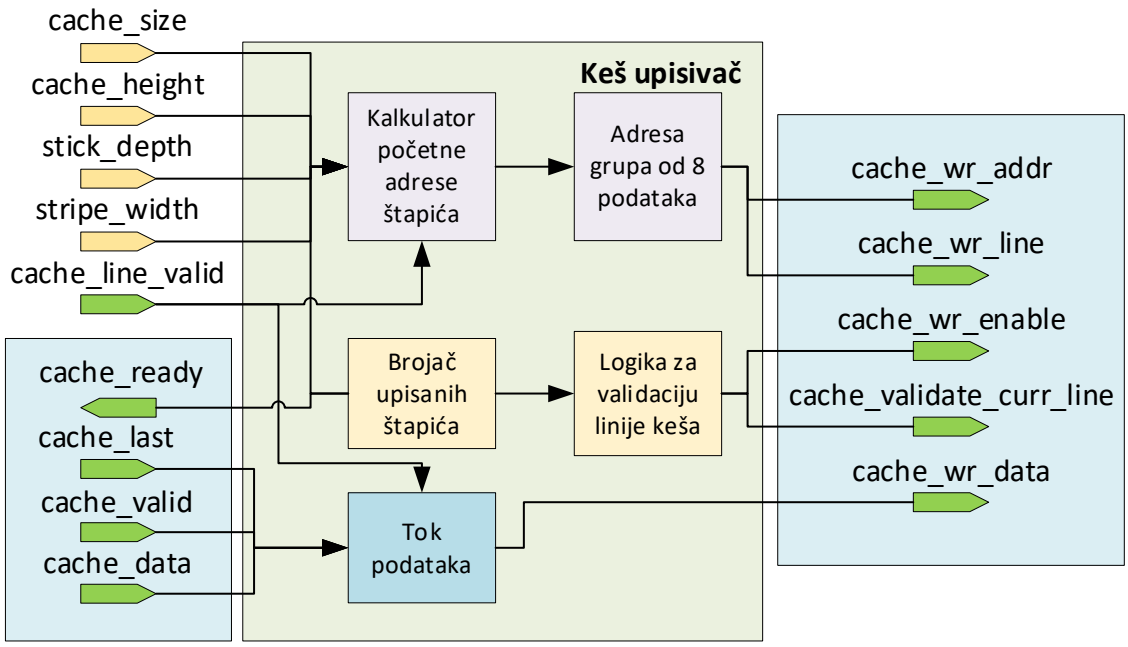
Memorija nije realizovana kao monolitni blok iz razloga što ćelije takve veličine nisu dostupne na današnjim FPGA platformama. Maksimalna širina podataka, sa kojima dostupne memorijske ćelije (BRAM) na Xilinx FPGA platformama mogu da rade, je 64-bita. Dodatno postoji još po 8 bita u svakoj memorijskoj lokaciji koji se najčešće koriste za specifične namene [50]. U konfiguraciji kada je port širine 64-bita, jedan BRAM ima 512 lokacija, ukupno 4kB. Da bi se formirala memorija širine 128-bita, potrebno je paralelno povezati dva BRAM-a. Ukoliko je potrebno implementirati veći broj lokacija, pored dodatnih BRAM-ova mora se uvesti i adresni dekoder i multiplekser na izlazu. Kako bi sve radilo na visokim frekvencijama, izlaz navedenog multipleksera je registrovan.

Primetimo da je granularnost keš memorije velika i da je korak sa kojim se povećava dubina jednak 512 lokacija. Alat često uspeva da optimizuje implementaciju i da smanji granularnost na 256 lokacija jer BRAM-ovi fizički imaju po dva porta za čitanje/upis. Granularnost treba uzeti u obzir prilikom određivanja parametra broja lokacija u memoriji. Čak i ukoliko postoji mogućnost da se veličina smanji za određeni broj lokacija, to neće nužno dovesti do smanjenja potrošnje raspoloživih resursa kao u slučaju ASIC implementacije.

6.4.2.3 Keš upisivač

Keš upisivač prihvata podatke iz DRAM-a dopremljenih preko DM-a i porta za podatke (slika 31). Pokretanje rada keš upisivača trigeruju validni podaci (*cache_valid*) na interfejsu

podataka. Svaki novi podatak uvećava postojeće brojače prema definisanim koracima od strane konfiguracionog interfejsa. Ovo znači da keš upisivač ne zahteva dodatnu kontrolu sa višeg hijerarhijskog nivoa. Ovakav pristup je moguć zbog unapred poznatih redosleda upisa podataka u keš memoriju na osnovu trenutne konfiguracije sloja. Pojednostavljen blok dijagram keš upisivača je prikazan na slici 32. Kao konfiguracione ulaze upisivač koristi veličinu keša (*cache_size*), visinu keša (*cache_height*) koja je jednaka KH + Stride, dubinu štapića izraženu u grupama od po 8 IFM tačaka (*stick_depth*), širinu IFM-a ili dela IFM-a koji se procesira (*stripe_width*) i status o zauzetim linijama keša (*cache_line_valid*). Ulaz za podatke je tipa AXI-Stream protokola, dok je izlazni interfejs ka keš memoriji takav da se bez izmena može povezati na BRAM interfejs.



Slika 32 Blok dijagram keš upisivača.

Interna struktura upisivača se može podeliti na četiri glavne celine, kalkulator početne adrese svakog štapića, adrese za trenutnu grupu od osam tačaka IFM-a, brojač upisanih štapića i logiku za validaciju linija keša. Kalkulator početne adrese štapića inkrementira adresu na kraju upisa svakog štapića. Adresa se inkrementira za dubinu štapića (*stick_depth*). Kako bi se formirale stvarne adrese za upis, potrebno je da se na početnu adresu štapića doda adresa trenutne grupe koja se upisuje. Sabrane adrese grupe i početne adrese štapića se prosleđuju keš memoriji preko *cache_wr_addr* porta. Ukoliko je nov podatak spreman za upis, uz formiranu adresu se prosleđuje i *cache_wr_enable* koji predstavlja signal za upis trenutnog

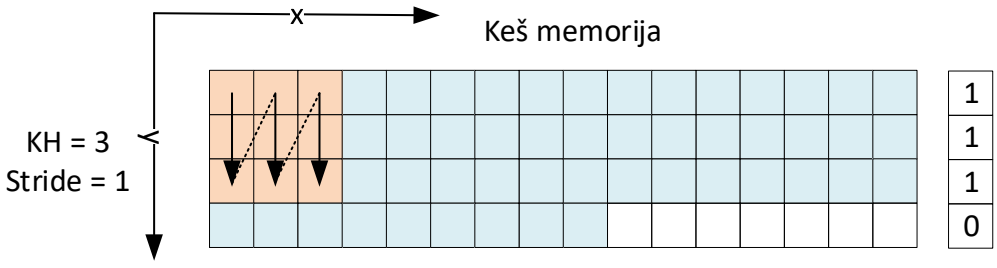
podata (*cache_wr_data*) u keš memoriju. Uz sve navedeno, kalkulator početne adrese štapića zna kada je upisan poslednji štapić u poslednjem redu keša i počinje ponovo da upisuje od adrese nula. Ovo je definisano konfiguracionim portom označenim kao *cache_size.*

Drugi deo keš upisivača, brojač upisanih štapića i logika za validaciju linija keša, kontrolišu *cache_validate_curr_line* port. Na kraju upisa svakog štapića se proverava da li je broj upisanih štapića u redu jednak *stripe_width*. Ukoliko jeste, potrebno je validirati trenutnu liniju u keš memoriji. Napomenimo i to da se kompletan mehanizam zaustavlja ukoliko trenutna linija keša nije slobodna za upis što se može odrediti na osnovu statusnog signala *cache_line_valid*.

Na kraju je potrebno izvršiti sinhronizaciju podataka sa generisanim adresama i ostalim komandama koje se šalju ka memoriji što je uloga bloka naziva tok podataka. Ovde nije reč o sinhronizaciji više takt domena već kašnjenju podataka tako da se na ulazu u memorijski blok podaci i odgovarajuće adrese nađu u istom taktu.

6.4.2.4 Keš čitač

Keš čitač predstavlja najkompleksniji modul u okviru IS bloka zadužen za isčitavanje potrebnih štapića u trenutnom snopu koji se procesira. Uz to keš čitač ima logiku koja oslobađa redove keš memorije koji više nisu potrebni za procesiranje. Kao i keš upisivač, i keš čitač nije kontrolisan sa više instance. Razlog je identičan, redosled čitanja je unapred poznat te je čitaču dovoljno da primi start komandu kao indikator da je konfiguracija na interfejsu validna i da isčitavanje može da počne. Redosled čitanja štapića iz memorije je prikazan na slici 33.



Slika 33 Redosled čitanja štapića iz keš memorije.

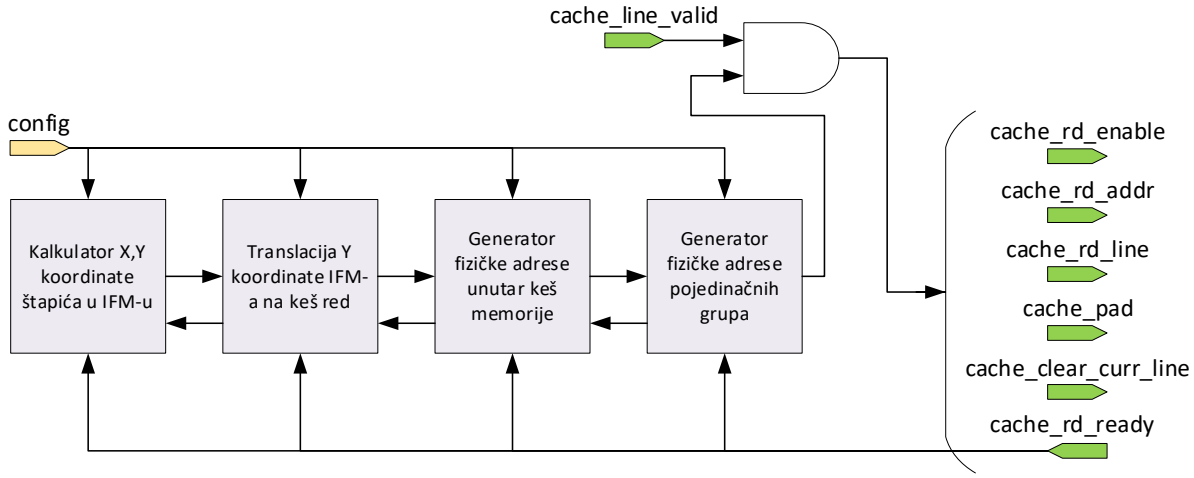
Da bi se ovakvo čitanje omogućilo potrebno je implementirati nešto kompleksniju logiku u odnosu na keš upisivač. Razlog za tako nešto leži u činjenici da se isti štapići čitaju više puta, pri čemu je broj ponovljenih isčitavanja različit za svaki štapić u zavisnosti od konfiguracije konvolucionog sloja. U slučaju na slici 33, štapić u gornjem levom uglu će biti pročitan samo

jednom, štapić u redu 0 i koloni 1 dva puta i tako redom. Kako bi se napravio univerzalni keš čitač sposoban da akcelerira nama poznate konvolucione slojeve potrebno je da se na ulaz čitača dovede sledeća konfiguracija.

Naziv konfiguracionog porta	Opis
start	Konfiguracija na ostalim portovima je validna. Čitanje može da počne.
stick_depth	Označava dubinu štapića, to jest, dubinu kernela koja se trenutno procesira. Pošto se nekad neće procesirati kompletna dubina IFM-a u jednom prolazu, Stick_depth nije isto što i Ifm_depth u opštem slučaju.
ifm_height	Visina IFM-a.
ifm_width	Širina IFM-a.
pad_top	Broj štapića koji označava *zero padding* na vrhu IFM-a.
pad_bot	Broj štapića koji označava *zero padding* na dnu IFM-a.
pad_left	Broj štapića koji označava *zero padding* na levoj strani IFM-a.
pad_right	Broj štapića koji označava *zero padding* na desnoj strani IFM-a.
kernel_h	Visina kernela.
kernel_w	Širina kernela.
stride_horizontal	Horizontalni korak konvolucije.
stride_vertical	Vertikalni korak konvolucije.
cache_height	Visina keš memorije (visina u broju štapića IFM-a).

Tabela 7 Konfiguracioni portovi keš čitača.

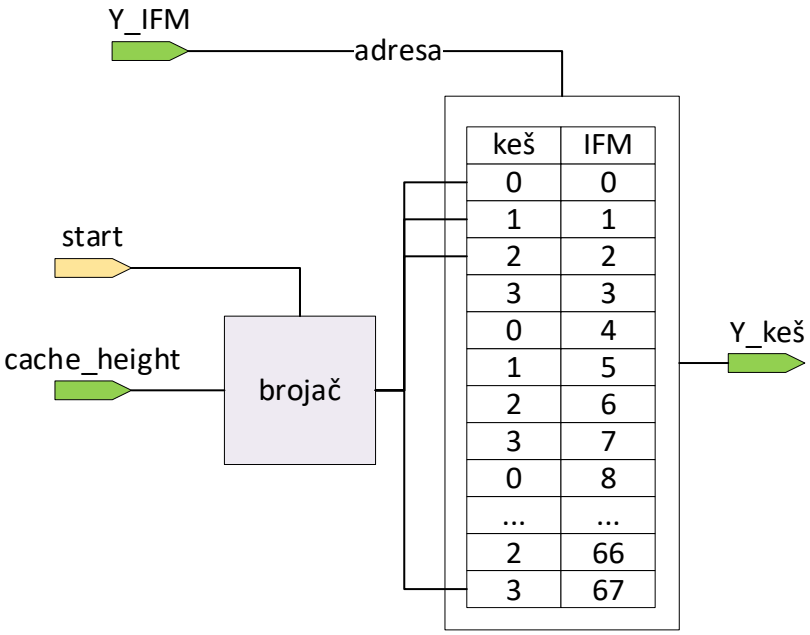
Primetimo da u tabeli 7 postoje i konfiguracioni podaci o *padding*-u što znači da keš čitač pored upravljanja memorijom ima ulogu i u kreiranju *padding*-a. U slučaju da je potrebno generisati štapić koji pripada *zero padding* delu IFM-a, keš čitač će ovo signalizirati keš memoriji koja će proslediti nule ka nizu PE-ova. Uprošćena blok šema keš čitača je prikazana na slici 34.



Slika 34 Uprošćeni blok dijagram keš čitača.

Prvi blok u nizu računa stvarne X, Y koordinate štapića u IFM-u prateći koordinatni sistem kao na slici 33. Ovo znači da je koordinata štapića u gornjem levom uglu (0, 0). Brojači unutar ovog modula generišu koordinate na osnovu konfiguracionih ulaza i predaju ih narednom bloku koji translira generisane Y koordinate IFM-a u Y koordinatu keš memorije.

Translacija Y koordinate IFM-a u Y koordinatu keš memorije je najzahtevnija jer suštinski predstavlja promenljivi modulo operator. Na primer, ako je visina keša kao na slici 33 jednaka četiri reda, to znači da će se redovi 0, 1, 2 i 3 IFM-a mapirati na redove 0, 1, 2 i 3 keš memorije. Pošto keš nema više redova to implicira da će se red 4 IFM-a mapirati na red 0 keša, red 5 IFM-a na red 1 keša i tako redom. Ova operacija je izuzetno jednostavna kako logički tako i za implementaciju, kada je visina keš memorije jednaka 2^n, međutim to nije uvek slučaj. Na primer, u slučaju da je KH = 3 i Stride = 2 keš memorija bi bila visine 5 redova. Za ovaj slučaj je potrebno implementirati modulo 5 logiku koja je daleko kompleksnija od modulo 2^n. Kako bi se kreirao univerzalan modulo operator potrebno je uzeti u obzir sve moguće kombinacije KH i Stride-a. Da bi se zadržala jednostavnost, modulo operator je realizovan pomoću brojača i male memorijske jedinice što je prikazano na slici 35.



Slika 35 Translacija Y koordinate IFM-a u Y koordinatu keš memorije.

Brojač se pokreće u trenutku kada start signal postane aktivan. Brojač broji kružno od 0 do *cache_height*, što je u primeru na slici 35 do četiri. Idući redom kroz adrese ove memorije brojač u svakom taktu upisuje narednu vrednost iz opsega 0-3 na sledeću adresu. Primetimo

da se IFM kolona ne čuva u memoriji već da predstavlja adresu lokacija u memoriji. Ova adresa odgovara IFM koordinati. Kolona nazvana keš, sadrži vrednosti svoje adrese po modulu *cache_height*. U toku rada keš čitača, prvi blok sa slike 34 dostavlja Y_IFM koordinatu koja inicira čitanje Y_keš koordinate iz memorije. Veličina ove memorije nije velika i realizovana je pomoću LUT-ova kao distribuirani RAM dostupan na Xilinx FPGA čipovima. Broj LUT-ova potreban za realizaciju zavisi od maksimalne visine IFM-a koja za veliku većinu poznatih CNN-ova nije veća od 300. Visina keša, koja definiše broj bita potrebnih za čuvanje modula, gotovo nikad nije dvocifrena. Imajući navedeno u vidu, sa velikom sigurnošću možemo reći da visina IFM-a u embeded aplikacijama neće preći 512, i da visina keša neće biti veća od 32. Ukoliko znamo da svaki LUT može da čuva 32 bita jasno je da za navedenu konfiguraciju potrebno 80 LUT-ova. Čak i da je potrebno smestiti dvostruko više vrednosti, na primer, zbog dvostruko veće visine IFM-a, udeo ovog bloka u ukupnoj potrošnji LUT-ova će ostati u granicama od oko 1% celog akceleratora. Naravno, u embeded aplikacijama se ne očekuju dimenzije IFM-a veće ni od 512. Na kraju, napomenimo i to da se osvežavanje sadržaja ove memorije vrši na početku svakog konvolucionog sloja, zato što dimenzije kernela i korak konvolucije u opštem slučaju nisu identični za sve slojeve neke mreže. Primetimo i to da ostatak sistema nema potrebu da čeka da se celokupna memorija osveži, već može istog trenutka početi sa procesiranje. Ovo je posledica toga što se prva adresa keš linije uvek poklapa sa rednim brojem prvog reda (adresa 0, red 0). Dalje, dok se ovaj red upiše, to jest, dok se ne dobije signal koji će omogućiti čitanje istog, uvek prođe dovoljno vremena da se cela tabela popuni novim vrednostima.

Sledeći u nizu blokova keš čitača je generator početne fizičke adrese štapića unutar keš memorije. Na početnu vrednost trenutnog štapića utiče širina dela IFM-a koji se obrađuje, dubina štapića kao i koordinate štapića unutar snopa koji se trenutno procesira. Obrazac po kome se računa bazna adresa štapića u keš memoriji je prikazana izrazom 5:

$$bazna\ adresa\ štapića = (Y_{keš} \bullet stripe_w + X) \bullet stick\_depth \quad (5)$$

U izrazu 5 *Y_keš* predstavlja red unutar keš memorije, *X* kolonu u kojoj se štapić nalazi dok *stripe_w* predstavlja širinu vertikalnog dela IFM-a koji se obrađuje.

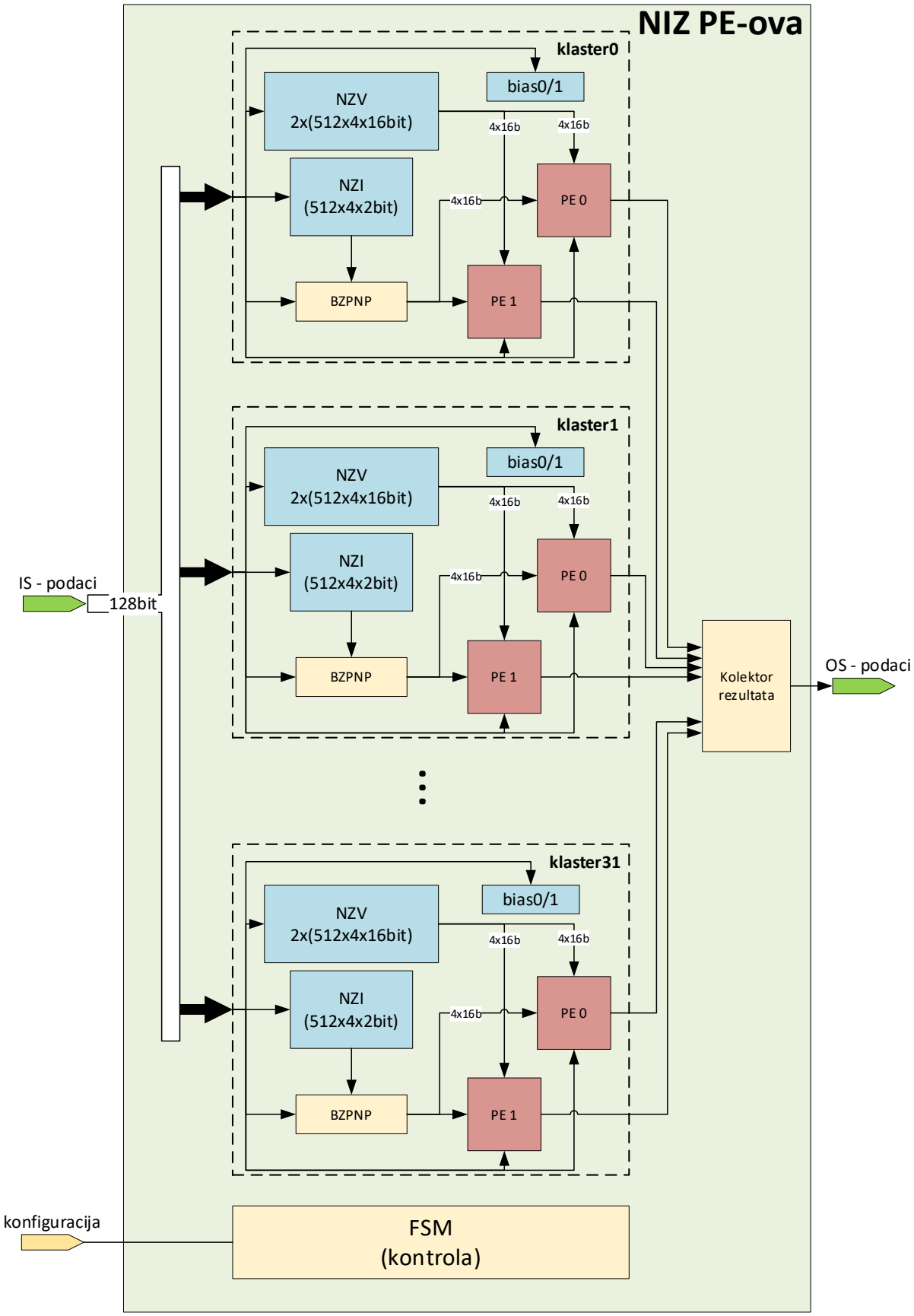
Poslednji u nizu blokova je generator fizičke adrese pojedinačnih grupa od po osam tačaka IFM-a. Ova adresa se inkrementira za jedan svaki takt i dodaje na baznu adresu trenutnog štapića. Po završetku isčitavanja svih podataka iz jednog štapića ovaj brojač se resetuje čime se menja stanje svih blokova koji prethode, to jest, prelazi se na sledeći štapić.

6.5 Niz procesorskih elemenata

Niz procesorskih elemenata (engl. *PE Array*), predstavlja centralnu procesorsku jedinicu Argus akceleratora. Niz PE-ova je visoko optimizovan da izvršava operaciju skalarnog proizvoda između kernela i odgovarajućih delova IFM-a bilo da je mreža orezana ili ne. Čine ga 32 PE-a grupisana u 16 klastera, 16 BZPNP pridruženih blokova, memorijski elementi za čuvanje *bias*/NZI/NZV-ova, blok za prikupljanje rezultata i prosleđivanje ka OS modulu i mašina stanja koja kontroliše tokove podataka unutar niza PE-ova. Blok dijagram niza PE-ova sa navedenim elementima je prikazan na slici 36. U nastavku su detaljno opisane mikroarhitekture svih blokova sa slike 36 kao i tok podataka prilikom procesiranja konvolucionih slojeva.

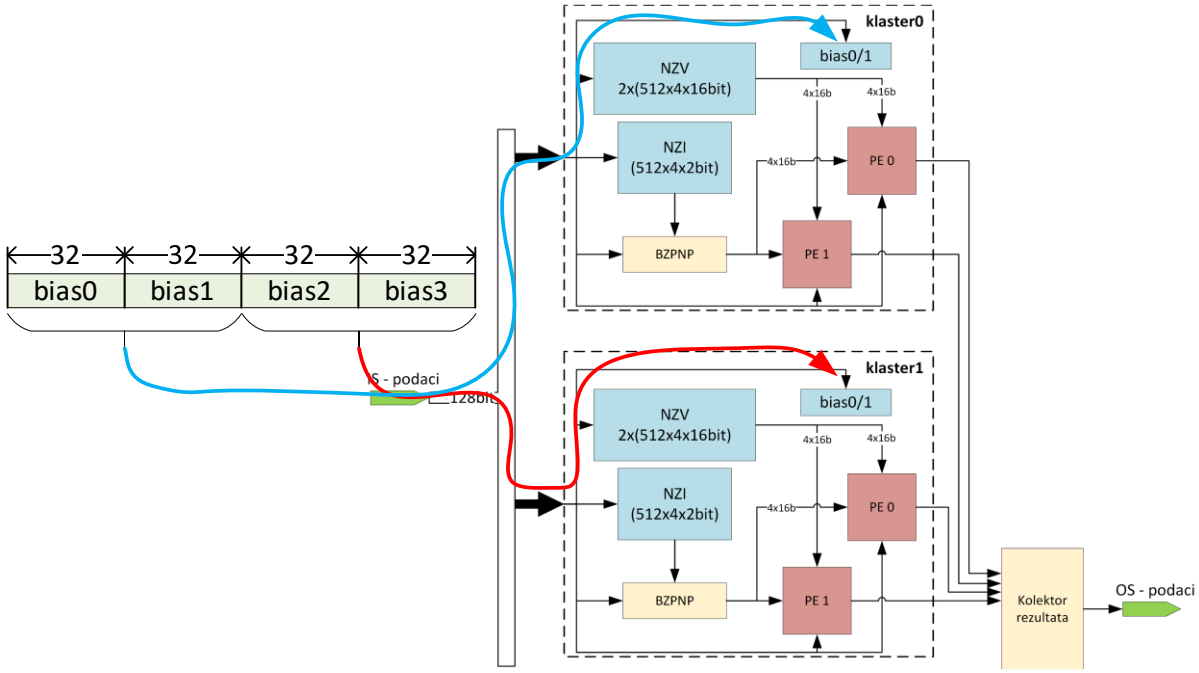
Bias memorijski element čine po dva 32-bitna registra u svakom klasteru. Svakom PE-u je dodeljen po jedan kernel što implicira da svakom PE-u pripada po jedan od dva navedena *bias* registra.

Pošto je za svaki klaster potrebno skladištiti samo dva *bias*-a, ukupno 64 bita, memorija je implementirana pomoću flip-flopova. Distribuirani RAM bi bio nepraktičan iz razloga što svaki LUT koji se koristi u ove svrhe može da sačuva 32 bita, a bio bi korišćen za čuvanje samo jednog bita. Distribuirani RAM je takav da se u svakom trenutku može pristupiti samo jednom bitu od 32 što znači da bi za svaki *bias* bilo potrebno 32 LUT-a, odnosno 1024 ukupno. Ovolika količina LUT-ova čini približno 10% kompletnog Argus akceleratora sa jednim CC modulom, što je neprihvatljivo. BRAM ima još veću granularnost tako da iz istih razloga nije prikladan za ovu namenu. Na kraju, treba napomenuti da se *bias*-i dopremaju preko zajedničke magistrale koja dolazi od IS modula. Ova magistrala je širine 128-bita te se *bias*-i dopremaju u grupama od po četiri kako bi se iskoristila maksimalna propusna moć magistrale. Naravno, za dopremanje *bias*-a, ni pojedinačno slanje ne bi predstavljalo veliku degradaciju performansi, ali je izvedeno na ovaj način da bi bilo u skladu sa smeštanjem NZV i NZI čiji transfer ima značajan udeo u vremenu obrade, posebno dubljih slojeva CNN-a. Slika 37 ilustruje smeštanje vrednosti *bias*-a u klastere 0 i 1 u prvom taktu.



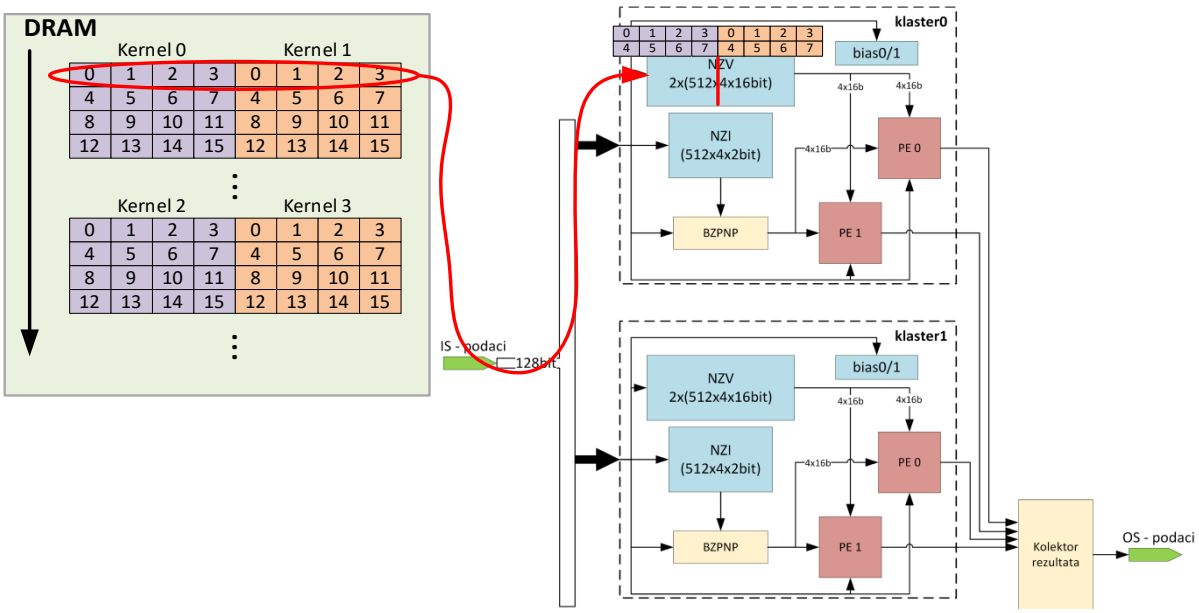
Slika 36 Arhitektura niza procesorskih elemenata.

NZV blok je namenjen za čuvanje parametara kernela bilo da je mreža orezana ili ne. Veličina NZV memorija je određena veličinom BRAM ćelija čija je maksimalna širina 64 bita. Navedena širina izuzetno pogoduje šablonu orezivanja koji od osam uzastopnih parametara orezuje četiri. To znači da se svaka orezana grupa može smestiti u jednu memorijsku lokaciju BRAM-a jer Argus akcelerator radi sa 16-bitnom aritmetikom. Pored toga što je zgodno smestiti sve parametre jedne grupe u jednu lokaciju, ovo je i potreban uslov za efikasno pristupanje memoriji jer BRAM u ovoj konfiguraciji ima samo jedan port za čitanje. Navedeno znači da će propusna moć BRAM-a biti uparena sa potrebama pridruženog PE-a, koji je u stanju da izvrši četiri MAC operacije u svakom taktu. U pogledu količine memorije, svaki NZV blok čine dva BRAM-a, svaki dimenzija 512x4x16bita.



Slika 37 Smeštanje bias-a unutar klastera.

Kao i u slučaju transporta *bias*-a i za NZV se koristi pun potencijal 128-bitne magistrale. Pošto svaki BRAM prihvata 64 bita u jednom taktu, jasno je da će u svakom taktu biti aktivna dva BRAM-a. Kako bi se lakše manipulisalo podacima, aktivna dva BRAM-a pripadaju istom klasteru. Da bi ovakav pristup bio moguć potrebno je učešljati parametre kernela iz istog klastera prilikom preprocesiranja i smeštanja u DRAM kao na slici 38.

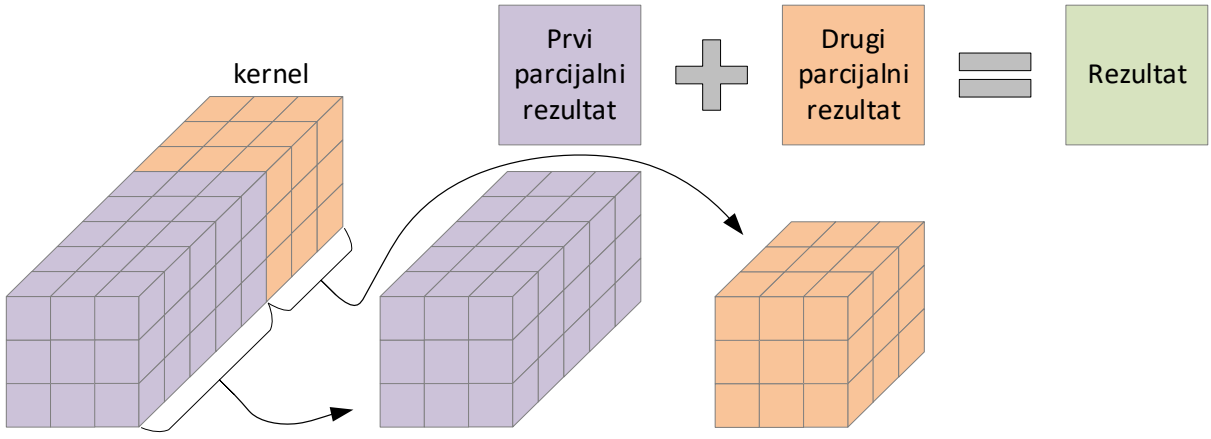


Slika 38 Učešljavanje parametara kernela unutar DRAM-a i njihov transport ka NZV blokovima.

Učešljavanje parametara dva kernela se vrši tako što se u DRAM naizmenično smeštaju blokovi od po četiri parametra, prvo jednog pa drugog kernela i tako redom. Ovakav pristup je potreban neovisno od toga da li je mreža orezana ili nije. Da bi se jasnije uvidele prednosti učešljavanja kernela uzmimo za analizu poslednji konvolucioni sloj VGG16 CNN-a. Ovaj sloj čine 512 kernela dimenzija 3x3x512. Ukupno, sloj ima 2.359.296 parametara u slučaju neorezane, odnosno, 1.179.648 u slučaju da je mreža orezana. Za dopremanje ovih, 16-bitnih parametara, u idealnom slučaju je potrebno najmanje 147.456 taktova (8 parametara u svakom taktu orezane mreže). Ako bi se izbeglo učešljavanje, bilo bi potrebno dvostruko više taktova (294.912) jer BRAM ne može da prihvati više od 64-bita podatka u jednom taktu. Sa druge strane, za procesiranje sloja u idealnom slučaju (100% efikasnost PE-ova) je potrebno 903.168 taktova ako je mreža orezana. Do ovog broja se može doći ako uzmemo da je broj tačaka OFM-a jednak 14x14x512, to jest, 100.352. Svaka tačka se može izračunati za 288 taktova što ukupno dovodi do 28.901.376 taktova ako jedan PE procesira kompletan sloj. Pošto Argus akcelerator sa jednim CC blokom ima 32 PE-a, ovaj broj treba podeliti sa 32 u idealnom slučaju. Dakle, računanje OFM-a traje 903.168 taktova, a dopremanje 147.456 u slučaju da su kerneli učešljani i 294.912 kada ne bi bili. Prevedeno u udeo u vremenu, učešljavanje skraćuje vreme procesiranja konkretnog sloja za više od 12%.

Imajući u vidu da je količina memorije za smeštanje NZV-ova ograničena jasno je da Argus arhitektura neće biti u stanju da obradi sve poznate veličine kernela u jednom prolazu čak i

kada su mreže orezane. Za primer, uzmimo takođe, poslednji konvolucioni sloj VGG16 CNN-a. Ukupno, svaki kernel ima 4608 parametara (3x3x512) ako je mreža neorezana, odnosno, 2304 u slučaju orezane mreže. Veličina BRAM-a je takva da može da se smesti samo 2048 parametara (4x512) što nije dovoljno ni za orezanu mreža sa 2304 parametra po kernelu. Kako bi arhitektura bila upotrebljiva i efikasna i u ovakvim primerima, potrebno je kernele procesirati parcijalno, što ćemo u nastavku nazivati parcijalna konvolucija. Slika 39 ilustruje ideju procesiranje slojeva kod kojih je broj parametara kernela veći od dimenzija BRAM memorije, na visokom nivou. U ovakvim slučajevima, kernel se deli na dva ili više manjih delova po dubini kernela. U primeru na slici 39, kernel je podeljen na ljubičasti deo, koji se prvi procesira, i narandžasti deo čije procesiranje sledi po završetku ljubičastog dela. U prvom prolazu se učitava ljubičasti deo kernela, a potom se kreće sa procesiranjem IFM-a, takođe, dubine jednake dubini ljubičastog dela kernela. Ovo je jedini slučaj kada dubina IFM-a nije jednaka dubini štapića koji se procesira. Rezultati obrade IFM-a ljubičastim delom kernela se smeštaju u DRAM. Kada su izračunati svi rezultati prvog dela parcijalne konvolucije, počinje učitavanje narandžastog dela kernela i računanje odziva za ovaj deo kernela. Pošto je konvolucija linearna operacija, prvi deo parcijalne konvolucije se može sabrati na odgovarajuće rezultate drugog dela parcijalne konvolucije. Ovo sabiranje se vrši paralelno sa računanjem odziva na narandžasti deo kernela što operaciju sabiranja maskira u vremenu. Sabrane vrednosti prvog i drugog dela parcijalne konvolucije daju konačan rezultat koji se smešta u DRAM. Detaljniji opis sabiranja i celokupnog postupka računanja konvolucije je prikazan u nastavku, tačnije, u poglavlju koje opisuje OS, koji potražuje od DRAM-a prethodne rezultate i sabira ih na tekuće vrednosti koje dostavlja niz PE-ova.

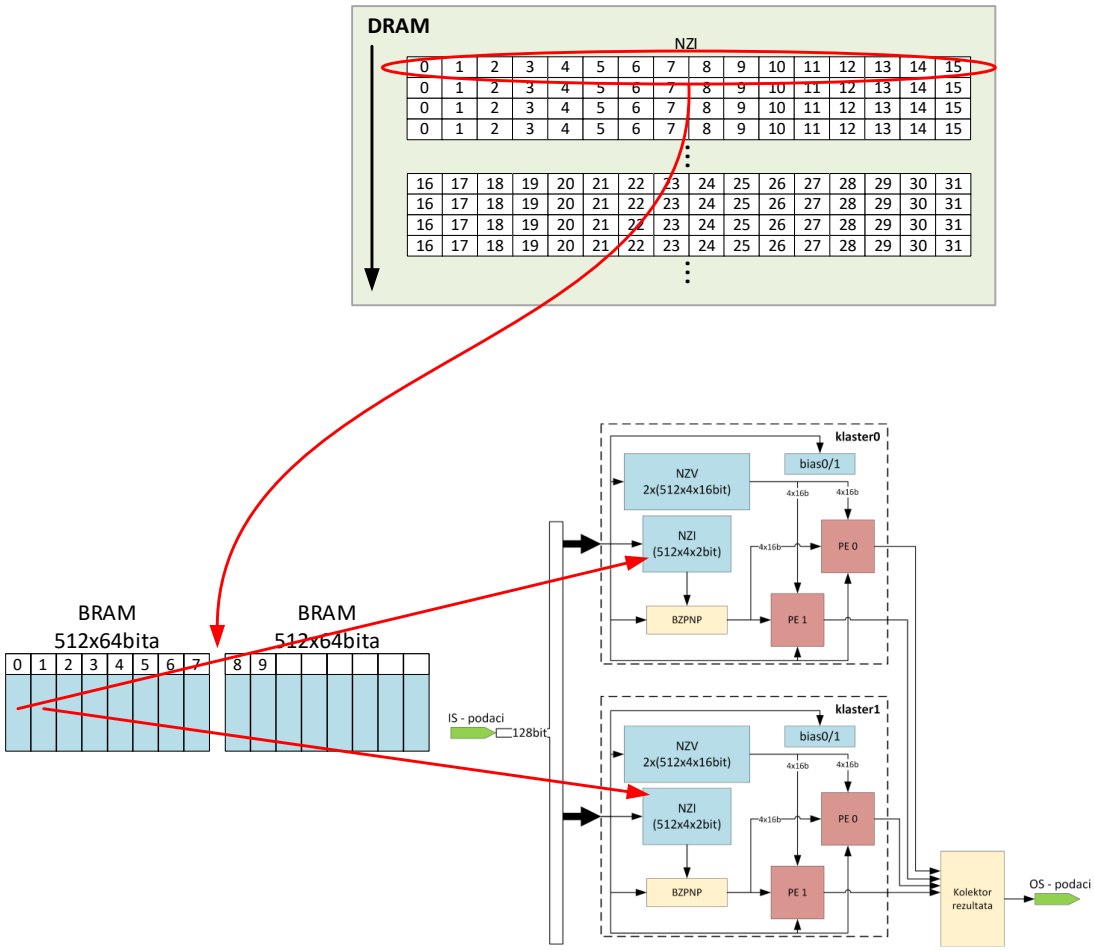


Slika 39 Računanje rezultata konvolucije iz dva prolaza, parcijalna konvolucioja.

Kerneli sa većim brojem parametara od dostupnih lokacija u BRAM-u nisu retkost u modernim CNN-ovima, posebno u slučaju kada se procesiraju neorezane mreže. U slučaju neorezanog VGG-16 poslednjih osam od trinaest slojeva se ne mogu procesirati bez mehanizma parcijalne konvolucije. Navedeno povlači za sobom da će se sve potencijalne manjkavosti ovakvog pristupa značajno odraziti na performanse celokupne arhitekture. Baš zbog toga je odabrano da se kerneli dele po dubini. Ovom podelom dobijamo ništa drugo nego više manjih slojeva standardnih oblika. Naravno, ovo nisu stvarni dodatni slojevi, niti je arhitektura CNN-a promenjana. Jedina razlika se uočava iz perspektive akceleratora koji vidi CNN sa više slojeva. Zbog toga, ovakav način deljenja kernela ne unosi dodatne potrebe u smislu hardvera te kompleksnost ostaje na prvobitnom nivou. Svakako, potrebno je drugačije smestiti parametre u DRAM-u prilikom prevođenja CNN modela iz Python-a u C što ne predstavlja komplikovan proces.

NZI blok sa slike 36 ima ulogu čuvanja NZI vrednosti za pripadajući klaster. Kao u slučaju NZV parametara, i NZI vrednosti se čuvaju u BRAM-ovima. Veličina NZI memorije mora da bude takva da može da čuva sve NZI vrednosti za 512 grupa od po četiri NZV parametra. Drugim rečima potrebno je da svaka grupa unutar NZV-a ima pripadajući NZI. Širinu svakog NZI-a za jednu grupu određuje broj parametara po grupi (četiri) i širina svakog indeksa. Pošto preostali parametri mogu biti selektovani multiplekserima 4-na-1 može se zaključiti da je za svaki indeks potrebno po dva bita, ukupno 8 bita po NZI grupi.

Kako bi se uparile propusne moći svih memorijskih blokova sa PE-om, koji može da procesira jednu grupu NZV-ova u jednom taktu, potrebno je da u svakom taktu NZI memorija može da dostavi 4 indeksa, to jest, NZI za jednu NZV grupu. Kako bi se NZI memorija efikasno mapirala na raspoložive BRAM-ove, a zadržala osobina da u svakom taktu može da se isporuči NZI za jednu grupu, osam klastera dele jedan BRAM. Već je rečeno da je širina jedne lokacije u BRAM-u maksimalno 64 bita, a da je dubina u ovom slučaju 512 lokacija. Pošto broj lokacija odgovara maksimalnom broju grupa NZV-ova potrebno je još samo iskoristiti pun potencijal širine svake lokacije. Da bi se maksimalno iskoristili memorijski potencijali, moguće je deliti jedan BRAM između osam klastera. Ovo znači da se u svakoj 64-bitnoj lokaciji BRAM-a nalaze NZI-evi za osam grupe NZV-ova koji pripadaju različitim klasterima. Prikaz deljenja BRAM-a između klastera i učitavanja indeksa iz DRAM-a je ilustrovan slikom 40.

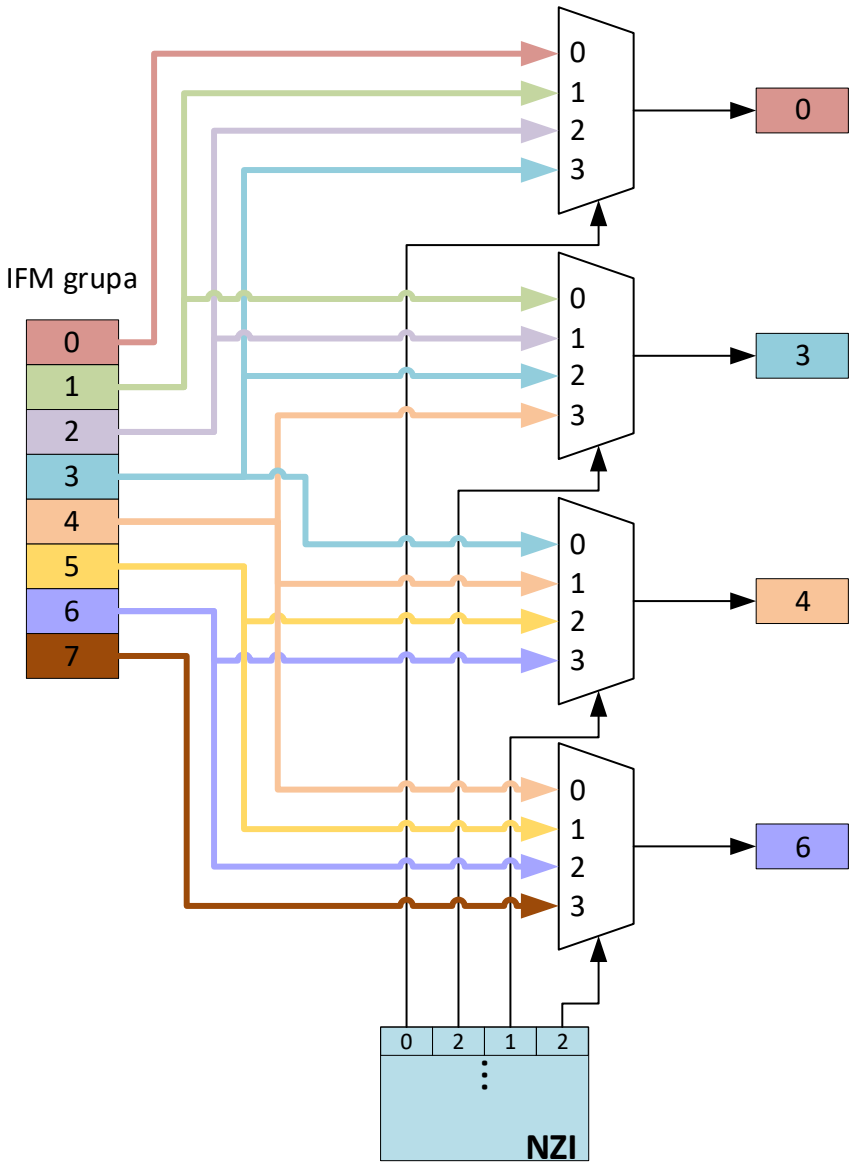


Slika 40 Deljenje BRAM ćelija između klastera i učitavanje NZI vrednosti iz DRAM-a u BRAM-ove.

Uočimo da je učešljavanje zarad iskorišćenja punog potencijala magistrale podataka prisutno i u slučaju NZI vrednosti. Takođe, sa slike 40 je jasno da se simultano učitavaju NZI-evi za 16 klastera, što znači 32 kernela. Da bi navedeno bilo moguće, potrebno je na pravilan način pripremiti NZI nizove u DRAM-u.

BZPNP prihvata NZI vrednosti za tekuću IFM grupu nad kojom vrši odabir četiri potrebne tačke IFM-a. Čine ga isključivo četiri 16-bitna multipleksera 4-na-1 kao na slici 41. Blok je u potpunosti realizovan kao kombinaciona mreža.

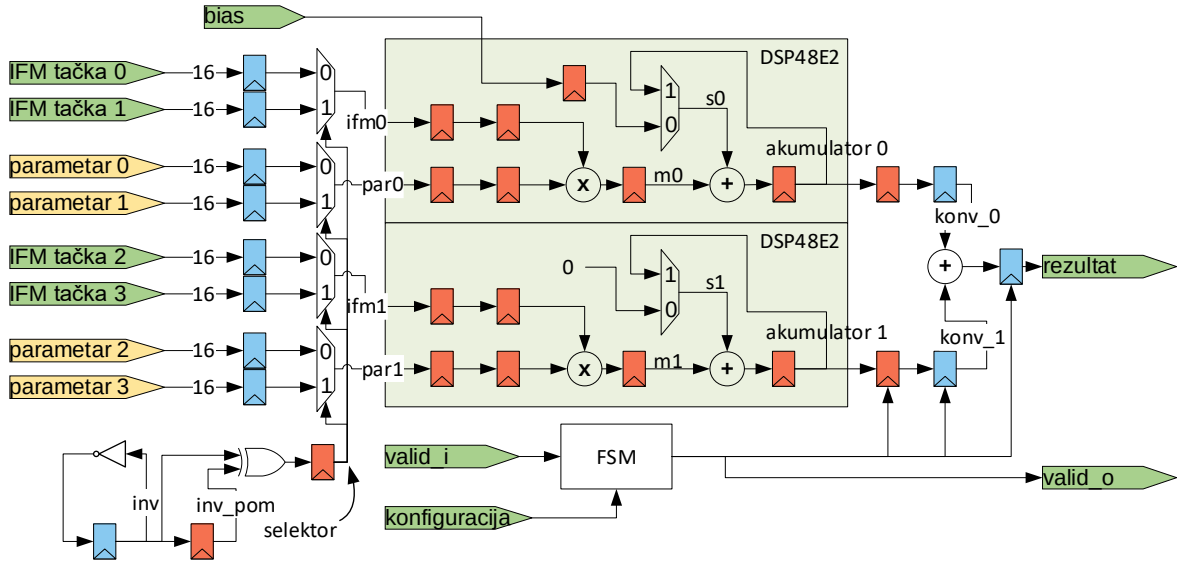
Primetimo da je prilikom formiranja NZI nizova unutar DRAM-a potrebno prevesti apsolutne pozicije koje svaka tačka u IFM grupi ima u relativne u odnosu na to koji deo IFM grupe se dovodi do multipleksera. Za NZI kombinaciju sa slike 41 (0, 2, 1, 2) BZPNP selektuje tačke IFM-a na pozicijama 0, 3, 4, 6.



Slika 41 Primer rada BZPNP bloka.

Procesorski elemenat predstavlja centralni blok akceleratora u kome se vrši izračunavanje konvolucija. Zbog prirode konvolucije, PE-ovi su visoko optimizovani za izvršavanje MAC operacija. Detaljna arhitektura PE-a je prikazana na slici 42. Na prvi pogled, može se uočiti da jezgro svakog PE-a čine dva DSP bloka. Konkretno, korišćeni su dostupni DSP blokovi na Xilinx MPSoC FPGA platformama, oznake DSP48E2 [51]. Tok podataka je takav da PE u svakom taktu prihvata četiri tačke IFM-a i odgovarajuće parametre. Ove četiri tačke IFM-a dostavlja BZPNP, dok se parametri učitavaju iz NZV memorije pridružene klasteru. Svaki od DSP-eva računa jednu polovinu konvolucije. Gornji DSP uzima IFM tačke 0/1 i parametre koji idu uz njih da bi izvršio dve MAC operacije i rezultat smestio u akumulator. Identičane korake izvršava i donji DSP sa razlikom što procesira IFM tačke 2/3 i odgovarajuće parametre. Ceo postupak se

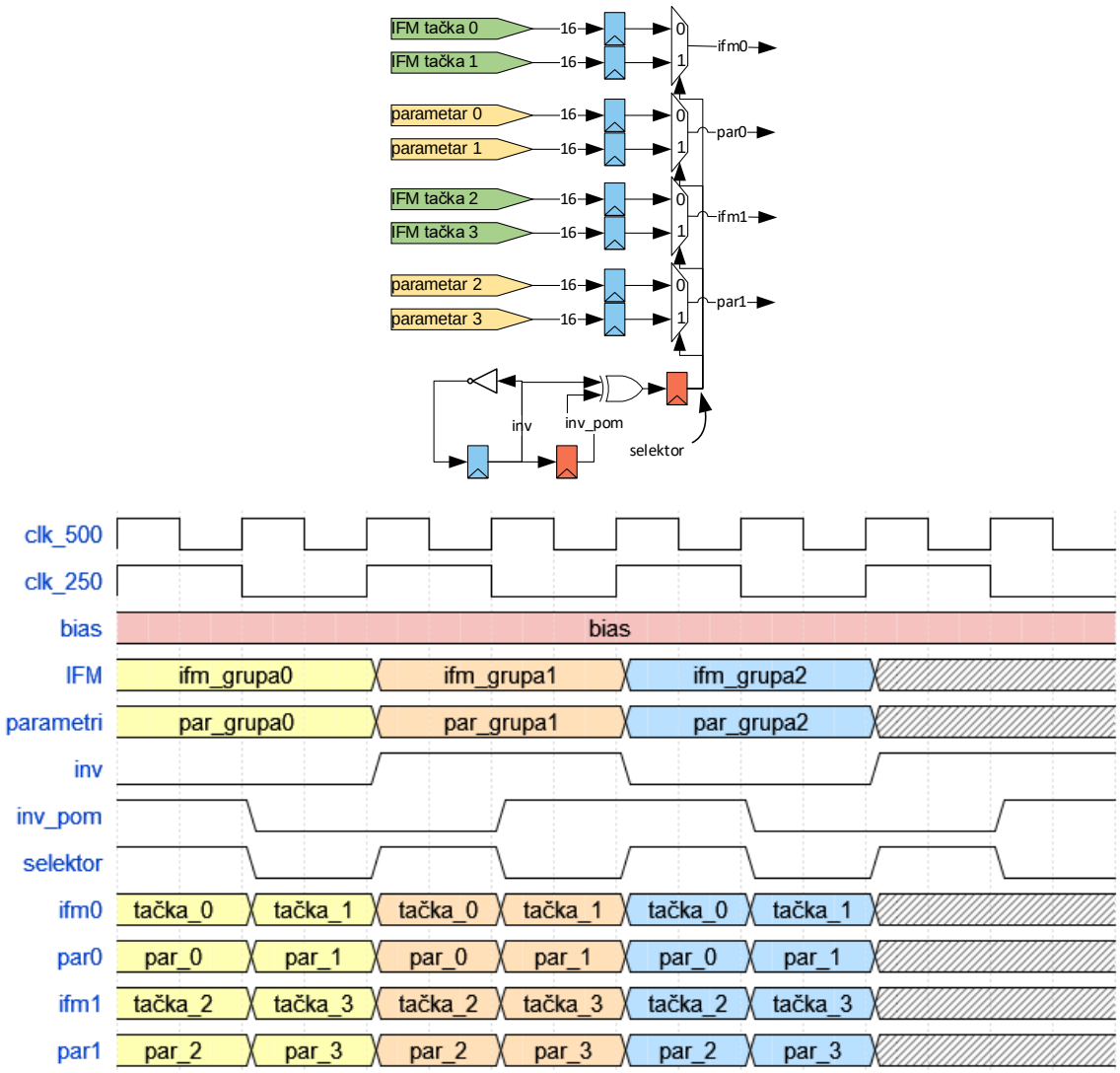
ponavlja dokle god se ne procesiraju sve ulazne tačke jednog snopa štapića IFM-a. Po završetku, rezultati akumulatora 0 i 1 se sabiraju pomoću izlaznog sabirača da bi se rezultat potom prosledio bloku koji prikuplja rezultate konvolucije.



Slika 42 Detaljan arhitektura jednog procesorskog elementa.

Pošto svaki DSP blok ima samo jedan fizički množač, a radi sa 16-bitnim ulaznim podacima, jasno je da u svakom taktu može da izvrši samo jednu MAC operaciju. Sa druge strane, u algoritmu 4 je navedeno da svaki PE ima kapacitet da izvrši četiri MAC operacije u jednom taktu, dve po DSP bloku. Kako bi se ostvarile navedene performanse sa dva DSP bloka, moguće je udvostručiti frekvenciju DSP bloka i tako povećati računske kapacitete svakog DSP-a. Ova tehnika se u literaturi može pronaći pod nekoliko naziva od kojih je *Multi-Pumping* [52], [53] najrasprostranjeniji. Osnovna ideja je da se podaci dopremaju dvostruko širom magistralom, što jeste slučaj na slici 42(četiri podatka na dva DSP bloka), a da DSP blok radi na tačno dvostruko većoj frekvenciji od okružujuće logike. Na ovaj način će DSP moći u svakoj periodi sporog takta da obradi po dve MAC operacije.

Na slici 42, registri su označeni različitim bojama u zavisnosti od frekvencije na kojoj rade. Plavi rade na frekvenciji od 250 MHz, što je frekvencija Argus akceleratora, dok su narandžastom bojom označeni registri koji rade na 500 MHz. Zbog rada na visokoj učestanosti za FPGA kola, izuzetnu pažnju je potrebno posvetiti prevođenju podataka iz sporog u brzi domen. Talasni oblik ulaznih signala i navedene logike je predstavljen slikom 43.

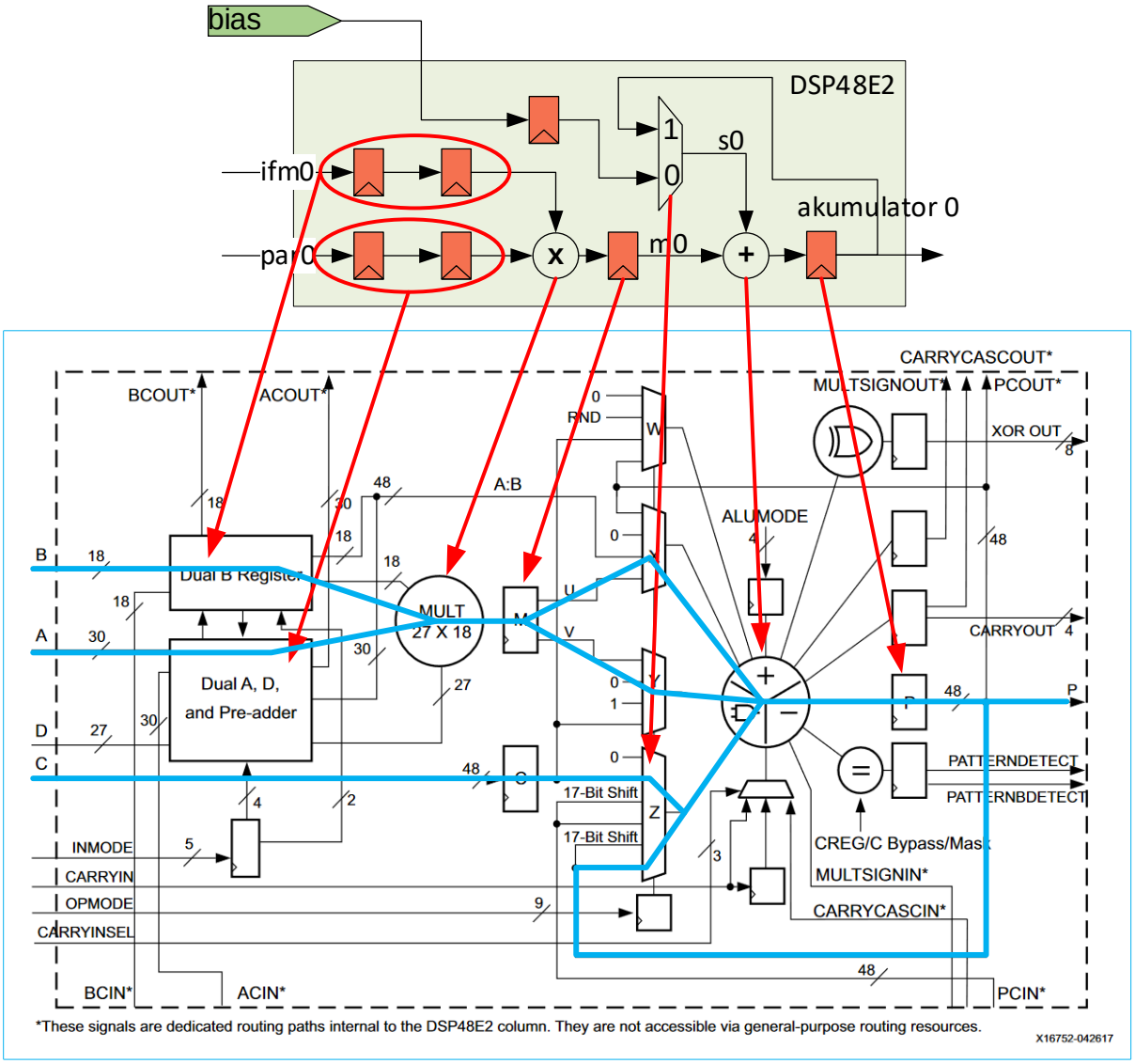


Slika 43 Talsni oblik signala na ulazu PE-a.

U sporom domenu se parametri smeštaju u registre ispred multipleksera koji prevode podatke iz jednog u drugi domen. Pošto u svakom sporom taktu pristiže po četiri IFM tačke i četiri parametra, multiplekserima je potrebno selektovati odgovarajuće podatke u svakom taktu brzog domena. Najkompleksniji deo ove logike predstavlja selekcioni signal koji je takav da u svakom taktu brzog domena menja svoj polaritet i tako selektuje odgovarajuće ulazne podatke. Signal nazvan *selektor*, formiraju tri flip-flopa i dve logičke kapije. Jedan flip-flop radi u sporom domenu i zadužen je za generisanje jednobitnog signala koji menja polaritet u svakom sporom taktu. Na slici 43 je ovaj signal nazvan *inv*. Da bi se u brzom domenu formirao ovakav signal potrebno je *inv* pomeriti za jedan brzi takt čime se formira *inv_pom*. XOR operator između ovih signala formira ulazni signal u izlazni flip-flop čiji izlaz predstavlja *selektor*. Primetimo da ne postoji nikakva sinhronizacija signala između dva klok domena.

Ovakav pristup je moguć samo u slučaju kada su takt signali fazno poravnati. Pored fazno poravnatih takt signala, izuzetno je važno da između registara u različitim domenima nema više od jedne logičke kapije, to jest, logike tolike da može da se mapira na maksimalno 1 LUT. Ovo je bitno kako bi se ulazni signali u *selektor* flip-flop stabilizovali već na pola periode sporog takta jer je to trenutak kada dolazi rastuća ivica brzog takta i ovaj signal biva semplovan. Takođe, selektor flip-flop nije u okviru *hard ip*-eva dostupnih na FPGA kolima već se sve implementira u programabilnoj logici. Ovo predstavlja dodatno opterećenje za kompajler posebno ako imamo u vidu da sve frekvencije preko 300 MHz u programabilnoj logici su teško dostižne bez izuzetne optimizacije od strane inženjera. Navedeno pravilo od maksimalno jednog LUT-a između flip-flopova na ovim učestanostima mora biti ispoštovano i između plavih registara podataka i prvih narandžastih registara unutar DSP blokova. Pošto se koriste 2-na-1 multiplekseri, moguće je svaki bit multipleksera mapirati na samo jedan LUT. Pored *selektor* signala, na slici 43 su prikazani izlazni podaci iz pomenutih multipleksera za tri grupe podataka koje predstavljaju hipotetički kernel dimenzija 3x1x8. Još jedan bitan detalj na koji treba obratiti pažnju je zadovoljavanje vremena uspostavljanja i držanja (engl. *setup* i *hold time*) za prve narandžaste registre unutar DSP blokova. *Setup time* će biti zadovoljen ako logika između flip-flopova u dva domena ne bude veća od jednog LUT-a što je ispoštovano. *Hold time* je zadovoljen tehničkim karakteristikama flip-flopova na Xilinx FPGA kolima pošto je *Tcq* (engl. *clock to Q*) *selektor* flip-flopa veće od *hold time*-a svih flip-flopova na FPGA kolu.

Na slici 44 je prikazano detaljno mapiranje jedne MAC jedinice PE-a na postojeći DSP48E2 *hard ip*. Plavim linijama su označene postojeće putanje DSP bloka koje se koriste u slučaju Argus PE-a. Ulazni podaci su dva puta zaregistrovani unutar *Dual B Register*, odnosno *Dual A, D, and Pre-adder* blokova. Iako logički nisu potrebni, njihovo korišćenje je obavezno u slučaju da se DSP koristi za rad na visokim učestanostima. Iz istog razloga je obavezno i korišćenje registra posle množača i posle sabirača. Izlazni registar predstavlja *akumulator*. U zavisnosti od toga da li je u toku obrada prve IFM grupe tekućeg skalarnog proizvoda, multiplekser Z će selektovati ulaz C ili registar P (akumulator) da odabere drugi sabirak na sabirač. Pošto su ulazi 16-bitni, izlaz množača, a samim tim i akumulator su 32-bitni, što znači da nije iskorišćen kompletan potencijal akumulatora.

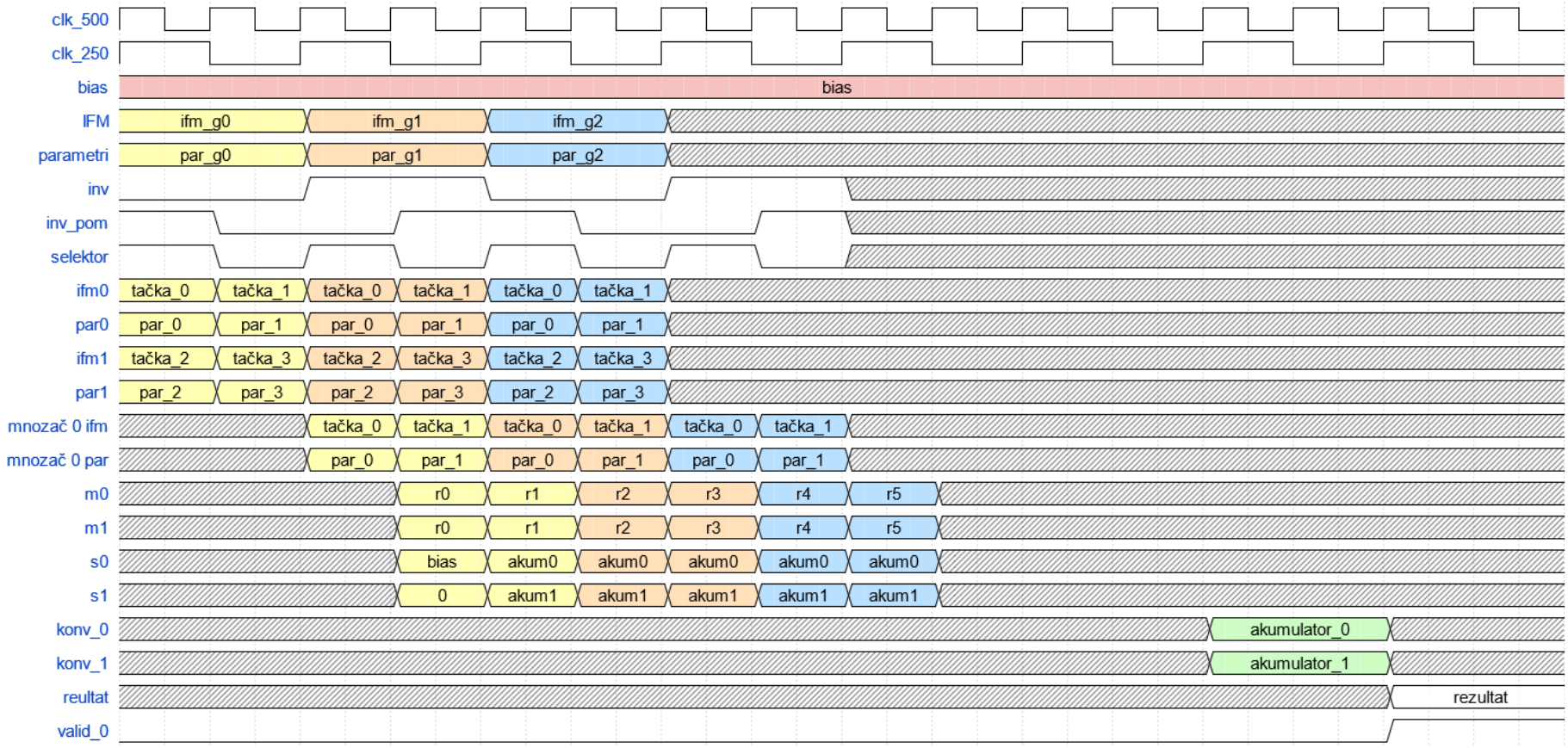


Slika 44 Mapiranje MAC jedinica na postojeći DSP blok, DSP48E2.

Iako je 500 MHz izuzetno visoka učestanost, čak i nedostižna za programabilnu logiku, za raspoložive DSP blokove to nije slučaj. Xilinx proizvodi FPGA kola podeljena u tri brzinska razreda, počevši od -1 do -3. Oznaku -1 nose najsporija dok -3 pripada najbržim FPGA kolima. Čak i za najsporija FPGA kola frekvencija od 500 MHz za DSP blokove je daleko od maksimalne koja iznosi oko 645 MHz. Za klasu kola označenu sa -2 ova maksimalna frekvencija se kreće i do 775 MHz, dok je za -3 maksimalna prijavljena učestanost DSP blokova 891 MHz [54]. Naravno, za postizanje navedenih frekvencija rada je obavezno iskoristiti sve prikazane registre na slici 44. Korišćenjem svih registara se maksimalno pajplajnuje (engl. *pipeline*) DSP. Punjenje *pipeline*-a zajedno sa talasnim oblicima svih relevantnih signala PE-a prilikom

izračunavanja jedne konvolucije je prikazana na slici 45. Primetimo da je i ovde korišćen primer sa hipotetičkim kernelom dimenzija 3x1x8.

Do sada opisanim delom PE-a, zaduženim za procesiranje podataka, upravlja po jedan FSM pridružen svakom PE-u. FSM nadgleda ulazni tok podataka i tako vodi evidenciju kolika je popunjenost *pipeline*-a. Pored toga, prati i frekvenciju preuzimanja rezultata od strane narednih blokova u akceleratoru. Ove dve informacije koristi kako bi generisao potrebne signale ka centralnom FSM-u koji će dalje da koordiniše rad IS modula. Ukoliko je potrebno, IS će biti pauziran dokle god se rezultati ne preuzmu od PE-a i oslobodi *pipeline* za nova računanja. Pored kontole toka podataka, FSM kontroliše i signale dozvole (engl. *enable, en*) registara pre i posle izlaznog sabirača. Takođe, vrši upravljanje multiplekserima unutar DSP blokova koji selektuju drugi sabirak na sabiraču. Za donji DSP blok na slici 42, ovaj multiplekser propušta nulu na početku svakog skalarnog proizvoda, što suštinski predstavlja brisanje prethodne vrednosti akumulatora. U slučaju gornjeg DSP-a, multiplekser propušta 32-bitnu vrednost *bias*-a čime se maskira vreme potrebno za ovu operaciju. Napomenimo i to da je *bias* poravnat sa formatom broja sa fiksnom tačkom koji se koristi unutar akumulatora, stoga je i vrednost 32-bitna. Naravno, vrednosti treba prilagoditi prilikom pripremanja CNN modela za DRAM, softverski. Ovakvim pristupom su izbegnuti pomerači sa promenljivim parametrima koji su vrlo zahtevni za implementaciju kada su u pitanju potrebni resursi. Mana je dodatno korišćenje raspoloživih flip-flopova, ali je to manje značajno jer je raspoloživost ovih resursa daleko povoljnija od LUT-ova, koji bi se koristili za navedene pomerače.



Slika 45 Talasni oblici svih relevantnih signala PE-a prilikom izračunavanja konvolucije za kernel dimenzija 3x1x8.

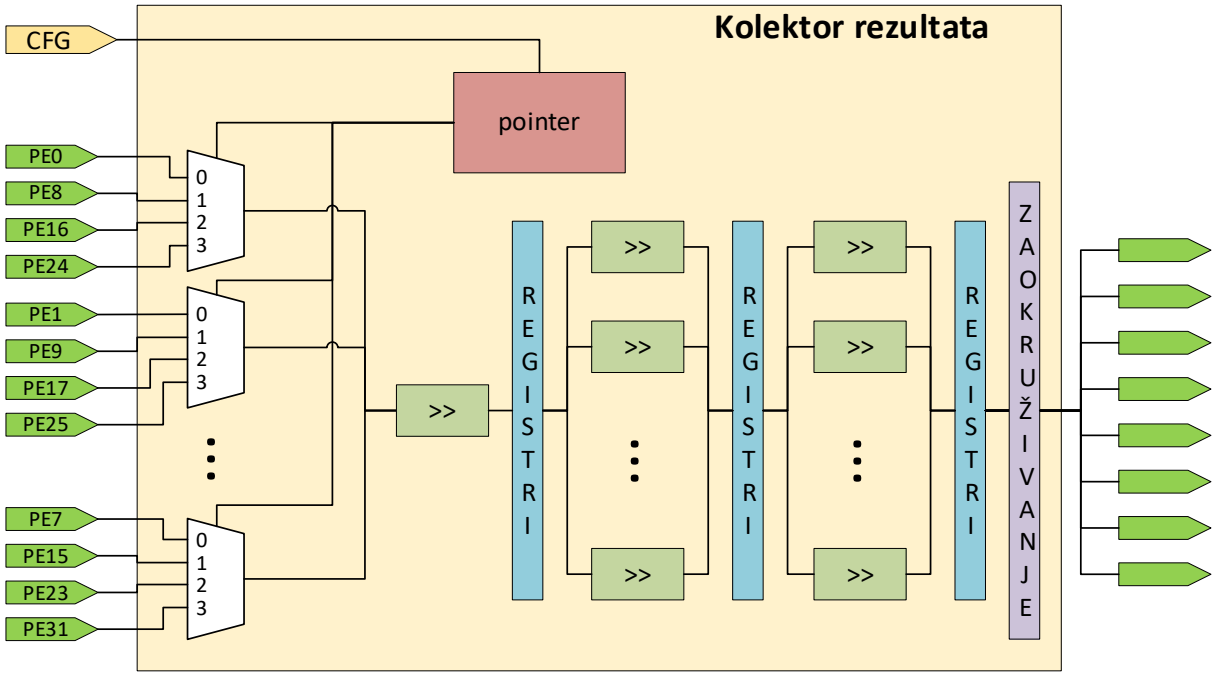
Iako se koristi nešto kompleksnija konfiguracija DSP48E2 bloka od standarden MAC operacije, postojeći kompajleri su dovoljno napredovali te nije potrebno instancionirati *hard ip* ove komponente i ručno podesiti parametre i povezati ulaze, već je dovoljno napisati kod na RTL nivou kao u listingu 1. Primetimo da imena registara odgovaraju nazivima prisutnim na slici 44 na kojoj je prikazano mapiranje na Xilinx DSP komponentu.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity macc_dsp is
  generic (A_W : natural := 16;
           B_W : natural := 16);
  Port(clk_i              : in std_logic;
       flush_n_i          : in std_logic;
       sreset_i           : in std_logic;
       a_i, bi, bias_i    : in signed(A_W-1 downto 0);
       p_o                : out signed(A_W+B_W-1 downto 0));
end macc_dsp;

architecture Behavioral of macc_dsp is
  signal a_r0, a_r1 : signed(A_W-1 downto 0);
  signal b_r0, b_r1 : signed(B_W-1 downto 0);
  signal m, p, c_reg: signed(A_W+B_W-1 downto 0);
begin
  process(clk_i)begin
    if clk_i'event and clk_i = '1' then
      if sreset_i = '1' then
        a_r0 <= (others=>'0');
        a_r1 <= (others=>'0');
        b_r0 <= (others=>'0');
        b_r1 <= (others=>'0');
      else
        a r0 <= a i;
        a_r1 <= a_r0;
        b_r0 <= b_i;
        b_r1 <= b_r0;
      end if;
    end if;
  end process;

  process(clk_i)begin
    if clk_i'event and clk_i = '1' then
      if sreset_i = '1' then
        c reg <= (others=>'0');
      else
        c_reg <= offset_i;
      end if;
    end if;
  end process;

  process(clk_i)begin
    if clk_i'event and clk_i = '1' then
      if sreset_i = '1' then
        m <= (others=>'0');
        p <= (others=>'0');
      else
        m <= b_r1 * a_r1;
        if flush_n_i = '0' then
          p <= m + c_reg;
        else
          p <= m + p;
        end if;
      end if;
    end if;
  end process;
  p_o <= p;
end Behavioral;
```

Listing 1 Mapiranje MAC jedinice na DSP48E2.

Kolektor rezultata je izlazni blok niza procesorskih elemenata, zadužen za prikupljanje rezultata od PE-ova. Uz prikupljanje, kolektor vrši i zaokruživanje vrednosti rezultata na 16 bita. Ovo je neophodno zato što su rezultati koje isporučuju PE-ovi 32-bitni. Naravno, ovakva redukcija preciznosti je moguća samo zato što je 16-bitna preciznost dovoljna za primenu prilikom procesiranja CNN-ova. Redosled operacija je takav da se prvo prikupljaju podaci od PE-ov, a zatim vrši zaokruživanje.

Prikupljanje se odvija u grupama od po osam rezultata u *Round-Robin* maniru. Širina magistrale od osam podataka je odabrana kako bi se maksimizovalo iskorišćenje DRAM kontrolera koji je u stanju da prihvati do 128-bita u svakom taktu (8 podataka širine 16 bita). Logiku za prikupljanje podataka čine osam multipleksera 4-na-1, pokazivač (engl. *pointer*) na trenutnu grupu od 32 PE-a i logika koja prati tokove podataka kako bi signalizirala okružujućim komponentama u kom se trenutno stanju nalazi kolektor. *Pointer* prihvata konfiguraciju u kojoj je najbitniji podatak o broju aktivnih PE-ova. Na primer, u slučaju da sloj ima samo 24 kernela, biće aktivni samo prvih šest od osam multipleksera. To znači da je u RR maniru potrebno prolaziti samo kroz grupe 0, 1 i 2, to jest, prikupljati podatke od PE-ova 0-7, 8-15 i 16-23. *Pointer* radi tako što se inkrementuje svaki put kada je kompletna grupa PE-ova na koju trenutno pokazuje spremna da dostavi rezultate. Kolektor u tom slučaju preuzima podatke i povećava vrednost za jedan, kako bi prešao na sledeću grupu i tako redom.

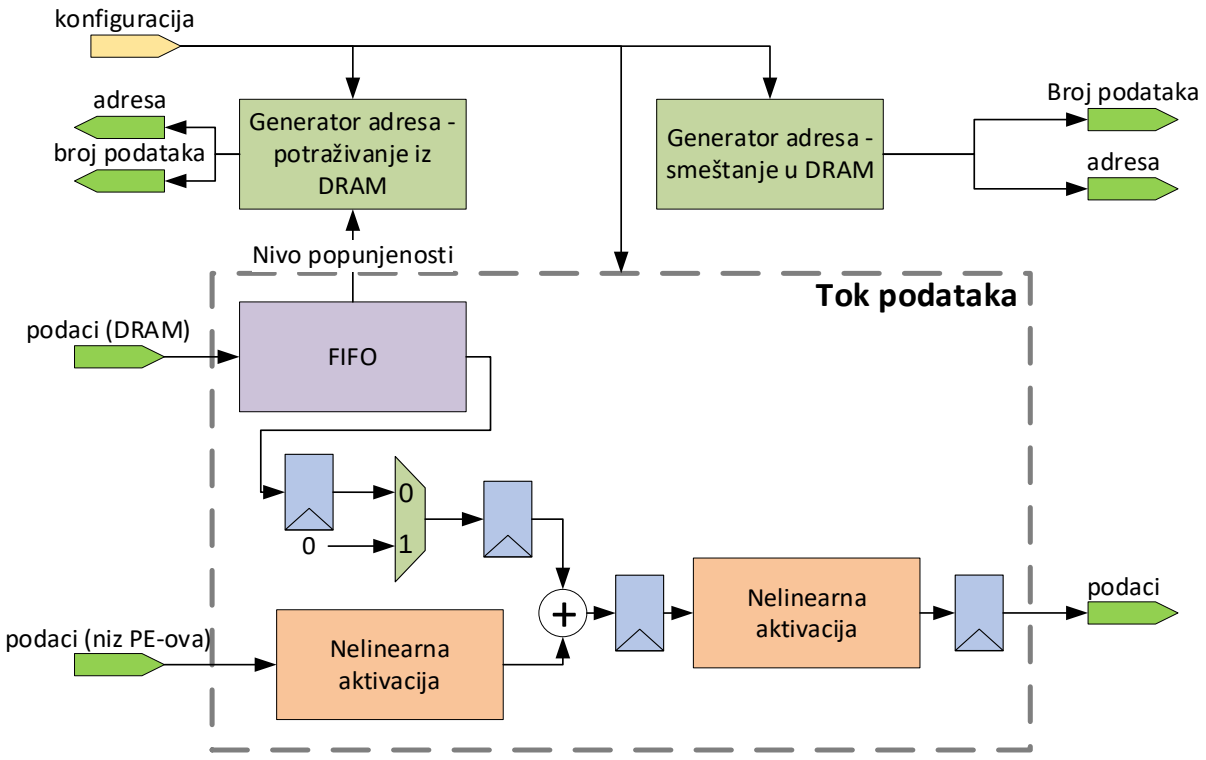


Slika 46 Mikroarhitektura kolektora rezultata.

Kako bi se ostvarilo efikasno zaokruživanje, bez remećenja performansi sistema (frekvencije), potrebno je proces pomeranja/zaokruživanja podeliti u nekoliko koraka. Ti koraci su predstavljeni *pipeline* strukturom koju čine pomerači i registri. Prvi i drugi nivo vrši pomeranje u većim koracima, od po 0, 4, 8 ili 12 mesta. Poslednji nivo ima mogućnost pomeranja sa korakom 1. Po završetku pomeranja, potrebno je izvršiti i zaokruživanje rezultata. U slučaju CNN-ova, rezultati su uglavnom neoznačeni brojevi zbog ReLU funkcije, tako da se ova operacija može izvesti samo jednim sabiračem. Ipak, u slučaju da je potrebno podržati i negativne brojeve, logika postaje nešto kompleksnija. Posle pomeranja i zaokruživanja, rezultati se prosleđuju OS modulu u grupama od po osam rezultata. Još jednom, arhitektura kolektora, kao i svakog bloka na putanji koja izračunava skalarne proizvode, mora da ima odgovarajuću propusnu moć kako ni jedna komponenta ne bi predstavljala usko grlo prilikom obrade podataka.

6.6 Izlazni tok podataka (OS)

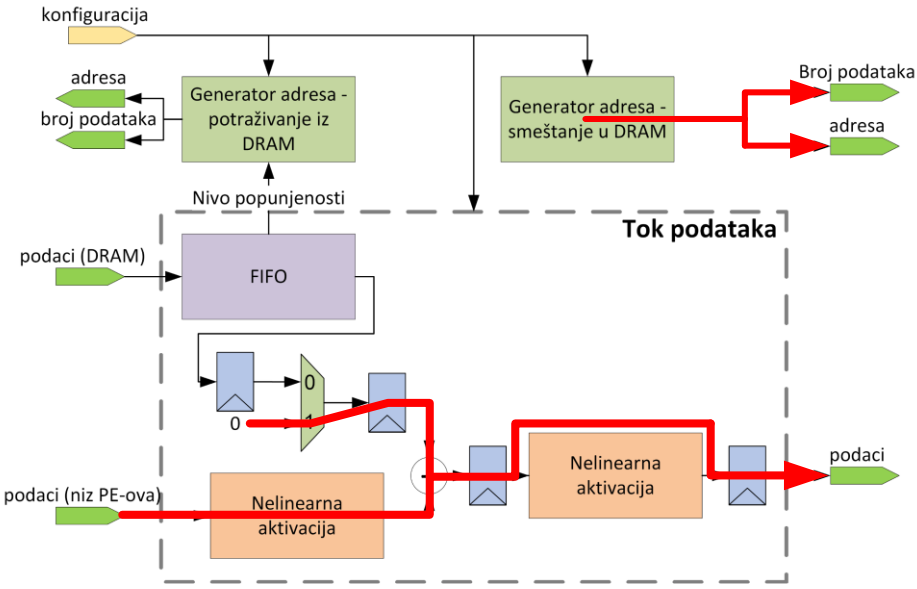
Osnovna uloga OS bloka je preuzimanje podataka od niza PE-ova i njihovo smeštanje u DRAM. Uz navedeno, OS po potrebi primenjuje aktivacione funkcije na ulazne podatke, ali ima i značajnu ulogu prilikom procesiranja slojeva sabiranja, odnosno parcijalnih konvolucija.



Slika 47 Mikroarhitektura izlaznog toka podataka (OS modul).

Slika 47 predstavlja mikroarhitekturu OS bloka podeljenu na kontrolni deo i tok podataka. Kontrolni deo čine dve mašine stanja, jedna za generisanje zahteva za smeštanje podataka u DRAM i druga za potraživanje podataka iz DRAM-a u slučaju parcijalne konvolucije, odnosno prilikom procesiranja nekih slojeva sabiranja.

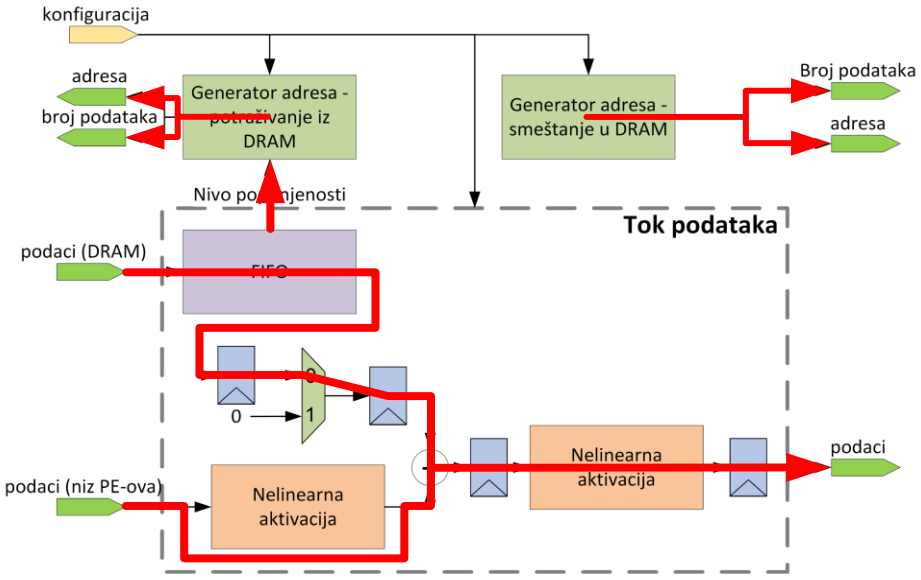
Najčešće korišćena konfiguracija OS modula se koristi prilikom procesiranja konvolucionih slojeva kod kojih nije potrebno primeniti parcijalnu konvoluciju. Aktivni delovi OS bloka u ovoj konfiguraciji su označeni crvenim linijama na slici 48. Od generatora adresa, aktivan je samo onaj koji kreira zahteve za smeštanje podataka u vidu početne adrese štapića u DRAM-u i njihovih veličina izražene u bajtima. Ovakav zahtev se prosleđuje *Data Mover*-u (DM) zajedno sa podacima koji dolaze iz toka podataka. Podaci se po potrebi obrađuju prvim blokom za nelinearnu aktivaciju i šalju na sabirač. Sabirač je u ovoj konfiguraciji neaktivan tako što je na njegov drugi ulaz doveden sabirak sa vrednošću nula. Drugi blok za nelinearnu aktivaciju se zaobilazi i rezultat se prosleđuje DRAM-u. Napomenimo i to da su sve magistrale podataka u okviru toka podataka veličine 8x16 bita.



Slika 48 Aktivne putanje prilikom procesiranja konvolucionih slojeva bez parcijalne konvolucije i sabiranja.

Za razliku od opisanih konvolucija, u slučaju parcijalne konvolucije, potrebno je uključiti i generator za potraživanje delova konvolucija iz DRAM-a i pripadajući tok podataka. Aktivni blokovi u ovom slučaju su prikazani na slici 49. Generator DRAM zahteva koji potražuje prethodno izračunate delove konvolucije kreće sa radom zajedno sa nizom PE-ova. Dok niz PE-ova računa prve rezultate preostalih delova konvolucije, DRAM kontroler odgovara na

zahteve i puni se FIFO bafer. Pošto tekuće računanje poslednjeg dela parcijalne konvolucije traje značajno duže od učitavanja prethodno izračunatih delova konvolucije, rad generatora DRAM zahteva je kontrolisan nivoom popunjenosti FIFO bafera. Kada se u baferu nalazi manje od 32 prethodno izračunata podatka, generator dobija signal da učita naredni niz iz DRAM-a. Primetimo da je prag popunjenosti bafera, 32, odabran da bude identičan PE_Num. Na ovaj način OS neće biti usko grlo jer će podaci uvek biti spremni te neće kočiti niz PE-ova zato što će OS uvek biti spreman da prihvati podatke. Naravno, navedeno važi samo u slučaju da DRAM kontroler može da prihvati podatke na izlazu OS jedinice.



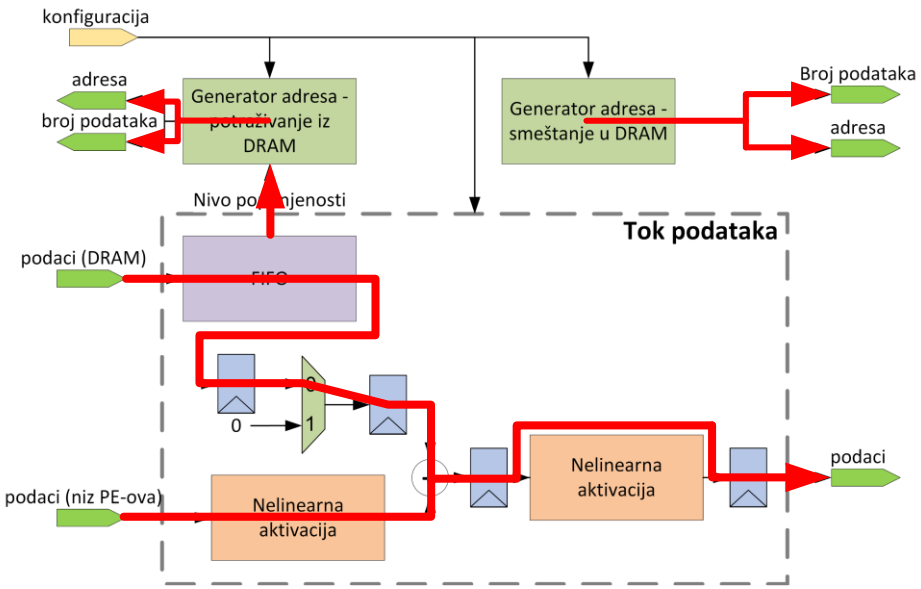
Slika 49 Aktivne putanje prilikom procesiranja parcijalnih konvolucija.

Na slici 49 se može uočiti da se u slučaju parcijalnih konvolucija zaobilazi prvi blok za nelinearnu aktivaciju. Ovo je potrebno uraditi kako bi se poslednji deo konvolucije sabrao u izvornom obliku sa prethodno izračunatim delom parcijalne konvolucije. Tek po sabiranju ova dva dela u konačan rezultat skalarnog proizvoda moguće je primeniti aktivacionu funkciju.

Iako to nije bio slučaj sa CNN-ovima koji su korišćeni za testiranje performansi Argus akceleratora, moguće je imati više od jednog međurezultata u toku računanja parcijalne konvolucije. Opisani proces je potrebno primenjivati prilikom skladištenja svakog dela konvolucije počevši od drugog. Na ovaj način Argus arhitektura podržava proizvoljan broj delova parcijalne konvolucije što znači da veličina kernela, to jest, dubina IFM-a ne predstavlja ograničenje koje može da diskvalifikuje arhitekturu da akcelerira bilo koji CNN model.

Primetimo da je u slučaju računanja prvog dela parcijalne konvolucije potrebno zaobići i izlaznu aktivacionu funkciju kako se ne bi narušila vrednost međurezultata.

Opisani tok podataka se može iskoristiti i za računanje rezultata određenih slojeva sabiranja. Proces je takav da se rezultati prethodno izračunatih konvolucija (kompletnih, ne delova parcijalne konvolucije) potražuju iz DRAM-a na isti način kao i delovi parcijalne konvolucije i šalju na sabirač preko FIFO bafera. Sa druge strane, niz PE-ova računa tekuću konvoluciju na koju OS primenjuje aktivacionu funkciju i rešenje dostavlja sabiraču kao drugi sabirak. U ovom trenutku, na sabiraču se nalaze dve vrednosti konvolucija koje pripadaju različitim slojevima unutar CNN modela. Sabiranjem ovih rezultata dobijamo odziv sloja sabiranja koji prati tekući konvolucioni sloj. Ovakav pristup skoro u potpunosti maskira vreme potrebno za računanje slojeva sabiranja koji se na ovaj način mogu procesirati. Od poznatih CNN modela, najviše benefita od ovakvog pristupa mogu imati mreže koje liče na ResNet CNN-ove.

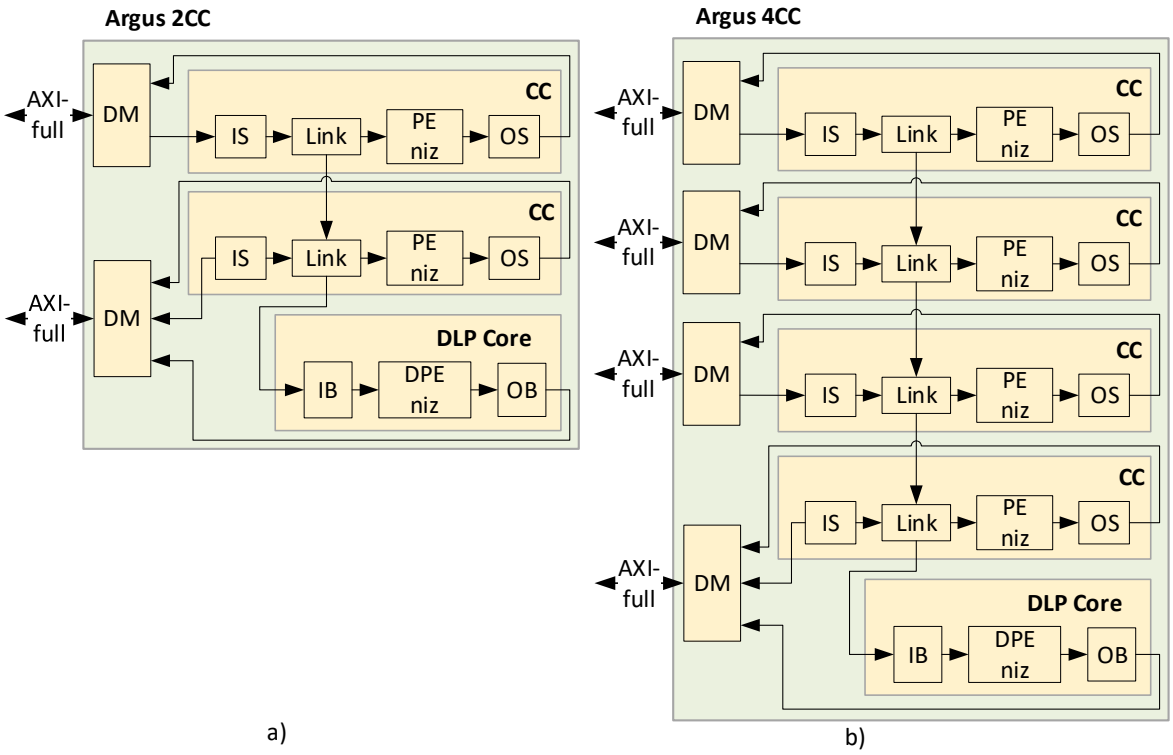


Slika 50 Računanje rezultata pojedinih slojeva sabiranja zajedno sa računanjem konvolucije.

6.7 Multi-core konfiguracija Argus akceleratora

Iako je jedan CC blok visoko optimizovan za efikasno računanje konvolucionih slojeva, postoje aplikacije u kojima njegove performanse nisu dovoljne. Svaki PE čine po dva DSP bloka, ukupno 64 u jednom CC modulu. Ukoliko rade na učestanosti od 500 MHz i pri tom je mreža orezana, jedan CC će teoretski imati performanse ekvivalentne 64 GMAC/s. U praksi ovo znači između 3 i 4 klasifikacije u sekundi ukoliko se slika procesira VGG16 CNN modelom. Ukoliko to nije dovoljno za određenu primenu, moguće je povezati nekoliko CC modula u takozvanu,

Multi-core konfiguraciju i time postići veći stepen paralelizacije. Na slici 51 je prikazan Argus akcelerator koji ima dva (a) odnosno, četiri CC bloka (b).



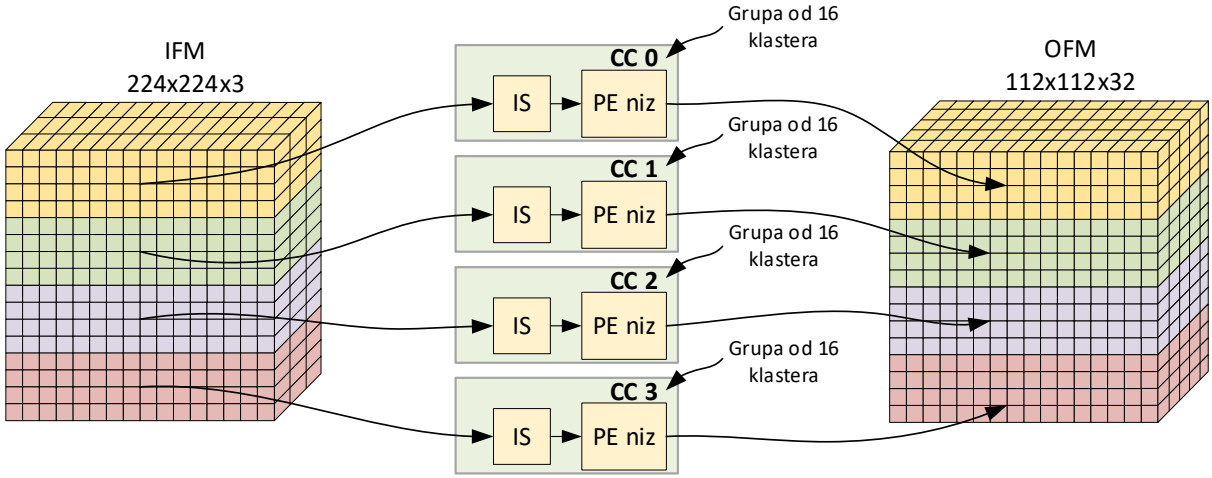
Slika 51 Multi-core konfiguracija akceleratora sa 2 (a) i 4 CC modula (b).

Svaki CC modul se povezuje sa DRAM kontrolerom preko zasebnog DM-a i pripadajućeg AXI-full interfejsa. Na pristupačnim FPGA baziranim SoC sistemima, DRAM kontoleri sa najmanje 4 AXI-full interfejsa su uobičajeni što znači da ni Argus arhitektura u konfiguraciji sa 4 CC modula neće predstavljati problem da se poveže na DRAM. Naravno, u slučaju nedostatka portova, moguće je projektovati odgovarajuće module koji bi arbitrirali AXI-zahteve čime bi se omogućilo povezivanje gotovo proizvoljnog broja CC blokova na memorijski sistem. Naravno, očekivati je degradaciju performansi ukoliko se značajno uvećava broj CC modula jer DRAM neće biti u stanju da opsluži sve CC blokove zbog limitirane propusne moći. Navedimo i to da povećavanje broja DLP jezgara neće imati značajan uticaj na performanse akceleratora. Razlog za to treba tražiti u činjenici da slojevi koje akcelerira ovaj deo Argus akceleratora uzimaju značajno manji vremenski udeo u odnosu na konvolucione slojeve u poznatim CNN-ovima.

Ukoliko Argus arhitektura ima više od jednog CC modula posledično je u stanju da procesira veći broj kernela u paraleli. Tako Argus akcelerator sa dva CC bloka može da obrađuje do 64

različita kernela istovremeno dok Argus sa četiri CC modula ima ima mogućnost procesiranja 128 kernela istovremeno. Ako je instanciran Argus akcelerator sa četiri CC bloka i ukoliko se procesira sloj koji ima 128 ili više kernela, akcelerator će raditi u režimu koji je opisan i za jedan CC. To znači da će svaki CC modul učitati svoje kernele, po 32 svaki, i da će svaki CC procesirati isti deo IFM-a samo sa različitim kernelom. U ovakvim slojevima Argus arhitektura sa četiri CC modula će biti efikasan kao i Argus sa jednim CC-om u smislu uposlenosti PE-ova.

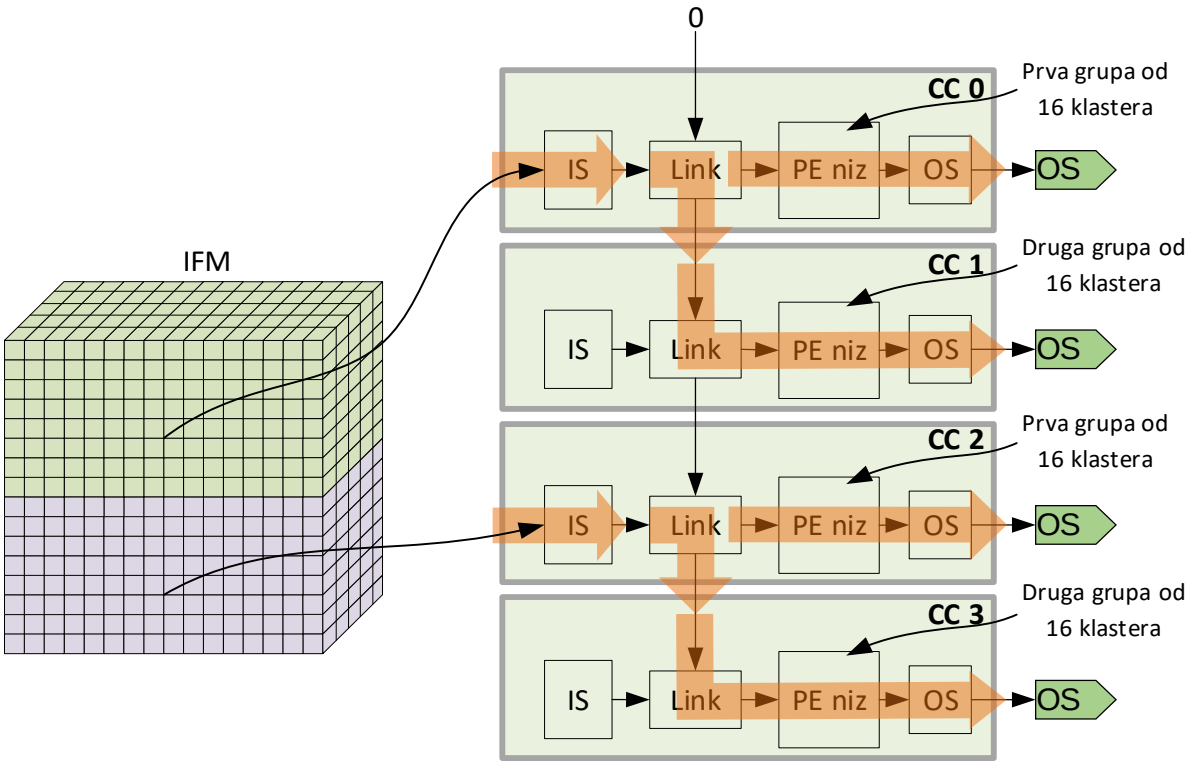
Navedeni odnos konfiguracije akceleratora i konvolucionih slojeva nije uvek ispunjen, posebno u početnim slojevima CNN mreža. Prvi sloj velike većine CNN-ova čine 32 kernela. Da bi se i u slučaju ovakvih slojeva zadržala visoka efikasnost, moguće je procesirati različite delove IFM-a različitim CC blokovima. Na primer, CC0 može da obrađuje gornju četvrtinu IFM-a, CC1 drugu četvrtinu od gore i tako redom. Podela IFM-a na četvrtine je prikazana različitom bojom na slici 52.



Slika 52 Efikasno procesiranje konvolucionog sloja sa 32 kernela pomoću Argusa sa 4 CC modula.

Posle konfigurisanja CC modula, počinje paralelno učitavanje ista 32 kernela u svaki od CC-ova. Po završetku učitavanja svaki CC ima identičnu kopiju parametara kernela i počinje obrada IFM-a. Svaki IS dobija drugačiju početnu adresu koja predstavlja adresu gornjeg levog štapića svake četvrtine IFM-a. Paralelno računanje konvolucije je praćeno simultanim radom pridruženih portova na DRAM kontroleru. Pošto nema deljenja ulaznih podataka između CC modula, ovakvi slojevi su najzahtevniji u pogledu propusne moći DRAM kontrolera. Takođe, zbog povećanog nivoa paralelizacije, veće opterećenje je prisutno i na strani upisa u DRAM. Povrh svega, primetimo da slojevi sa manjim brojem kernela obično obrađuju plitke IFM-ove što povlači za sobom činjenicu da računanje pojedinačnih tačaka OFM-a ne traje dugo te je

količina podataka na izlazu značajno veća nego u slučaju slojeva koji se nalaze dublje unutar mreže. Na sreću slojevi sa 32 kernela su prisutni samo na početku mreže, a najčešće je to slučaj samo sa prvim slojem. Već sledeći sloj uglavnom ima 64 ili više kernela što drastično smanjuje pritisak na strani čitanja podataka iz DRAM-a. Takođe, računanje izlaznih rezultata traje duže te je smanjena i količina podataka na izlazu Argus akceleratora ka DRAM-u. Ovakav slučaj je prikazan na slici 53. Slika je detaljnija kako bi se jasno uočila uloga *Link* bloka i redukcija razmene podataka između DRAM-a i Argus arhitekture.



Slika 53 Efikasno procesiranje konvolucionog sloja sa 64 kernela pomoću Argus akceleratora sa 4 CC modula.

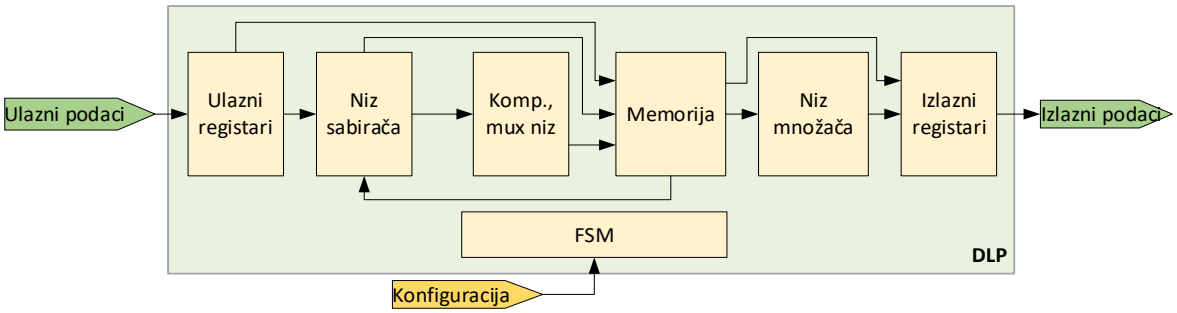
Na početku, CC0 učitava parametre prvih 32 od 64 kernela. Simultano CC1 preko svog IS modula učitava drugih 32 kernela. Isti postupak konfigurisanja se dešava u CC modulima 2 i 3. Po završetku učitavanja parametara sloja, CC0 će imati kopiju parametara identičnu CC-u 2. Takođe, sadržaj interne memorije CC modula 1 i 3 će biti isti. Sledeći korak je procesiranje IFM-a koje se u ovom slučaju može podeliti na dva dela. Gornja polovina IFM-a će se obraditi pomoću CC modula 0 i 1 dok će ljubičasta (donja) polovina biti procesirana CC blokovima 2 i 3. Ukoliko znamo da CC moduli 0 i 1 izračunavaju odziv nad istim delom IFM-a, istim snopom štapića, jasno je da nije potrebno duplirati zahteve ka DRAM-u već je to moguće uraditi pomoću samo jednog IS modula. Ulogu keširanja IFM-a za CC module 0 i 1 će preuzeti IS

unutar CC0, dok će za CC-ove 2 i 3 ovu ulogu imati IS koji pripada CC-u 2. IS unutar CC bloka 0 će slati redom snopove štapića ka svom PE nizu dok će *Link* blok iste podatke prosleđivati ka CC modulu 1. *Link* modul CC bloka 1 će dalje proslediti IFM ka svom PE nizu koji će raditi paralelno sa PE nizom unutar CC-a 0. Na ovaj način IS unutar CC bloka 1 je u stanju mirovanja čime se izostavlja nepotrebna razmena podataka sa DRAM-om. Naravno, ista operacija se odvija unutar CC modula 2 i sa razlikom da se učitava donja polovina IFM-a. Primetimo da se ista optimizacija deljenja podataka između CC blokova može izvesti za slojeve koji imaju 128 ili više kernela. U slučaju da sloj ima 128 kernela, sva četiri *Link* modula će biti uposlena. *Link* blok unutar CC modula 0 će slati podatke Linku unutar CC bloka 1 koji će dalje prosleđivati podatke ka CC modulu 2 i tako redom koliko god da ima CC blokova u akceleratoru.

6.8 DLP jezgro

DLP (engl. *Dense Layer Processing*) je zaseban deo Argus akceleratora specijalizovan za izračunavanje *max pooling*, *average pooling* i sloja sabiranja (engl. *Adding*). Kako bi se postigla visoka propusna moć, modul je *pipeline*-ovan tako da ima šest nivoa protočne obrade kao na slici 54. U zavisnosti od konfiguracije sloja koji se obrađuje, neki od nivoa protočne obrade se mogu zaobići. Izračunavanje odziva sloja se uvek radi u tri koraka. U koraku jedan se prvi štapić iz tekućeg snopa štapića učitava u blok koji se zove memorija. Učitavanje prvog štapića se vrši preko ulaznih registara. Sledeći korak predstavlja procesiranje svih ostalih štapića trenutnog snopa. Ukoliko je u pitanju *max pooling* sloj drugi štapić će se porediti sa prvim koji je učitan u koraku jedan. Zatim će se međurezultat smestiti u internu memoriju. Potom će se treći štapić učitavati i upoređivati sa prethodno izračunatim međurezultatom iz memorije. Rezultat ovog poređenja se takođe smešta u memoriju i tako redom. Po obradi svih štapića tekućeg snopa krajnji rezultat se preko izlaznih registara šalje izvan DLP jezgra, to jest, u DRAM. Vredi navesti da se u prvom koraku uvek koriste samo ulazni registri i memorija kako bi se ista inicijalizovala početnim vrednostima prvih štapića svakog snopa, nezavisno od sloja koji se računa.

Kako bi se iskoristio pun potencijal ulazno-izlazne magistrale od 128 bita, sve jedinice unutar DLP modula su vektorizovane tako da u svakom taktu mogu da obrađuju po osam 16-bitnih vrednosti. Takođe, kompletna aritmetika unutar DLP jezgra je 16-bitna sa izuzetkom memorije i niza množača koji je zbog prirode operacije 24-bitan.



Slika 54 Blok dijagram DLP jezgra.

U slučaju *max pooling* sloja u prva dva koraka su aktivna prva četiri bloka sa slike 54. U prvom koraku se inicijalizuje memorija preko ulaznih registara. U drugom koraku se računa razlika između učitanog štapića, odnosno međurezultata u memoriji pomoću niza sabirača. Rezultat se prosleđuje komparatorskom nizu koji odlučuje koje rezultate će smestiti u memoriju. Po završetku obrade poslednjeg štapića iz snopa, aktivira se izlazni niz registara preko kojih konačni rezultati napuštaju DLP blok.

Prvi korak *average pooling*-a je identičan onom u slučaju *max pooling* sloja. Memorija se inicijalizuje vrednostima prvog štapića trenutnog snopa. U drugom koraku se učitava naredni štapić koji se dovodi na niz sabirača. Kao drugi sabirak se isčitava prvobitno upisani štapić u memoriju. Zbir ova dva štapića se upisuje u memoriju. Dalje, preko ulaznih registara se učitava treći štapić koji se dovodi do sabirača. Prethodno izračunata tekuća suma se čita iz memorije i sabira sa trećim štapićem. Proces se ponavlja dokle god se ne saberu svi štapići unutar trenutnog snopa. Po završetku sabiranja, rezultati iz memorije se dovode na niz množača koji rade usrednjavanje tako što se zbir svih štapića snopa množi sa recipročnom vrednošću broja štapića unutar snopa. U slučaju da je snop dimenzija 2x2, koeficijent množenja će biti 0.25. Naravno, ovaj broj je preračunat u 16-bitnu aritmetiku sa fiksnim zarezom i preko konfiguracionog ulaza doveden do niza množača. Poslednji korak je da se izlazi niza množača smeštaju u DRAM preko niza izlaznih registara.

Takođe, i sloj sabiranja ima identičan prvi korak. Ovog puta se memorija inicijalizuje štapićima iz prvog IFM-a, označimo ga sa IFM0. U drugom koraku se učitava štapić koji odgovara učitanom, ali iz drugog IFM-a, označimo ga sa IFM1. Štapić IFM-a 0 se iz memorije dovodi na sabirač gde se sabira sa štapićem iz IFM-a 1. Ukoliko je sloj takav da se sabiraju samo dva IFM-a, onda se rešenje smešta u memoriju, da bi iz iste bio prosleđen na izlaz preko izlaznih registara. Ukoliko je potrebno sabrati više od dva IFM-a, rezultat sabiranja se smešta u

memoriju gde čeka da se učita odgovarajući štapić iz sledećeg IFM-a. Ovaj proces se može ponavljati proizvoljan broj puta, to jest, dokle god se ne saberu svi IFM-ovi koji participiraju u datom sloju.

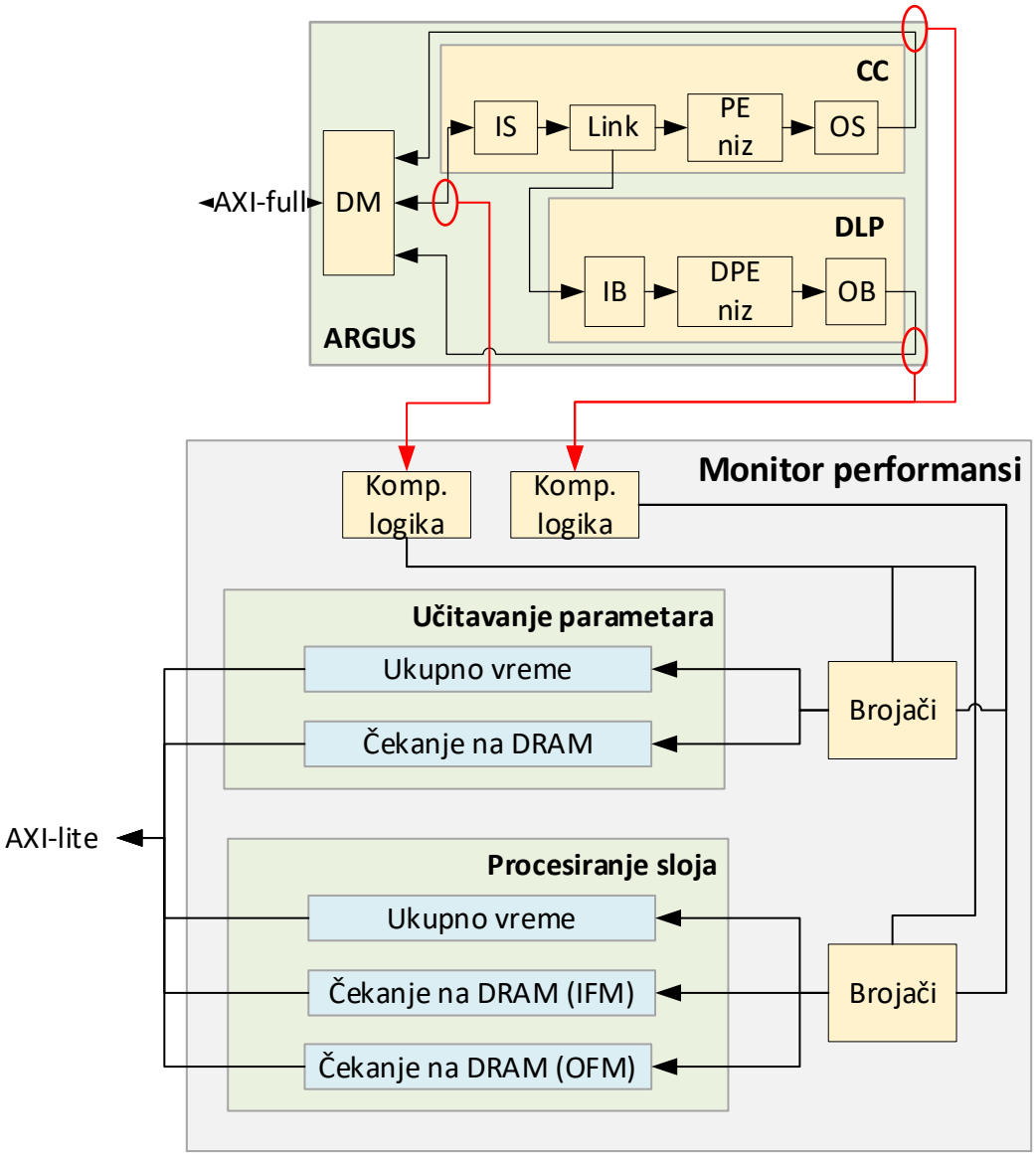
6.9 Monitor performansi

Da bi se evaluirale tačne performanse akceleratora potrebno je implementirati modul koji će vršiti nadgledanje procesiranja na FPGA čipu. Po obradi svakog sloja, monitor će sačuvati indikatore uposlenosti magistrala koje je moguće pročitati preko AXI-lite interfejsa pristupom registarskoj mapi. Ovi indikatori su dovoljni za računanje efikasnosti arhitekture. Jezgro ovog monitora čini nekoliko registara, brojača i jednobitni komparatori te kao takav predstavlja zanemarljivo uvećanje celokupnog akceleratora. Sa druge strane daje egzaktne podatke o performansama Argus akceleratora zajedno sa okružujućim memorijskih sistemom. Ovo je potrebno uraditi kako bi se video uticaj ograničene propusne moći DRAM kontrolera na simulacione (teoretske) performanse akceleratora. Dodatni razlog, koji nameće potrebu uračunavanja performansi DRAM kontrolera u ukupne performanse, potiče od ideje da Argus akcelerator treba da bude implementiran unutar embeded sistema i u istima ostvari performanse koje prevazilaze postojeća rešenja. Drugim rečima, cilj disertacije je da se pokaže da ovakav pristup zaista ima prednosti u odnosu na dosadašnje u okviru postojećih, a ne hipotetičkih sistema.

Da bi se pronašla uska grla, koja onemogućavaju postizanje maksimalnih teoretskih performansi, potrebno je nadgledati isključivo ulazne i izlazne magistrale. Ovakav pristup je moguć jer je akcelerator projektovan tako da unutrašnje komponente imaju identičnu, uparenu, propusnu moć za sve relevantne/poznate slojeve. Jedan od ekstremnih primera koji je uzet u razmatranje prilikom projektovanja arhitekture je prvi Pointwise sloj u MobileNet CNN-u. Ovaj sloj je takav da zahteva stoprocentno iskorišćenje ulazne i izlazne magistrale akceleratora. Ukoliko magistrale budu u stanju da ispoštuju ovakav pritisak, Argus CNN akcelerator će dostići teoretski maksimum performansi. Performanse i efikasnost samog akceleratora je moguće proveriti u okviru simualcije RTL-a tako što će se ulazi i izlazi maksimalno opsluživati od strane simulacionog modela. Napomenimo i to da se Argus arhitektura ponaša kao master i u smislu magistrala i generalno jer inicira sve transfere i procesiranje. Zbog navedenog, akcelerator će pokušati što bolje da iskoristi dostupne propusne moći memorijskog sistema. Iz navedenog, jasno je da je efikasnost Argus arhitekture

na FPGA sistemu, moguće utvrditi posmatrajući samo naznačene magistrale na slici 55 i to sa izuzetnom tačnošću.

Na slici 55 se mogu uočiti dva podbloka, jedan za monitoring performansi prilikom učitavanja parametara modela i drugi koji nadgleda magistrale u toku konvoluiranja IFM-a i kernela. Blok koji prati transakcije u toku učitavanja parametara ima dva registra, jedan koji skladišti ukupno vreme potrebno za učitavanje parametara sloja i drugi koji prijavljuje koliko se dugo čekalo na DRAM da isporuči zahtevane podatke. Naravno, vreme je u ovom slučaju izraženo u broju taktova te je prilikom očitavanja vrednosti svih registara unutar monitora performansi potrebno pomnožiti ih sa trajanjem jedne periode takt signala (4 ns).



Slika 55 Monitor performansi Argus akceleratora na čipu.

Blok na slici 55, koji je aktivan u toku konvoluiranja, označen kao ,,Procesiranje sloja”, sadrži 3 registra. Prvi koji beleži ukupno vreme potrebno za obradu sloja. Drugi koji daje informaciju koliko taktova je Argus akcelerator bio spreman da prihvati tražene podatke (IFM), ali DRAM nije mogao da odgovori i treći, u kome je sačuvan podatak o vremenu provedenom čekajući na DRAM da preuzme izlazne rezultate sloja. Uz navedene registre, modul čine i brojači za sve od navedenih podataka, komparatorska logika koja nadgleda *valid* i *ready* signale AXI-stream magistrala koje se posmatraju. Iako su na slici 55 registri (plavi pravougaonici) prikazani da pripadaju registarskoj banci monitora performansi, u okviru stvarne implementacije oni su deo globalne registarske banke akceleratora kojoj procesorski sistem pristupa preko AXI-lite interfejsa.

Jasno je da je za merenje apsolutnih performansi dovoljno samo zbirno vreme provedeno prilikom učitavanja parametara i konvoluiranja, međutim dodatni indikatori direktno ukazuju na to šta je najveća prepreka u postizanju teoretskih maksimuma arhitekture. Ovo naročito može biti interesantno u toku procesiranja sloja jer se jednostavno može uočiti da li procesiranje više usporava čitanje ili upisivanje u DRAM što dalje može biti korisno za analizu rada DRAM kontrolera. Takođe, navedeni podaci mogu biti od koristi ukoliko se odluči za projektovanje izlaznih bafera između akceleratora i DRAM-a. Ipak, dalja analiza memorijskog sistema nije deo ove disertacije.

7 Funkcionalna verifikacija i hardversko testiranje akceleratora

Kako bi se proverila funkcionalna korektnost, razvijeni akcelerator je podvrgnut testiranju, to jest, procesu koji se naziva verifikacijom hardvera. Pored funkcionalne korektnosti, simulacije u okviru simulatora mogu pomoći u otkrivanju mana arhitektura i uskih grla. Uz funkcionalnu verifikaciju (naziv koji se najčešće koristi za proveru RTL-a u okviru simulatora), Argus CNN akcelerator je podvrgnut i testiranju na stvarnom hardveru, tačnije, Xilinx MPSoC čipu oznake ZCU104. Testiranje na hardveru potvrđuje ispravnost prethodnog testiranja u simulatorima. Dodatno, u okviru testiranja na čipu se otkrivaju obično greške vezane za integraciju akceleratora u sistem, ali se mogu očitati i performanse kompletnog sistema u realnim uslovima eksploatacije. U nastavku, poglavlje je podeljeno na funkcionalnu verifikaciju u kome je opisan postupak, korišćeni alati i prikazani najinteresantniji digitalni talasni oblici signala Argus arhitekture. Drugi deo poglavlja donosi verifikaciju na pomenutom hardveru, način testiranja i takođe, karakteristične talasne oblike, ali ovog puta sa akcentom na integraciju akceleratora u sistem.

7.1 Funkcionalna verifikacija – teorijske osnove

Iako verifikacijia nije u fokusu ove disertacije, proces testiranja hardvera ove kompleksnoti je nezaobilazan deo razvoja. Generalno gledano, verifikacija je postala najzahtevniji deo u razvoju današnjih čipova. Nagli rast kompleksnosti čipova je ispraćen naglim razvojem verifikacionog dela grane privrede. Svedoci smo postojanja kompanija koje se specijalizovano bave verifikacijom na raznim nivoima, što je do samo pre jedne decenije bilo gotovo nezamislivo. Čak i sama pozicija verifikacionog inženjera nije bila toliko popularna, da bi danas, broj ovako profilisanih inženjera uveliko prevazišao broj dizajn inženjera. Navedeno je, pored rasta kompleksnosti čipova, uzrokovano i visokom cenom razvoja i proizvodnje. Pronalazak grešaka (odstupanja hardvera od specifikacije) u ranoj fazi razvoja značajno smanjuje troškove razvoja nego kada se isti pronalaze u proizvedenom čipu. Ovo je toliko značajno da verifikacioni tim počinje razvoj simulacionog okruženja uporedo sa dizajn timom koji projektuje čip. Gotovo uvek je slučaj da verifikacioni tim broji veći broj inženjera od dizajn tima što donekle oslikava kompleksnost zadatka verifikacije.

Naravno, verifikacija ASIC i FPGA čipova se donekle razlikuje, ali osnovni ciljevi i koraci su identični. Oba procesa počinju od funkcionalne specifikacije čipa iz koje se kreira verifikacioni

plan. Drugi korak u nizu je razvoj verifikacionog okruženja da bi se u trećem koraku okruženje srelo sa RTL-om čime započinje proces simulacije i otklanjanja grešaka. U ovom procesu se razvijaju i regresije testova koje se iznova pokreću prilikom većih promena unutar RTL-a. Na ovaj način dizajn i verifikacioni tim preveniraju unošenja dodatnih grešaka u postojeći kod tako što regresija daje indikaciju da li su prethodno korektne funkcionalnosti sada narušene (neki testovi koji su prolazili bez grešaka sada prijavljuju neslaganja). Od ovog koraka se postupak verifikacije u ASIC i FPGA tehnologijama najčešće razlikuju, tako što se u slučaju ASIC verifikacije nastavlja sa rigoroznim simuliranjem koje uključuje i tehničke karakteristike logičkih kapija i ostalih ćelija koje se koriste. Uz navedeno, dizajn tim pokreće i linter, alat koji otkriva razne nepravilnosti, od pogrešne sinhronizacije između klok domena, preko grešaka u korišćenju aritmetičkih operanada do raznih drugih problema koji mogu biti uzroci nepravilnog funkcionisanja čipova. Naravno, ovaj postupak je moguće i poželjno primeniti i na dizajn koji je namenjen FPGA platformama, ali se u praksi često ne izvodi. Osnovni razlog je cena uklanjanja grešaka prilikom produkcije. U slučaju ASIC čipova, probni čipovi dolaze u laboratorije na proveru gde bivaju podvrgnuti proveri pomoću odabranog skupa testova. Ukoliko se otkrije greška koja ima značajan uticaj na funkcionalnost čipa, potrebno je isti ispraviti u RTL-u, proći kroz sve procese verifikacije i utvrditi zašto je ista promakla, te na posletku proizvesti novi čip. Sa druge strane, otkrivenu grešku na FPGA platformama je potrebno ispraviti u RTL-u, proći kroz postupak verifikacije i generisati novi fajl koji će biti učitan u FPGA kolo. Trajanje ispravke grešaka na FPGA kolima se meri od nekoliko sati do nekoliko dana u zavisnosti od kompleksnosti, dok je to u slučaju ASIC čipova najčešće nekoliko meseci.

Kako bi se navedeni postupci ubrzali, ali i učinili otpornijim na greške, konstantno se razvijaju nove metodologije i jezici za verifikaciju hardvera. Jedan od prethodno najrasprostranjenijih jezika, koji je i dalje u upotrebi je ,,e“ i prateća metodologija skraćenog naziva eRM (engl. *e Re-use Methodology*). Iako je doživela široku upotrebu, danas je polako istiskuje UVM (engl. *Universal Verification Methodology*) i SystemVerilog kao jezik. Suštinski gledano, sve do sada razvijene metodologije dele istu osnovu. Gledano sa visokog nivoa apstrakcije, u svakom verifikacionom okruženju možemo prepoznati komponente koje pobuđuju ulazne signale DUT-a (engl. *Design Under Test*), blokove koji prikupljaju izlaze DUT-a i jedinice koje vrše proveru korektnosti prikupljenih rezultata (engl. *checker*). Najkompleksniji delovi

verifikacionog okruženja su najčešće *checker*-i koji pored provere izlaznih rezultata DUT-a u sebi sadrže i model DUT-a pisan od strane verifikacionog tima na visokom nivou apstrakcije.

Odabir metodologije i način testiranja zavise od nekoliko faktora. Najbitniji je svako tip hardvera koji se testira u smislu da li je DUT kontrolno orijentisan i uglavnom upravlja okružujućim komponentama ili ima zadatak da procesira velike količine podataka određenim algoritmita. Uz tehničke karakteristike DUT-a, u današnje vreme i raspoloživost, ali i obučenost ljudskih resursa ima veliki uticaj na odabir načina testiranja. Usled trenutnog nedostatka kadra, i same preferencije inženjera igraju bitnu ulogu u odabiru metodologije.

U slučaju ove doktorske disertacije, ne postoji verifikacioni tim koji bi vršio testiranja. Aplikacija akceleratora je orijentisana na procesiranje velikog broja podataka što znači da nije potrebno praviti kompleksne verifikacione komponente koje bi pobuđivale ulazne signale u tačno određenim vremenskim intervalima. Zbog navedenog, verifikacija RTL-a pomoću simulatora je izvršena uzimajući u obzir osnove dobrog verifikacionog okruženja, a to su razdvajanje okruženja koje stimuliše ulazne signale, dela koji prikuplja rezultate i blokova koji proveravaju korektnost rezultata. Kako bi se dodatno ubrzao postupak testiranja, RTL verifikacija je izvedena pomoću C programskog jezika. SystemVerilog, koji je korišćen za pisanje *testbench*-a, ima mogućnost poziva C funkcija koje su korišćene za pripremu i proveru podataka. C jezik nije izvorno namenjen verifikaciji hardvera te u tom smislu i vreme potrebno za kreiranje okruženja nije optimalno. Ipak, razvijene C funkcije su bile ponovno korišćene prilikom testiranja akceleratora na FPGA razvojnom okruženju gde nije bilo potrebno značajno modifikovati iste. Pored toga što navedene funkcije nije bilo potrebno ponovno pisati, one su i proverene u okviru simulatora gde su otklonjeni svi poznati nedostaci. Ovako testirane funkcije značajno smanjuju vreme verifikacije na hardveru. Razlog za to proizilazi iz činjenice da ukoliko se na hardveru pojave greške možemo sa velikom sigurnošću tvrditi da su uzroci negde na strani hardvera, a ne u softveru. Kako bi se dodatno ubrzao proces testiranja, model DUT-a je preuzet iz Pythona, tako što su ekstrahovane ulazne i izlazne matrice slojeva direktno, uz korišćenje postojećih biblioteka. Na ovaj način su izbegnute greške u modelu koje bi neminovno bile prisutne u slučaju da se model pisao u okviru C-a. Ukratno, C funkcije na zahtev DUT-a dovode ulaznu matricu na ulaz Argus arhitekture, a prikupljene rezultate porede sa izlaznom matricom koja je, kao i ulazna, pripremljena uz pomoć Python modela CNN-a.

Ovakav vid testiranja je poznat i kao verifikacija pomoću zlatnih vektora (engl. *Golden vector approach*) koji je gotovo idealan za ovakav tip hardvera.

Kao što je navedeno, verifikacija je u najvećoj meri zadatak verifikacionog tima, međutim, neke delove često obavljaju dizajneri, ali i softverski inženjeri. Podela se najčešće vrši na osnovu nivoa na kome se verifikacija sprovodi. Najčešća podela verifikacije je na sledeće nivoe:

1. Dizajnerski nivo – Dizajner razvija jednostavano okruženje, najčešće pomoću nekog HDL jezika i proverava bazičnu fukncionalnost dela dizajna. Ovakve provere se najčešće vrše vizuelnom inspekcijom talasnih oblika pomoću simulatora. Greške koje se otkrivaju nisu kompleksni i obično su to nepovezani signali, pogrešan rad kontrolnih jedinica i slično.
2. Nivo zasebne jedinice (engl. *Unit*) – Prilikom formiranja verifikacionog plana, tim selektuje kompleksne module na kojima je potrebno sprovesti zasebnu verifikaciju i dodeljuje ih najčešće jednom verifikacionom inženjeru. Ovo su uglavnom kompleksni moduli unutar jezgra koji su često i univerzalni te postoji mogućnost njihovog ponovnog korišćenja. Zbog navedenog, od velikog interesa je da budu dobro testirani.
3. Nivo jezgra (engl. *Core*) – Ovaj nivo predstavlja testiranje na nivou zasebnih celina, koji kao takvi, imaju jako veliku univerzalnost. Takođe, jezgra najčešće bivaju instancirana nekoliko puta na istom čipu ili čak i na nekoliko potpuno različitih čipova. Sve ovo govori o značaju verifikacije na ovom nivou, jer potencijalno može kompaniji značajno uštedeti resurse u budućnost. Pored funkcionalnosti, akcenat je i na integraciji prethodno verifikovanih *Unit*-a.
4. Nivo čipa – Testiranje na nivou čipa se najčešće odvija u laboratorijama gde posebno obučeni inženjeri sprovode verifikaciju pomoću posebno odabranog skupa testova. Ovaj vid testiranja je najsporiji te je potrebno pažljivo odabrati testove koji aktiviraju, što veći deo čipa, to jest, pokrivaju što veći deo njegove funkcionalnosti. Pored funkcionalnog testiranja, u laboratoriji se sprovodi i niz generičkih testova koji utvrđuju da li je proizvodni proces izveden korektno. Drugim rečima, ovi testovi otkrivaju fizička oštećenja na čipovima nastala u procesu proizvodnje.

5. Nivo sistema – Po integraciji čipa u sistem, vrši se dodatno testiranje koje proverava interakciju razvijenog čipa sa ostalim čipovima na razvojnoj ili konačnoj štampanoj ploči.
6. Hardver/softver koverifikacija – Ukoliko je hardver takav da na sebi ima i procesore opšte namene, vrlo je verovatno da će se na njima izvršavati neki program. Softver najčešće upravlja okružujućim hardverskim komponentama te je potrebno proveriti da li program na pravilan način upošljava ili očitava rezultate rada čipova na sistemu. Najčešći problemi koji se u ovoj fazi otkrivaju su nepravilnosti u okviru softvera, ali i do sada neotkrivene greške unutar hardvera koje najčešće imaju veze sa vremenskom komponentom slanja/primanja komandi i statusa. Interesantno za ovaj nivo testiranja je da se velika većina propuštenih grešaka najčešće zaobilazi tako što se krajnji softver modifikuje tako da neispravne funkcionalnosti aktivira na način kako se greške ne bi ispoljili kod krajnjeg korisnika. Ovo se uglavnom radi prilikom pisanja softvera na niskom nivou poznatog kao drajver (engl. *driver*).

Verifikacija Argus akceleratora nije izvršena na svim navedenim nivoima, posebno ne za svaku komponentu. Ipak, da bi se proces ubrzao, postojala je podela na kompleksne jedinice koje su imale zasebna verifikaciona okruženja.

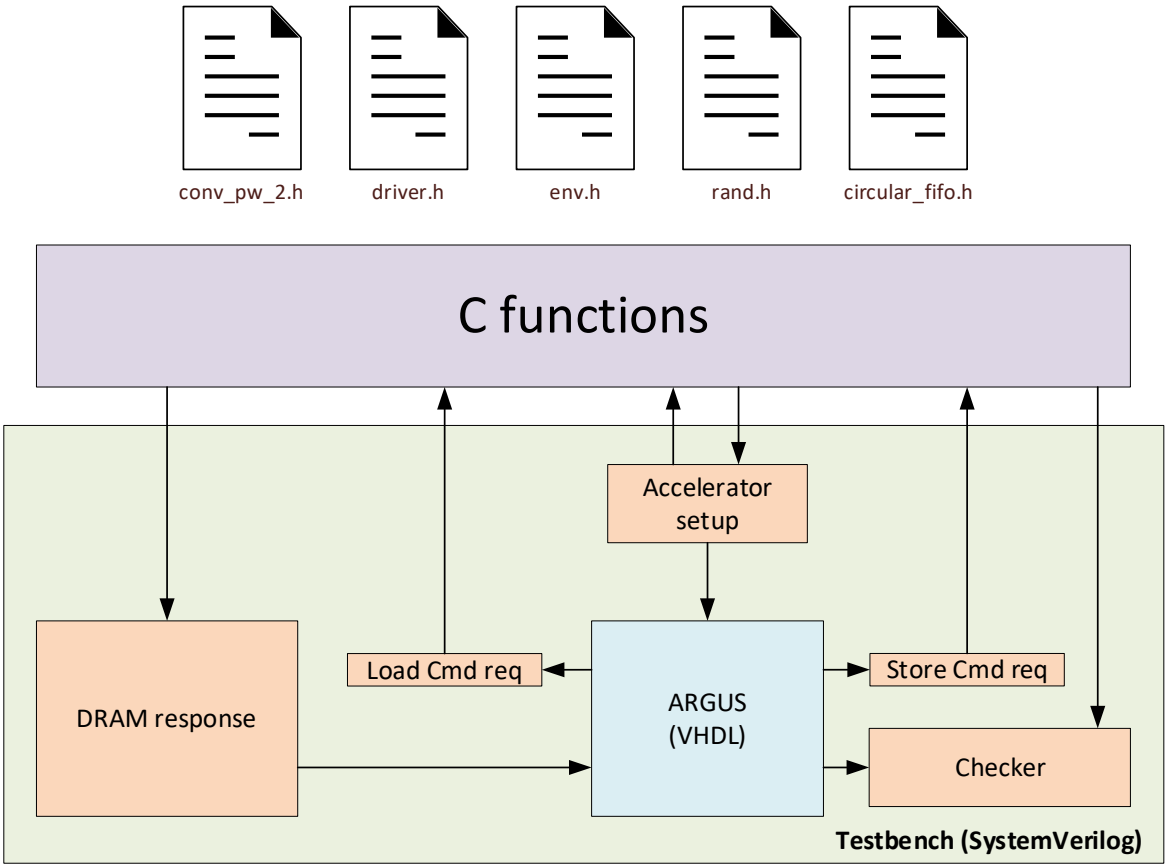
Kao rezime uvodnog dela funkcionalne verifikacije, možemo reći da je testiranje na najvišem nivou, nivo jezgra, izvršeno koristeći pristup zlatnih vektora. Matrice su ekstrahovane iz javno dostupnih modela CNN-ova pisanih u Python jeziku. Zbog prirode CNN-ova, ali i implementacije akceleratora, testiranje je moguće izvršiti sloj po sloj bez gubitka na kvalitetu verifikacije. Dakle, iz dostupnih modela CNN-a su izdvojeni ulazni i izlazni podaci jednog sloja. Ulazni podaci se dovode na ulaz akceleratora, a rezultati izračunavanja se porede sa odgovarajućom ekstrahovanom matricom. Ovakav pristup je moguć jer je konfiguracija CNN-a unapred poznata i ne zavisi od rezultata koji se dobijaju na izlazu slojeva mreže. U nastavku poglavlja je prikazan tehnički deo testiranja sa osvrtima na uvedene teorijske osnove funkcionalne verifikacije.

7.1.1 Verifikaciono okruženje

Verifikaciono okruženje će biti opisano u *top-down* maniru, idući od okruženja za testiranje na *Core* nivou ka kompleksnim okruženjem na nivou *Unit*-a. Na slici 56 je prikazano

verifikaciono okruženje na najvišem nivou koje uključuje kompletan Argus akcelerator kao DUT sa izuzetkom *Data Mover*-a (DM) i konfiguracione registarske banka. Zbog prethodnog poznavanja prirode DM-a i kako bi se izbeglo integrisanje Xilinx-ovog IP-a unutar okruženja, kreirani su pomoćni procesi (engl. *task*) koji simuliraju rad DM-a. Sa druge strane, konfiguraciona registarska banka je izostavljena jer je ona generisana pomoću Xilinx-ovog alata koji kreira registarske banke proizvoljne veličine, a kojima se pristupa pomoću AXI-lite interfejsa. Ukoliko bi ova komponenta bila deo okruženja, bilo bi neophodno integrisati gotove komponente koje bi stimulisale banku na pravi način (AXI-lite protokol) ili još komplikovanije, napisati ovakvu komponentu. Mana ovakvog pristupa je što će deo hardvera (registarska banka i njena veza sa Argus akceleratorom) ostati ne testiran pre verifikacije na stvarnom hardveru. Ipak, rizici su izuzetno mali zato što se kod u potpunosti generiše od strane alata, a integracija je izuzetno jednostavna i svodi se na povezivanje signala. Takođe, akcelerator je tako koncipiran da je u toku procesiranja određenog sloja, konfiguracija stacionarna. Drugim rečima, nema promene vrednosti konfiguracionih registara, što se u literaturi može pronaći pod nazivom *on-the-fly* promena konfiguracije. Sve druge komponente Argus arhitekture su sadržane u plavom kvadratu na slici verifikacionog okruženja (slika 56).

Opis komponenti okruženja ćemo početi od narandžastih blokova napisanih u SystemVerilog-u. Jedini *task testbench*-a koji inicira transakcije je nazvan *Accelerator setup*. Odmah po pokretanju simulacije, ovaj *task* postavlja reset signal na aktivno čime se Argus CNN akcelerator dovodi u inicijalno stanje. Dok je reset aktivan, iz C-a se preuzimaju konfiguracioni parametri i dovode na ulaz akceleratora. Po stabilizaciji konfiguracionih ulaza, reset se otpušta i pokreću se ostali *task*-ovi unutar *testbench*-a, ali u pasivnom režimu. Ovo znači da ostali procesi nadgledaju interfejse DUT-a čekajući komande.



Slika 56 Verifikaciono okruženje na najvišem nivou (Core level).

Zamislimo da se trenutno simulira konvolucioni sloj i da akcelerator započinje procesiranje. Prvo će IS zatražiti parametre sloja (bias, nzi, nzv) preko *task*-a nazvanog *Load Cmd req*. Ovaj *task* preuzima zahteve koji dolaze u obliku opisane komande DM-a (početna adresa u DRAM-u i količina podataka koji se potražuju). Kako bi se na što verniji način simuliralo izvršavanje na stvarnom hardveru, ovaj zahtev se prosleđuje FIFO baferu koji je implementiran u C delu okruženja. Bafer je programabilne dubine koja je konfigurisana na 5 zahteva. Drugim rečima, to znači da će u prvih pet taktova Argus arhitektura dobiti obaveštenje da je uspešno prihvaćeno prvih pet zahteva, a da pri tom niti jedan podatak neće biti dostavljen akceleratoru. *DRAM response task* je takođe aktivan i proverava da li u FIFO baferu ima pristiglih zahteva. Pošto prihvati prvi zahtev, *DRAM response* učitava parametre koji su skladišteni u nizovima unutar C-a i prosleđuje ih na ulaz Argus akceleratora. Opet, da bi verifikacija bila što približnija realnoj situaciji, zahtevi se obrađuju sa pauzama čije je trajanje određeno generisanim, pseudo-slučajanim brojem. Na ovaj način se najviše proverava nezavisnost funkcionisanja hardvera od vremenske komponente. Pored pauzi između obrada zahteva, ni isporuka zahtevanih podataka nije uvek kontinualna već postoje pauze. Na ovaj

način se proverava ispravnost toka podataka i njegova otpornost na promenljive performanse DRAM kontrolera. Naravno, prilikom provere postojanja uskih grla i performansi arhitekture, navedene pauze su konfigurisane na vrednost 0, kako bi se maksimalno opteretio DUT. Sve navedene situacije su detaljno opisane na slikama 58 i 59.

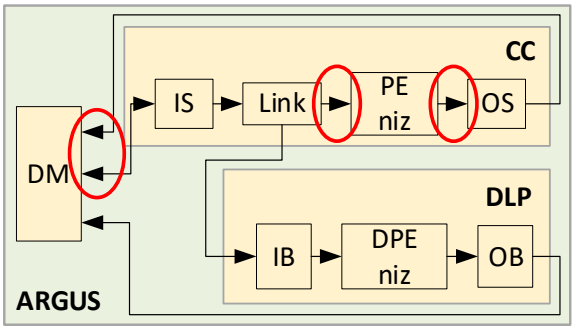
Pošto DM opslužuje i izlaznu stranu DUT-a, *Store Cmd req* se ponaša na isti način kao i *Load Cmd req task*. Takođe, u C delu se nalazi FIFO bafer koji se isčitava od strane *Checker task*-a. Pored komandi iz FIFO bafera, *Checker task* uzima i podatke iz izlazne matrice sloja kako bi izvršio poređenje onoga što generiše DUT sa očekivanim rezultatima. Primetimo da se FIFO bafer koristi na više mesta u okviru C-a te je on napisan kao univerzalan bafer u okviru jedne od C biblioteka koje se koriste u okviru disertacije. Pored FIFO bafera, kreirana je i zasebna biblioteka koja služi za generisanje pseudo-slučajnih brojeva koji se koriste prilikom testiranja otpornosti DUT-a na nezavisnost od vremenske komponente.

Iako nije prikazan, deo verifikacionog procesa je i program napisan u Python jeziku koji generiše neke od .h fajlova simulacionog okruženja. Pre nego se pokrene simulacija potrebno je pozvati ovaj program koji učitava CNN model, čiji sloj će se simulirati, i na nekoj od slika iz ImageNet-a pokreće procesiranje ekstrahujući ulaznu i izlaznu matricu odgovarajućeg sloja. Pored IFM-a i OFM-a preuzimaju se i parametri sloja, kerneli i konfiguracija sloja. Potom program generiše .h fajlove u koje smešta parametre zajedno sa IFM-om i OFM-om. Konfiguracioni parametri su smešteni u okviru strukture, dok su ostali parametri i matrice smešteni u nizove. Konfiguracioni deo, pored karakteristika sloja (visina/širina kernela, IFM-a...) sadrži i početne adrese ovih nizova u memoriji. Adrese će biti dodeljene u toku kompajliranja te će se zajedno sa ostalim parametrima proslediti DUT-u u okviru *Accelerator setup task*-a. Primetimo da će se isto dešavati i prilikom simulacije na stvarnom hardveru, što znači da je moguće iskoristiti identičan C kod.

Dakle, pored pomenutih biblioteka, C deo verifikacionog okruženja inkorporira .h fajlove koji sadrže parametre sloja i IFM/OFM matrice. Takođe, postoje i funkcije koje pristupaju nizovima u okviru .h fajlova na zahtev SystemVerilog *task*-ova kao i pomoćne funkcije koje služe za instancioniranje i jednostavnu manipulaciju FIFO baferima. Neizostavan deo na kraju simulacije su i funkcije koje oslobađaju zauzete delove memorije kako ne bi došlo do poznatog curenja memorije koje može imati značajne posledice u slučaju regresija sa većim brojem testova.

7.1.1.1 Top-level verifikacija

Za simulacione primere na najvišem nivou hijerarhije je korišćen drugi *Pointwise* konvolucioni sloj MobileNet v1 CNN-a. Specifičnost *Pointwise* konvolucionih slojeva je da su im kerneli dimenzija 1x1xIFM_D. Ova karakteristika u mnogome olakšava razumevanje talasnih oblika signala zato što nema ponovnog korišćenja delova IFM-a između računanja uzastopnih tačaka OFM-a, to jest, nema potrebe za keširanjem IFM-a unutar IS modula. To dalje implicira da se kompletan IS zaobilazi. Na slikama koje slede su prikazani tokovi podataka između većih blokova konvolucionog jezgra, naznačeni na slici 57.



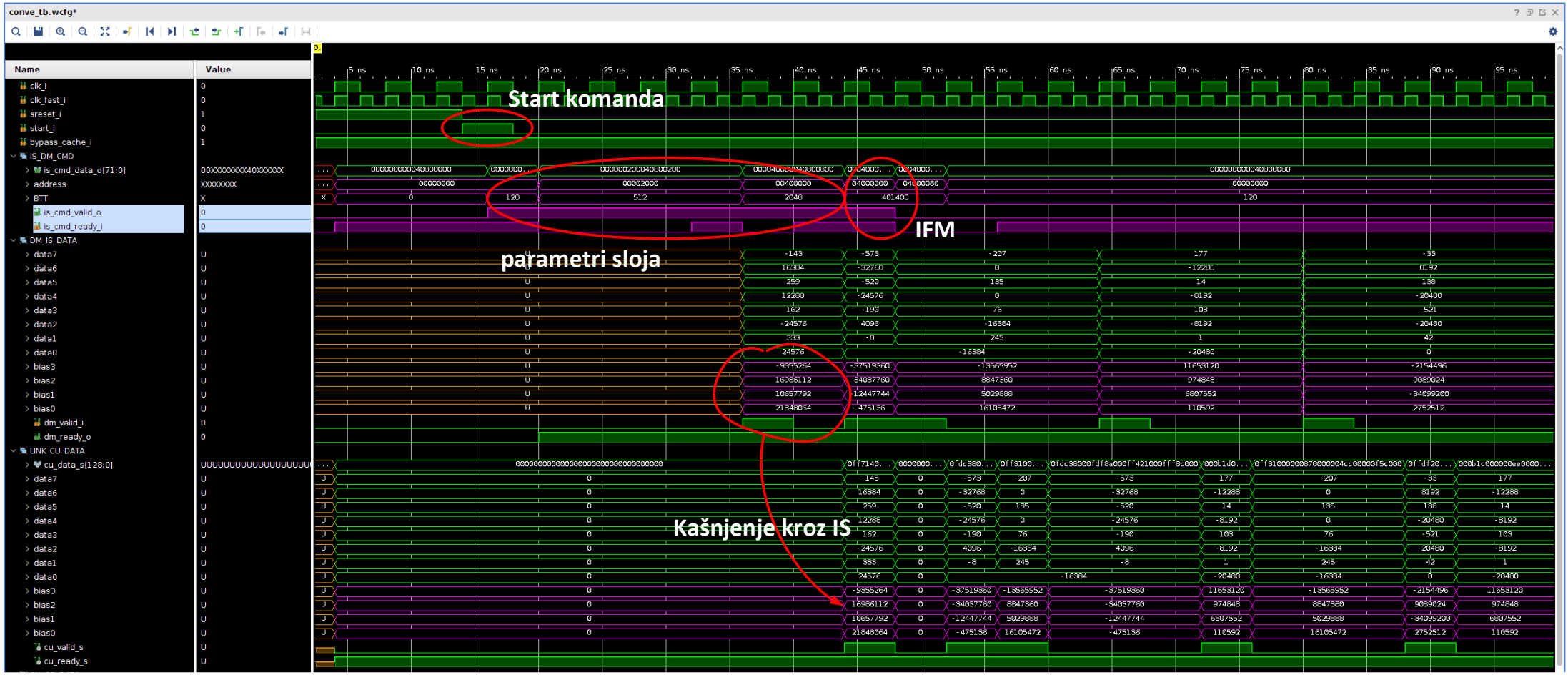
Slika 57 Tačke opservacije signala u slučaju drugog Pointwise sloja MobileNet v1 CNN-a.

Na slici 58 su akcentovani talasni oblici signala koje generiše IS na početku procesiranja sloja. Bez obzira što se keširanje ne koristi, IS je blok koji generiše sve zahteve ka DRAM-u te je značajan deo bloka aktivan.

Prva dva signala na slici 58 su takt signali, jedan frekvencije 250 MHz i drugi 500 MHz, a zatim sledi sinhroni reset. Po otpuštanju reseta, DUT dobija start komandu koja pokreće procesiranje. U okviru grupe IS_DM_CMD signala vidimo AXI-Stream interfejs koji šalje zahteve ka DRAM-u. U slučaju konvolucionih slojeva, to su parametri sloja podeljeni u tri transakcije i IFM zahtev/i. Pošto je keširanje suvišno ono se zaobilazi unutar IS-a, a IFM se potražuje samo jednim zahtevom. Drugim rečima, umesto da se IFM transferuje štapić po štapić, u slučaju *pointwise* konvolucija se kontinualno doprema kompletan IFM. Na ovaj način se maksimizuje iskorišćenje propusne moći DRAM kontrolera jer bi potraživanje štapić po štapić bilo neefikasno posebno kod slojeva čija je dubina IFM-a plitka. Ukoliko analiziramo vremensku komponentu na ovom interfejsu, primetićemo da je Argus akcelerator spreman da u svakom taktu isporuči novu komandu što se vidi na osnovu konstantno aktivnog *valid*

signala. Kao što je rečeno u uvodnom delu, bafer koji prihvata komande to radi sa pauzama čije je trajanje randomizovano. Ovo se reflektuje u obliku *ready* signala koji nije stalno aktivan.

Druga dva interesantna interfejsa na slici 58 su DM_IS_DATA i LINK_CU_DATA. Oni predstavljaju tok podataka od DRAM-a ka akceleratoru. Prvi interfejs je direktna veza sa DRAM-om, dok se drugi nalazi između Link modula i niza PE-ova. DM_IS_DATA predstavlja odgovor DRAM kontrolera na prethodno opisane zahteve. Kao i u slučaju komandnog interfejsa i ovaj poštuje AXI-Stream protokol. Primetimo da se prvo prenose *bias*-i i da postoji određeno kašnjenje kroz IS od dva spora takt perioda što je posebno naznačeno na slici 58. Ovo je posledica registara unutar IS modula koji su aktivni iako se keširanje zaobilazi. Iako nisu potrebni u funkcionalnom smislu, potreba za ovim registarima proizilazi iz relativno visoke frekvencije na kojoj akcelerator radi te je potrebno *pipeline*-ovati i jednostavnije delove kombinacione logike. Kao i u slučaju komandnog interfejsa, i ovde se podaci dopremaju sa pauzama između svake transakcije što se može primetiti na *valid* signalu koji je povremeno neaktivan. Sa druge strane, akcelerator je uvek spreman da prihvati nove podatke te je *ready* signal uvek aktivan. Napomenimo i to da je zarad jednostavnosti *tlast* signal AXI-Stream protokola izostavljen jer su njegove promene spore te nisu vidljive u prikazanom vremenskom intervalu simulacije.



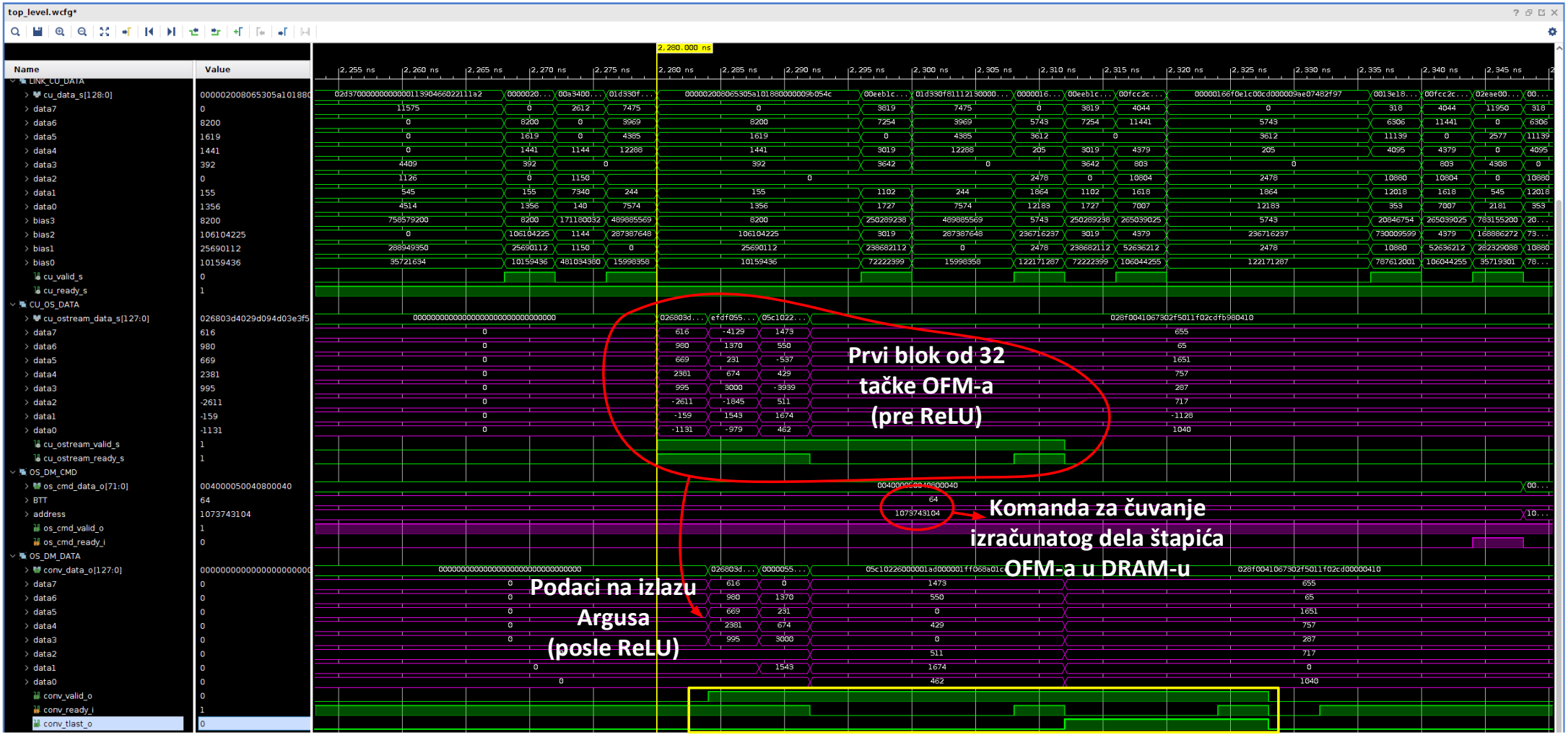
Slika 58 Početak procesiranja drugog Pointwise sloja MobileNet v1 mreže.

Slika 59 prikazuje izlaznu stranu Argus arhitekture sa akcentom na tri interfejsa: OS_DM_CMD, CU_OS_DATA i OS_DM_DATA. Prvo, komandni interfejs šalje DRAM-u zahteve za smeštanje podataka koji se izračunavaju. Veličina transfera podataka je 64 bajta dok je adresa definisana početnom adresom alociranog bafera unutar C verifikacionog okruženja i pozicijom trenutnog štapića unutar OFM-a. Veličina svakog dela štapića koji se skladišti u jednom dahu je definisana brojem tačaka OFM-a (jednak broju aktivnih PE-ova) i veličinom rezultata koji je uvek 2 bajta (16-bitna aritmetika). Otuda i veličina zahteva od 64 bajta, 32 PE-a po 2 bajta podatak.

CU_OS_DATA interfejs tok podataka od niza PE-ova ka OS modulu. Karakteristično za ovaj deo akceleratora je da se rezultati uvek prosleđuju u nekom vidu *burst* moda. Drugim rečima, sve tačke OFM-a koje se trenutno računaju su spremne istovremeno. Ovo je posledica paralelnog rada PE-ova koji su ravnomerno opterećeni zbog prirode algoritma za orezivanje. Prvi izračunati štapić za *pointwise* sloja od četiri bloka sa po osam rezultata je prikazan na slici 59. Jasno se može videti da ovi podaci posle prolaska kroz OS bivaju izmenjeni u smislu da je na njih primenjena ReLU aktivaciona funkcija te su negativni rezultati dobili vrednost nula. Finalni podaci su prikazani u okviru OS_DM_DATA interfejsa.

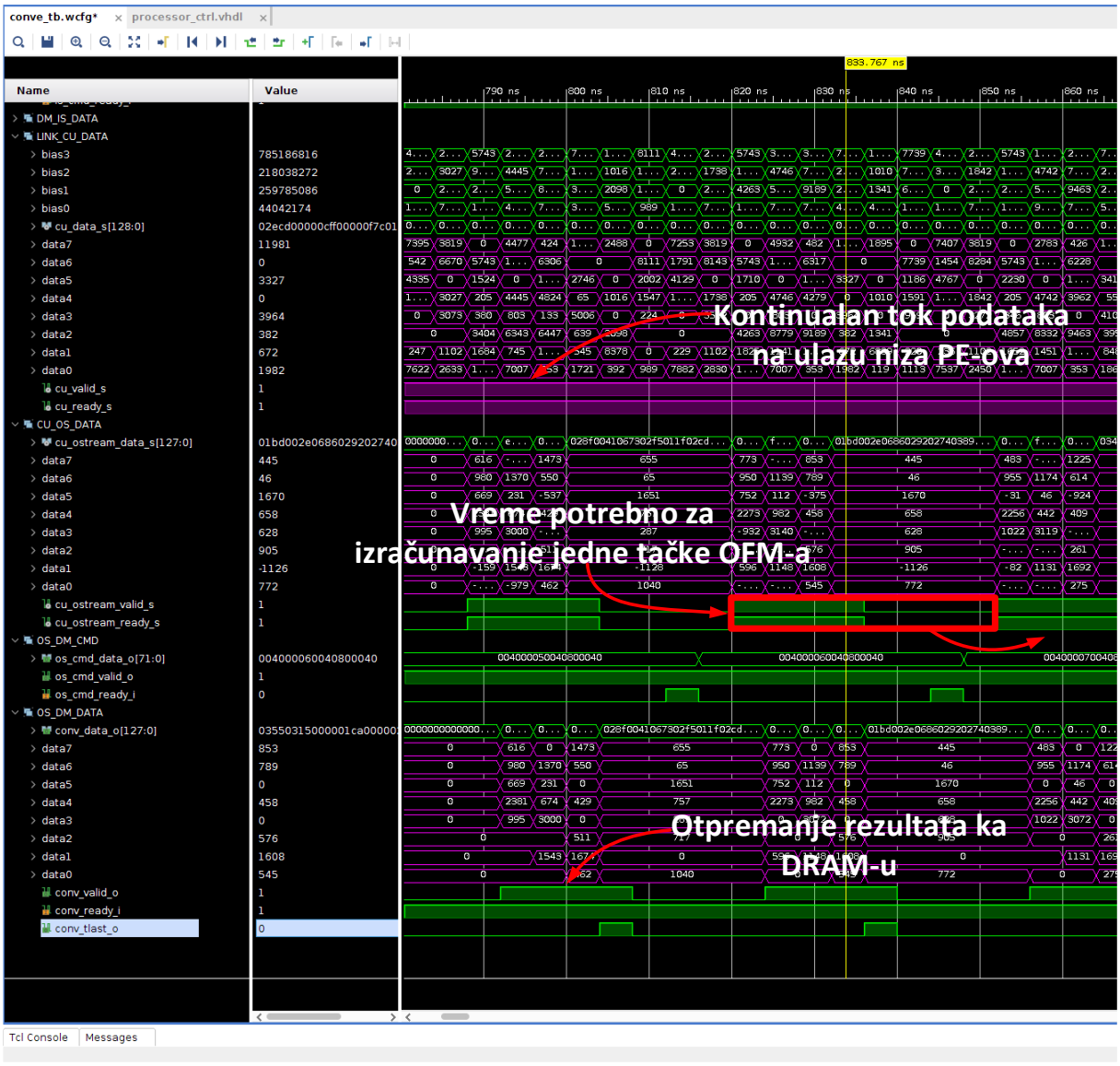
Pored podataka, na slici 59 se može analizirati i vremenska komponenta procesiranja konvolucionog sloja pomoću Argus akceleratora. Prvi interfejs od gore (potpuno zeleni signali) doprema podatke IFM-a ka nizu PE-ova, i njegov tok iz navedenih razloga nije kontinulan kako bi se testirala ispravnost DUT-a. Takođe, ni spremnost DRAM-a da prihvati podatke nije kontinualna pa se tako podaci na OS_DM_DATA interfejsu preuzimaju sa pauzama. Ovo je označeno žutim pravougaonikom koji akcentuje *valid/ready* signale AXI-Stream protokola ka DM-u. Prvi put do sada, ovde se može videti i aktivnost *tlast* signala u okviru AXI-Stream interfejsa (*conv_tlast_o* signal). *tlast* signal je aktivan na poslednjoj grupi od osam podataka svakog zahteva. Ovakva aktivnost *tlast* signala je obavezna zbog interne implementacije DM-a i DRAM kontrolera. Odsustvo pravilnog rada *tlast* signala će uzrokovati zastoj DM-a i DRAM-a što dalje ima za posledicu zaustavljanje kompletnog akceleratora. Ovo je izražen problem naročito u slučajevima kada se između slojeva ne resetuje kompletan akcelerator.

Da bi se ispitala efikasnost akceleratora u idealnim slučajevima potrebno je idealizovati ponašanje DRAM kontrolera što je ilustrovano slikom 60.



Slika 59 Izlazni podaci iz niza PE-ova i OS modula, pre i posle aktivacione funkcije.

Kako bi se maksimalno opteretio akcelerator, ulazna magistrala u niz PE-ova je konstantno aktivna (*valid* je 1). Sa druge strane, da izlazna magistrala ne bi usporavala akcelerator, *conv_ready_i* signal (OS_DM_DATA interfejs) je takođe konstantno aktivan što znači da je DRAM spreman da preuzme rezultate. Ukoliko su protoci na magistralama upareni, to jest, nema uskih grla unutar akceleratora, očekivano je da je ulaz u niz PE-ova spreman da prihvati novi blok od osam podataka u svakom taktu. Ovo jeste slučaj i to se oslikava u konstantno aktivnom *cu_ready_s* signalu u okviru LINK_CU_DATA interfejsa. Takođe, vreme trajanja računanja jedne tačke OFM-a je osam taktova što je očekivano ako se uzme u obzir veličina kernela koja je 1x1xIFM_D, a IFM_D je jednak 64 tačke, odnosno osam grupa od po osam, a pri tom znamo da je kapacitet PE-a takav da može da obradi jednu grupu u jednom taktu.



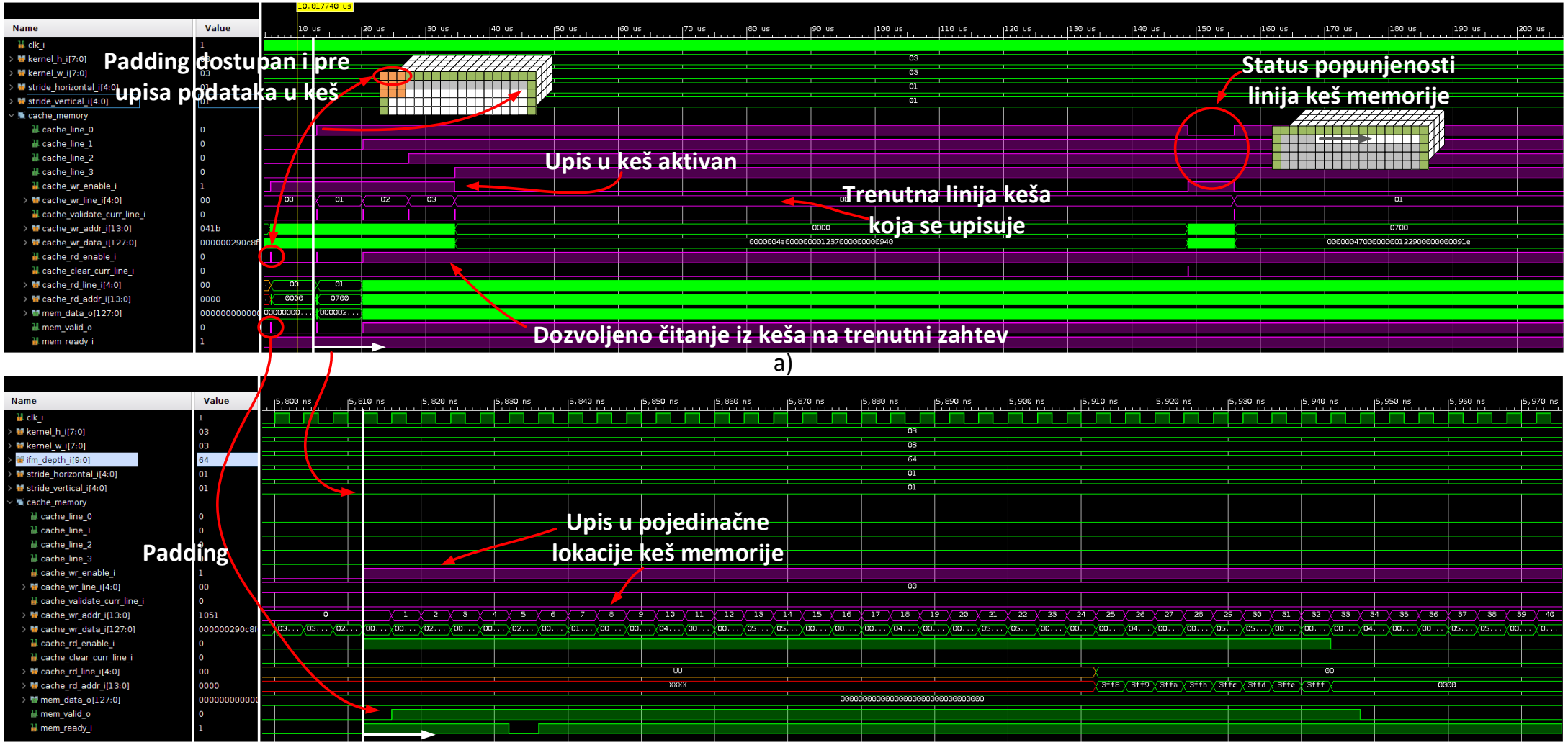
Slika 60 Provera efikasnoti i postojanja uskih grla akceleratora u slučajevima idealnog DRAM kontrolera.

Navedeno je potvrđeno na visokom nivou interfejsom CU_OS_DATA na slici 60. Crveni pravougaonik predstavlja vreme u kome se računaju naredne tačke OFM-a, odnosno periodu potrebnu da se izračuna jedna tačka OFM-a. Detaljniji prikaz računanja konvolucija i performansi PE-ova će biti prikazan u okviru simulacionih rezultata niza PE-ova koji slede.

7.1.1.2 Ulazni tok podataka (IS)

Iako IS nije bio testiran na *Unit* nivou, u nastavku su prikazani karakteristični talasni oblici koji oslikavaju rad keš memorije IS modula. Kako bi upotreba keša imala smisla, korišćen je drugi konvolucioni sloj VGG-16 mreže kod koga imamo preklapanja segmenata IFM-a između susednih tačaka OFM-a. Ovaj sloj karakterišu kerneli dimenzija 3x3x64, IFM dimenzija 224x224x64 i korak po visini i širini koji su jednaki i iznose 1. Konfiguracija se može videti i na slikama 61 i 62 u okviru prvih 5 signala. Na slici 61 su prikazani talasni oblici prilikom upisa podataka u keš, tačnije, na samom početku procesiranja navedenog sloja. Idući od vrha možemo primetiti četiri signala nazvana *cache_line_0/1/2/3* koji predstavlja indikaciju popunjenosti svake linije keš memorije. Podsetimo se da svaka linija keš memorije odgovara jednoj horizontalnoj liniji IFM-a. To znači da ukoliko keš ima četiri linije, u njega možemo smestiti četiri linije IFM-a. Takođe, broj linija keš memorije zavisi od visine kernela i vertikalnog koraka konvolucije i računa se kao njihov zbir, što je objašnjeno u sekciji koja detaljno opisuje arhitekturu IS modula.

Idući sa leva na desno, uspravna bela linija označava trenutak kada je upis u prvu liniju keš memorije završen, što se indikuje postavljenjem signala *cache_line_0* na 1. Isti trenutak je označen i na umanjenoj, hipotetičkoj slici keš memorije gde je sivom bojom označena zauzetost lokacija keš memorije (zelena boja označava *padding*). Kako vreme odmiče, može se primetiti da se popunjavaju i preostale tri linije. Po završetku popunjavanja kompletne keš memorije, *cache_wr_enable_i* signal se spušta na 0 što se dalje signalizira DRAM-u koji pauzira dopremanje podataka. Pored popunjavanja memorije, na slici keš memorije je narandžastom bojom istaknut kvadrat veličine 3x3xIFM_D koji se procesira prilikom računanja gornje leve tačke OFM-a, to jest, kvadrat IFM-a koji se prvi procesira. Ono što se može primetiti je da su horizontalna tri štapića ovog kvadrata u *padding*-u (zeleni kvadratići oko keš memorije) te su oni dostupni i pre upisa prve linije.



Slika 61 Ilustracija popunjavanja linija keš memorije (a) i detaljan prikaz upisa u prvu liniju keša (b).

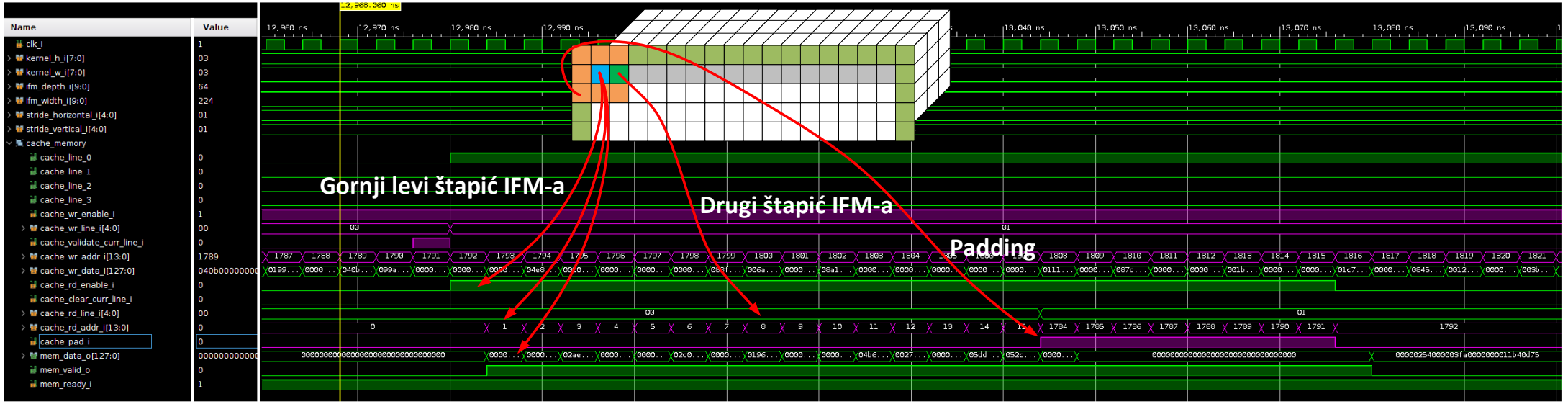
Padding delovi se odmah prosleđuju nizu PE-ova što je vidljivo na *mem_valid_o* signalu na početku simulacije (zaokružen crvenim, slika 61a). Još jedan interesantan trenutak je postavljanje *cache_rd_enable_i* signala na jedan. U ovom momentu je i druga linija keš memorije popunjena, što zajedno sa padding-om na vrhu IFM-a čini ukupno tri linije spremne za procesiranje. Dalje, IS nastavlja upis u treću liniju keša, ali to ne ometa dalje čitanje jer su svi podaci za računanje prvog reda OFM-a dostupni te *cache_rd_enable_i* ostaje aktivan do daljnjeg. Vrlo brzo nakon ovog momenta se deaktivira signal *cache_wr_enable_i* što znači da je keš memorija popunjena. Idući na desno na slici 61a), možemo primetiti zaokruženi trenutak u kome se oslobađa prva linija keša (crvena elipsa) i ponovo postavlja *cache_wr_enable_i* na jedan. Ovo znači da je prva linija IFM-a u potpunosti iskorišćena i da na njeno mesto može da se upiše sledeća, u ovom slučaju peta linija IFM-a. Pošto keš ima visinu od četiri linije IFM-a, to znači da po oslobađanju prve preostaju još tri, što je dovoljno za računanje narednog reda OFM-a. Da nema zastoja u računanju izlaza se može videti po tome što u ovom periodu signal *cache_rd_enable_i* ostaje nepromenjen, a *mem_valid_o* ostaje i dalje aktivan (niz PE-ova nastavlja preuzimanje podataka).

Slika 61b) predstavlja uvećani deo početka upisa u keš prikazanog na 61a). Uspravna bela linija označava početak upisa u prvu liniju što je praćeno postavljanjem signala *cache_wr_enable_i* na jedan u ovom trenutku. Od interesantnih signala može se uočiti aktiviranje *cache_wr_addr_i* koji oslikava kontinualni upis u keš memoriju inkrementujući adrese za jedan. Napomenimo i to da se podaci smeštaju u 128-bitnim blokovima što znači da svaka memorijska lokacija ima ovu veličinu. Zbog 16-bitne aritmetike, navedeno znači da u svaku lokaciju staje po osam tačaka IFM-a. Još jednom, na 61b) se jasnije vidi prosleđivanje *padding*-a zaokruženog na slici 61a). Signal *mem_valid_o* se postavlja na 1, *mem_ready_i* je takođe aktivan što znači da niz PE-ova preuzima podatke sa *mem_data_o* magistrale. Na magistrali podataka svaka tačka ima vrednost 0 i tako narednih 24 takta. Ova 24 takta odgovaraju prenosu 3x8x8 podataka, to jest, 3 gornja štapića u *padding*-u, gde svaki od njih ima dubinu koja odgovara IFM_D (8x8=64).

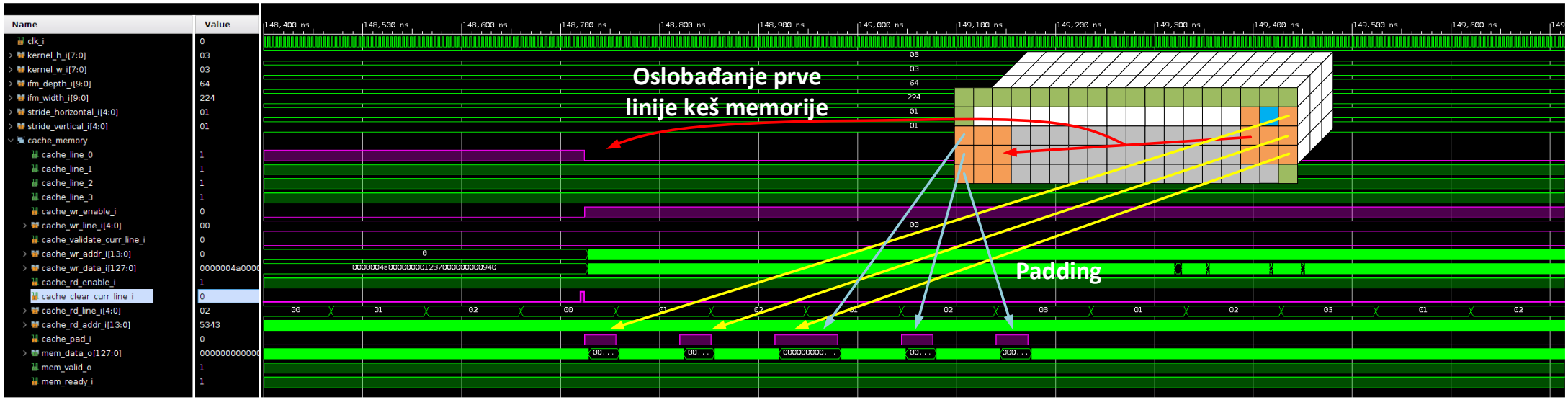
Slika 62 detaljnije prikazuje proces čitanja iz keš memorije a) i oslobađanje linija keša (b). Takođe, detaljno je prikazan deo aktiviranja *padding*-a prilikom vertikanog koraka konvolucije. Na slici 62a) se može primetiti popunjavanje prve linije keš memorije postavljanjem signala *cache_line_0* na 1. Ovo je izazvano upisom poslednjeg podatka prve

linije IFM-a na adresu 1791 (*cache_wr_addr_i*). Primetimo da se adresa poklapa sa širinom i dubinom IFM-a, jer je širina 224 štapića, a svaki štapić zauzima 8 lokacija u memoriji (8x224=1792). Upis se nesmetano nastavlja dalje preko adrese 1792 dok čitanje počinje od adrese 0 (*cache_rd_addr_i*). Naravno, isčitavanje je ispraćeno aktivacijom *mem_valid_o* signala. Da bismo potvrdili funkcionalnu korektnost, detaljno je strelicama obeleženo čitanje štapića iz prve linije keš memorije. Prvi štapić, označen plavom bojom, se nalazi na adresama od 0 do 7. Po završetku prvog, drugi štapić (zeleni kvadrat) kreće od adrese 8 do 15 što se vidi na adresnom signalu (*cache_rd_addr_i*). Po prosleđivanju ova dva štapića nizu PE-ova, na red dolazi *padding* koji se nalazi u trećem redu kvadrata 3x3 IFM-a koji se trenutno procesira. Prelazak na slanje *padding*-a se kontroliše signalom *cache_pad_i* koji se posle adrese 15 postavlja na 1. Narednih 8 taktova se nizu PE-ova prosleđuje *padding,* te se vrednost 0 postavlja na *mem_data_o* magistralu. Po završetku *padding*-a, procesiranje se zaustavlja jer druga linija keš memorije nije još uvek popunjena. Napomenimo da se ovakve pauze dešavaju samo na početku obrade sloja. Drugim rečima, dešavaju se samo prilikom inicijalnog popunjavanja keša. Naravno, moguće je i u nekim ekstremnim slučajevima gde računanje traje jako kratko, a korak konvolucije je veliki pa DRAM ne stigne da dopremi novi red IFM-a dok se prethodni procesira. Ovakvi slučajevi su izuzetno retki u slučaju današnjih CNN-ova.

U toku verifikacije, najzahtevniji slučaj provere funkcionalne korektnosti IS modula se odnosio na trenutak kada se završava računanje tekućeg reda OFM-a i prelaz na sledeći OFM red. U ovom trenutku je potrebno osloboditi liniju/e keš memorije u zavisnosti od vertikalnog koraka i podesiti adrese upisa/čitanja na odgovarajuće vrednosti. Ovaj period je prikazan na slici 62b). Na slici možemo primetiti da ljubičasti signal, *cache_line_0,* dobija vrednost 0 što je indikacija da je prva linija keš memorije slobodna za upis. Istovremeno počinje upis što se oslikava u talasnom obliku signala *cache_wr_enable_i*. Na ovaj način se daje maksimalno moguće vreme IS modulu da dopremi sledeći red IFM-a. Posle plavog štapića sledi *padding*, čije je vreme izvršavanja vidljivo postavljanjem signala *cache_pad_i* na 1. Kako bi se ispratio kompletan *padding*, moguće je pratiti žute strelice koje pokazuju na vremenske intervale u kojima se nule prosleđuju nizu PE-ova. Da sve funkcionise kako je očekivano se može primetiti u trenutku prelaska iz prvog u drugi red kada je trajanje *padding*-a dvostruko duže što je očekivano jer imamo dva *padding* štapića, jedan za drugim.



a)



b)

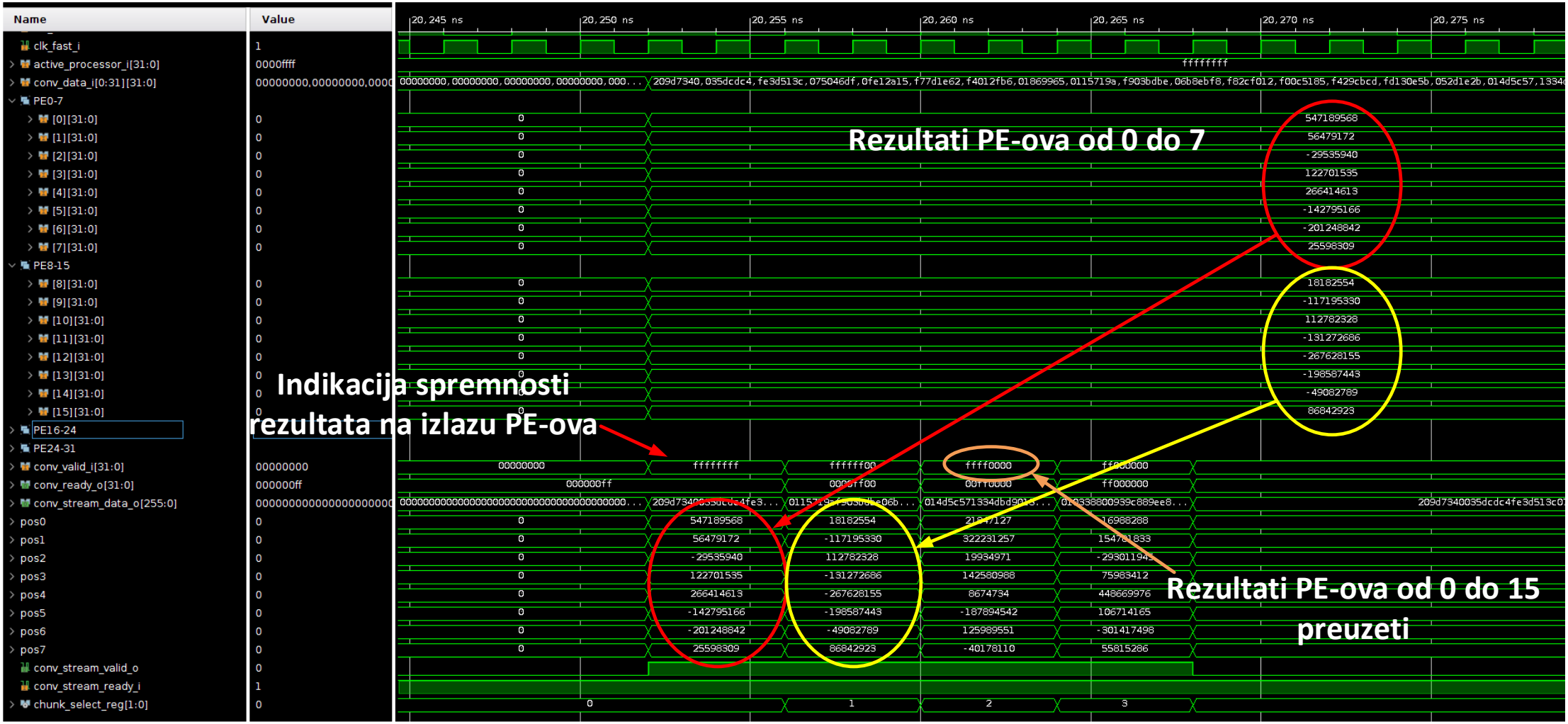
Slika 62 Čitanje iz keš memorije prilikom izračunavanja prve tačke OFM-a (a) i detaljan prikaz oslobađanja linije keš memorije(b).

U periodu u kome je *cache_pad_i* signal aktivan, a na njega pokazuju i žuta i plava strelica, dešava se prelaz iz jednog u drugi red. Pored funkcionalne korektnosti bitno je maksimizovati iskorišćenje raspoloživih performansi niza PE-ova. Visoka efikasnost se može postići ukoliko IS dostavlja podatke bez pauza. Da je ovo slučaj jasno se vidi na signalu *mem_valid_o* koji je na 62b) konstantno postavljen na 1. Da se svaki takt dešava trasfer jasno je jer je i *mem_ready_i* signal takođe aktivan. Primetimo i da ne postoji prekid u protoku podataka ni prilikom slanja *padding* dela (signal *cache_pad_i* aktivan). Ovo je postignuto pažljivim projektovanjem kontrolne logike IS modula.

7.1.1.3 Niz procesorskih elemenata

Svakako, najkompleksniji deo niza PE-ova je PE koji ima centralnu ulogu u bloku. Zbog dvostruko veće frekvencije od okruženja, na kojoj rade DSP blokovi, bilo je izuzetno važno implementirati kontrolnu logiku na način da se ne naruše pravila dobrog dizajna, ali i da se maksimizuje iskorišćenje raspoloživih potencijala/hardverskih resursa. Iako je funkcionalna verifikacija u velikoj većini slučajeva prvi vid verifikacije sa kojim se sreće RTL, u slučaju PE-a to nije bilo tako. Kako bi se proverile mogućnosti platformi, prvo je napisan jednostavan RTL koji koristi DSP blok na navedeni način te je kod testiran na FPGA sistemu. Tek kada se ustanovilo da je stvarno moguće izvesti ovako nešto, započet je proces dalje gradnje arhitekture. Pored testiranja mogućnosti FPGA platformi, nekoliko RTL implementacija je pokušano kako bi se kompajler naveo da koristi tačno željene elemente DSP blokova, kako je opisano u poglavlju 6.5 koji se bavi prezentovanjem krajnje arhitekture. Iako je funkcionalnoj verifikaciji prethodilo testiranje na stvarnom hardveru, jednostavni PE koji je bio korišćen za ovo testiranje nema mnogo sličnosti sa onim koji se trenutno koristi u okviru Argus arhitekture te su rezultati rada probnog hardvera ovde izostavljeni.

Pored PE-a, u nastavku su prikazani talasni oblici i kolektora rezultata. Podsećanja radi, kolektor preuzima rezultate izračunavanja od PE-ova i prodleđuje ih OS bloku u okviru magistrale veličine osam rezultata. Slika 63 ilustruje rad kolektora rezultata na primeru drugog konvolucionog sloja VGG-16 CNN-a. Naravno, tip sloja nije od velike važnosti za analizu rada niza PE-ova, jer PE-ovi, a tako i ostali blokovi unutar niza PE-ova, nemaju informacije o oblicima kernela i sloja.

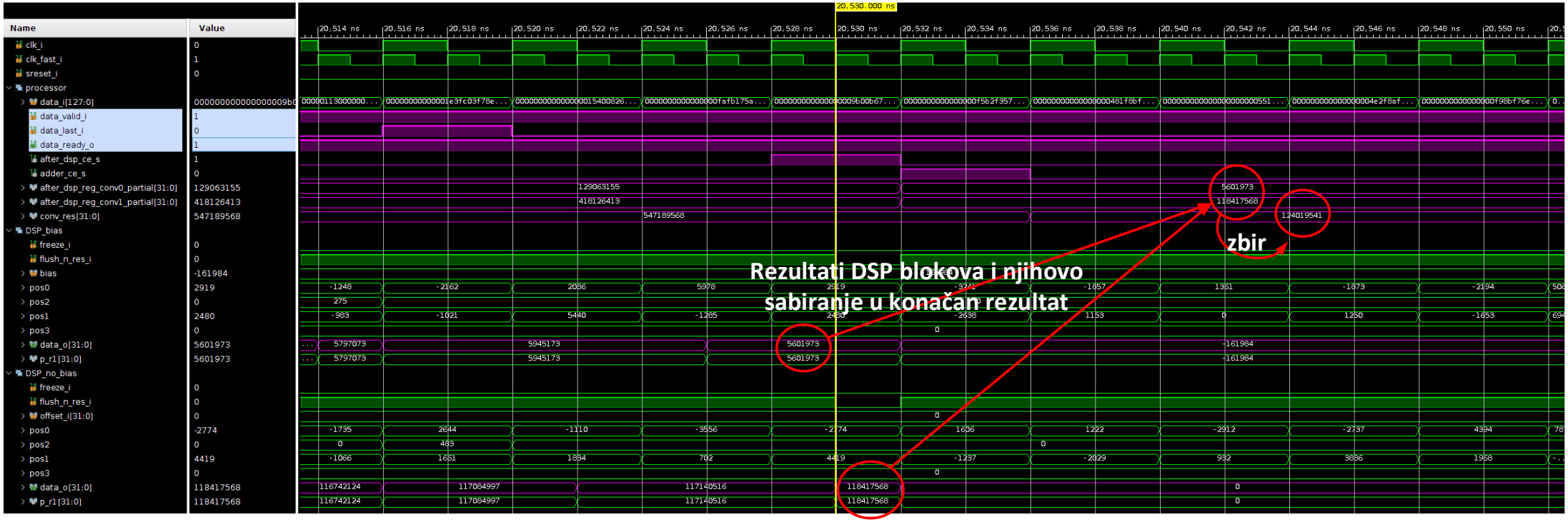


Slika 63 Preuzimanje rezultata od PE-ova i njihovo grupisanje u magistralu širine osam podataka.

Od konfiguracionih ulaza, najbitniji je signal *active_processor_i*, koji obavestava kolektor da li je pojedinačan PE aktivan u ovom sloju ili nije. Kodovanje je takvo da svakom PE-u pripada po jedan bit. Pošto sloj ima 64 kernela, svih 32 PE-a je aktivno te je ovaj signal postavljen na sve jedinice. Sledeći signal (*conv_data_i*) je niz od 32, 32-bitna podatka, to jest, niz koji okuplja sve izlaze PE-ova. Kako bi se lakše uvidela funkcionalna ispravnost, ovaj signal je podeljen na grupe od po osam PE-ova koji se zajedno obrađuju pomoću kolektora rezultata. Prva grupa od osam PE-ova je nazvana PE0-7, druga PE8-15 i tako redom. Pored podataka, bitan signal za opservaciju je i *conv_valid_i* koji daje indikaciju kada je rezultat na izlazu PE-a spreman za preuzimanje. Primetimo da svaki PE ima dodeljen po jedan bit u ovom signalu i da su svih 32 bita postavljeni na 1 u istom taktu. Ovo je posledica ujednačene raspodele računskih zahteva između kernela što je direktna posledica korišćenog algoritma za orezivanje. Poslednji bitan signal za rad kolektora je *chunk_select_reg*, koji predstavlja selekcioni signal multipleksera koji selektuje odgovarajuću grupu od osam PE-ova i prosleđuje ih na izlaz. Po prijemu informacije da su rezultati spremni, ovaj brojač ide u *Round-Robin* maniru od 0 do 3 i selektuje odgovarajuće PE-ove. Kada je 0, na izlaz se prosleđuju rezultati PE-ova od 0 do 7, kada je 1 PE-ova od 8 do 15 i tako redom. Takt u kome se rezultati PE-ova 0 do 7 prosleđuju na izlaz je multipleksera je uokviren crvenim. Preuzimanje rezultata sledeće grupe PE-ova je prikazano žutom bojom. Kao i većina ostalih interfejsa, izlaz multipleksera i kolektora generalno poštuje AXI-stream protokol. Zbog toga, pored magistrale podataka koriste se i kontrolni signali *conv_stream_valid_o* i *conv_stream_ready_i*. Sa druge strane, iako je ulazni interfejs neki derivat AXI-stream protokola, ne možemo reći da je u potpunosti identičan. Naime, ulazni signali su grupisani te ulazni *valid/ready* (*conv_valid_i*/*conv_ready_o*) signal ima širinu od 32 bita, po jedan za svaki PE. Primetimo da u toku obrade prve grupe od osam PE-ova (crvena elipsa), izlazni *conv_ready_o* signal dobija vrednost *hex 000000FF*. To je signal prvoj grupi PE-ova da su njihovi rezultati preuzeti što znači da isti neće biti validni u sledećem taktu. Ovo se oslikava promenom *conv_valid_i* signala u sledećem taktu na vrednost *hex FFFFFF00*.

Pored svih do sada prikazanih talasnih oblika od interesa, u nastavku je još prikazan i rad centralnog modula svakog CNN akceleratora, PE-a. Početak i kraj računanja jedne izlazne tačke OFM-a korišćenjem PE-a je prikazano na slici 64a) i b). Slika 64a) prikazuje početak računanja koje podrazumeva učitavanje bias-a i brisanje sadržaja drugog DSP bloka. Slika 64b) prikazuje završnu fazu izračunavanja, to jest, sabiranje rezultata DSP blokova.

a)



b)

Slika 64 Računanje jedne tačke OFM-a pomoću PE-a.

Podsećanja radi, dva DSP bloka čine jedan PE. Svaki DSP radi na dvostruko većoj frekvenciji od okružujuće logike, što znači da može da izvrši dve MAC operacije u jednom taktu. Pošto PE čine dva DSP-a, jasno je da je kapaciteti jednog PE-a mogućnost izvršavanja četiri MAC operacije u jednom taktu. DSP blokovi su fizički razdvojeni, njihovi rezultati će predstavljati dve polovine vrednosti jedne izlazne tačke OFM-a što znači da će se za dobijanje konačnog rešenja ovi rezultati morati sabrati. Interesantni momenti računanja odziva su njegov početak i kraj. Početak računanja odziva prati odlazak signala *flush_n_res_i* na vrednost 0 što se može uočiti na slici 64a). Ovaj signal se koristi kako bi se iz DSP bloka obrisala vrednost prethodnog izračunavanja, odnosno, da se učita vrednost *bias*-a. Ljubičasti signali pripadaju grupi signala DSP bloka koji inkorporira vrednost *bias*-a dok su zelenom bojom označeni signali koji pripadaju DSP-u koji vrši kalkulacije ne uzimajući *bias* u obzir. Primetimo da se posle vraćanja *flush_n_res_i* signala na 1 vrednosti *data_o* signala postavljaju na vrednost *bias*-a odnosno, na 0 u slučaju drugog DSP bloka. Naravno, *data_o* signal je izlazni signal DSP bloka što znači da oslikava trenutnu vrednost akumulatora. Iako izračunavanje počinje odmah posle negativnog pulsa *flush_n_res_i* signala, vrednosti akumulatora se neko vreme ne menjaju što je direktna posledica procesiranja *padding*-a. Indikacija da se procesira *padding* može biti vrednost 0 na signalima *pos2* i *pos3*. Ovi signali predstavljaju IFM tačke dok su parametri kernela na *pos0* i *pos1*. Prva MAC operacija koja ima rezultat različit od nula je zaokružena crvenim. Ulazni argumenti MAC operacije su vrednosti -161984 (akumulator) i činioci množenja, -955 i 1517. Rezultat ove MAC operacije je -161984 + (-955 x 1517) = -1610719 što se može videti posle prve promene *data_o* signala u okviru DSP-a koji uzima *bias* u obzir. Ova promena *data_o* signala se dešava sa nekoliko taktova zakašnjenja što je posledica dubine *pipeline*-a unutar DSP bloka. Računanje se dalje nastavlja na isti način dok se i poslednji činioci ne uzmu u obzir. Trenutak završetka procesiranja je prikazan na slici 64b).

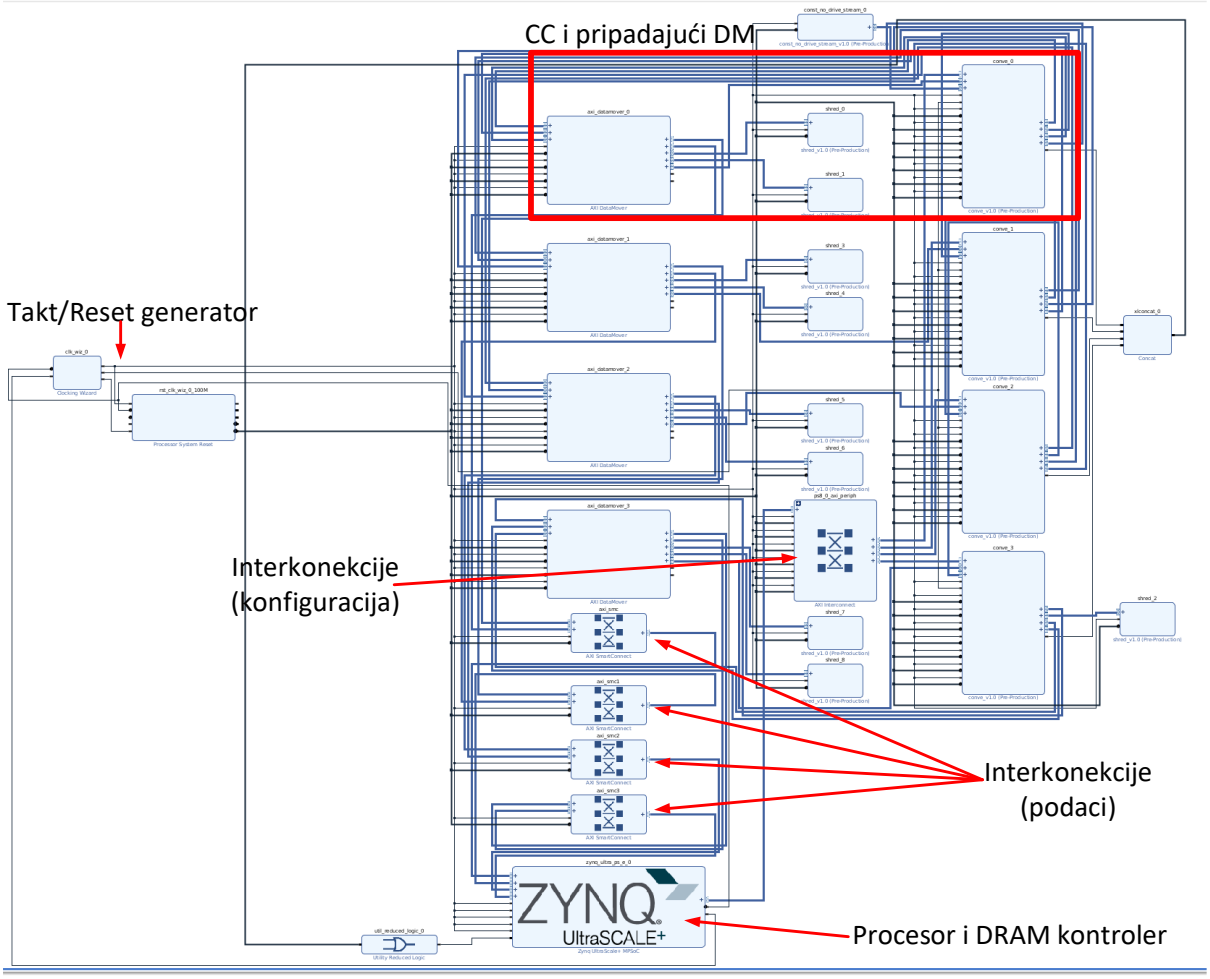
Signal *data_last_i*, na slici 64b) (selektovan), označava poslednju grupu podataka u tekućem izračunavanju, što je indikacija kontrolnoj logici da kreira odgovarajuće signale za preuzimanje rezultata od strane izlazne logike PE-a. Izlazni kontrolni signali su označeni ljubičastom bojom i zaduženi su za kontrolisanje izlaznih registara (*after_dsp_ce_s* i *adder_ce_s*). Prvi, *after_dsp_ce_s,* predstavlja *clock enable* signal za prihvatne registre posle DSP blokova. Ovi registri rade na frekvenciji sistema, što se i može zaključiti iz trajanja signala koji je jednak periodi sporog takta. Primetimo da se u taktu posle aktivacije *after_dsp_ce_s* signala,

vrednosti registara *after_dsp_reg_conv0/1_partial* osvežavaju najnovijim rezultatima sa izlaza DSP blokova (zaokruženi crvenom su vrednosti akumulatora koji se smeštaju u navedene registre). Sve što je preostalo je da se vrednosti iz *after_dsp_reg_conv0/1_partial* saberu i upišu u registar nazvan *conv_res*. Signal dozvole upisa u *conv_res* registar je *adder_ce_s* koji, kako mu ime kaže, dozvoljava upis rezultat sabirača u registar. Ovime je računanje završeno i rezultat je spreman za preuzimanje od strane kolektora rezultata. Primetimo da u toku predaje rezultata iz DSP blokova okružujućoj logici nema pauze, već se nastavlja dalje procesiranje sloja. Ovo je jasno uočljivo ako se pogledaju selektovani signali *data_valid_i* i *data_ready_o* (selektovani, ulazni kontrolni signali u PE). Ovi signali konstantno imaju vrednost 1 što znači da se razmena podataka dešava u svakom taktu. Kao potvrda da su akumulatori ispražnjeni i da je otpočto sledeće izračunavanje, možemo pogledati signal *flush_n_res_i* koji ima negativan puls negde na sredini slike 64b), posle čega je akumulator jednog DSP-a postavljen na vrednost koju ima bias, dok je drugi postavljen na 0. Opisani proces se ponavlja dokle god se ne izračunaju sve tačke OFM-a.

7.2 Verifikacija na FPGA platformi

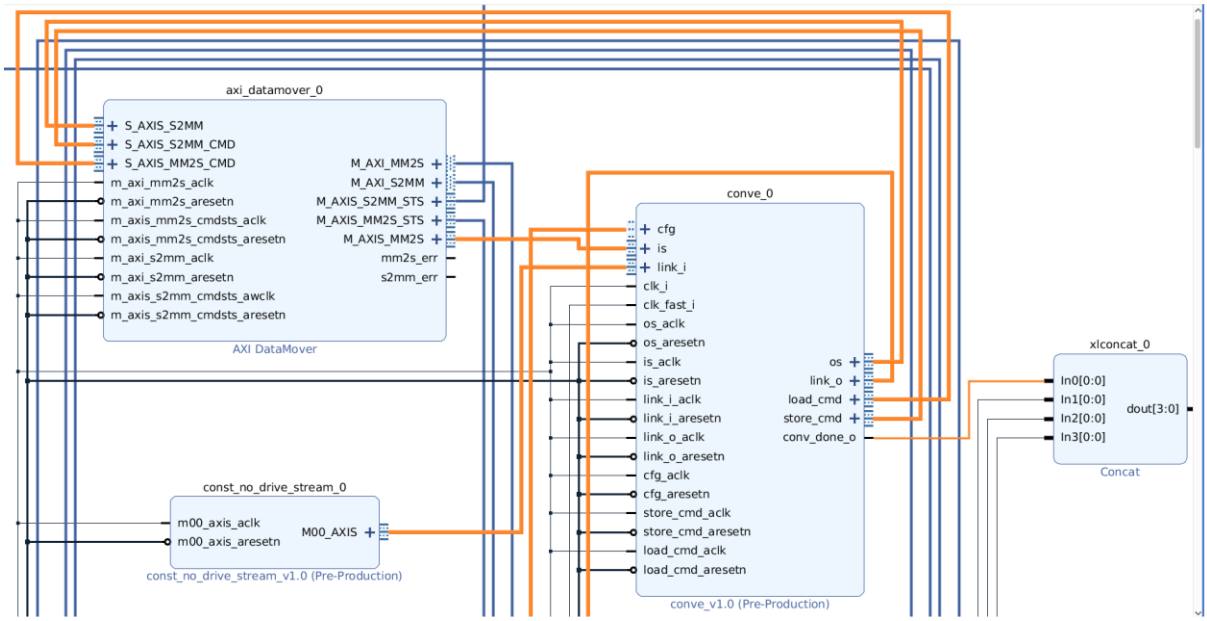
Pre testiranja na FPGA sistemu, potrebno je zapakovati razvijene RTL kodove u IP-eve i izvršiti povezivanje IP-eva sa procesorskim sistemom i reset/klok generatorom. Na slici 65 je prikazan izgled sistema koji je implementiran na FPGA čipu. U dnu slike se može primetiti procesorski sistem koji u sebi sadrži procesore opšte namene, ali i DRAM kontroler na koji su povezana četiri jezgra, svaki u konfiguraciji sa jednim CC modulom. Na vrhu slike su uokvireni, jedno jezgro sa pripadajućim *DataMover*-om. Detaljnija konekcija sa DM-om je prikazana na slici 66. Kako bi se povezali DM-ovi sa procesorskim sistemom, Vivado alat automatski instancira *Axi smart connect* (SMC) blokove na *Axi-Full* konekcije između DRAM kontrolera i DM-ova. Pored DRAM konekcija, procesorski sistem je povezan i preko *Axi-Lite* protokola sa svim razvijenim jezgrima kako bi dostavio konfiguracione parametre za svaki sloj. Pošto faza konfigurisanja traje izuzetno kratko u poređenju sa procesiranjem slojeva, jedan port procesorskog sistema je dovoljan za sva četiri jezgra. Na kraju, obavezni deo svakog digitalnog sistema je i takt/reset generator koji se može videti u skroz levom kraju na slici 65. Reset generator se automatski instancira od strane alata prilikom povezivanja. Takt generator je konfigurisan prema potrebama sistema tako da isporučuje potrebna dva takt signala do 250 MHz i 500 MHz fazno poravnata. Napomenimo i to da je za takt generator iskorišćena gotova komponenta

kompanije Xilinx, naziva *Clock Wizard*. Prilikom povezivanja, bitno je ispoštovati da takt signali budu povezani na odgovarajuće portove jezgara koja se testiraju, ali i svih ostalih sekvencijalnih komponenti sistema.



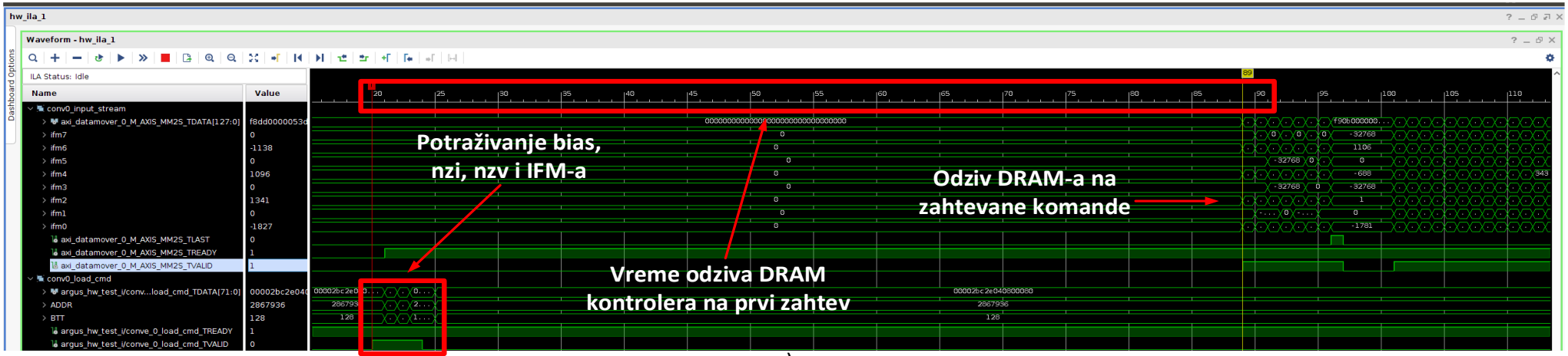
Slika 65 Prikaz sistema na kome su vršena merenja na FPGA SoC-u.

Slika 66 donosi detaljan prikaz povezivanja jezgra sa DM-om koristeći *Axi-Stream* protokole. Tako možemo videti port imena *is* koji je povezan na DM zajedno sa izlaznim interfejsom naziva *os*. Pošto je reč o prvom od četiri jezgra, *link_i*, koji predstavlja ulaz za *Link* komponentu, je povezan na blok koji je konstantno u stanju mirovanja, to jest, *valid* signal je povezan na vrednost 0. Sa druge strane, *link_o* je spojen na sledeće jezgro. Pored navedenog, mogu se videti i *load/store_cmd* interfejsi koji dostavljaju odgovarajuće komande DM-u za transfer podataka iz odnosno, ka DRAM-u. Naravno, da bi se svaki sloj posebno konfigurisao, postoji i *cfg* interfejs Axi-Lite tipa, preko kog se prihvataju konfiguracije. Na kraju, u desnom delu slike se može primetiti komponenta naziva *Concat* koja sumira četiri *conv_done_o* signala iz svakog jezgra i generiše odgovarajući prekid za procesor po završetku procesiranja sloja.

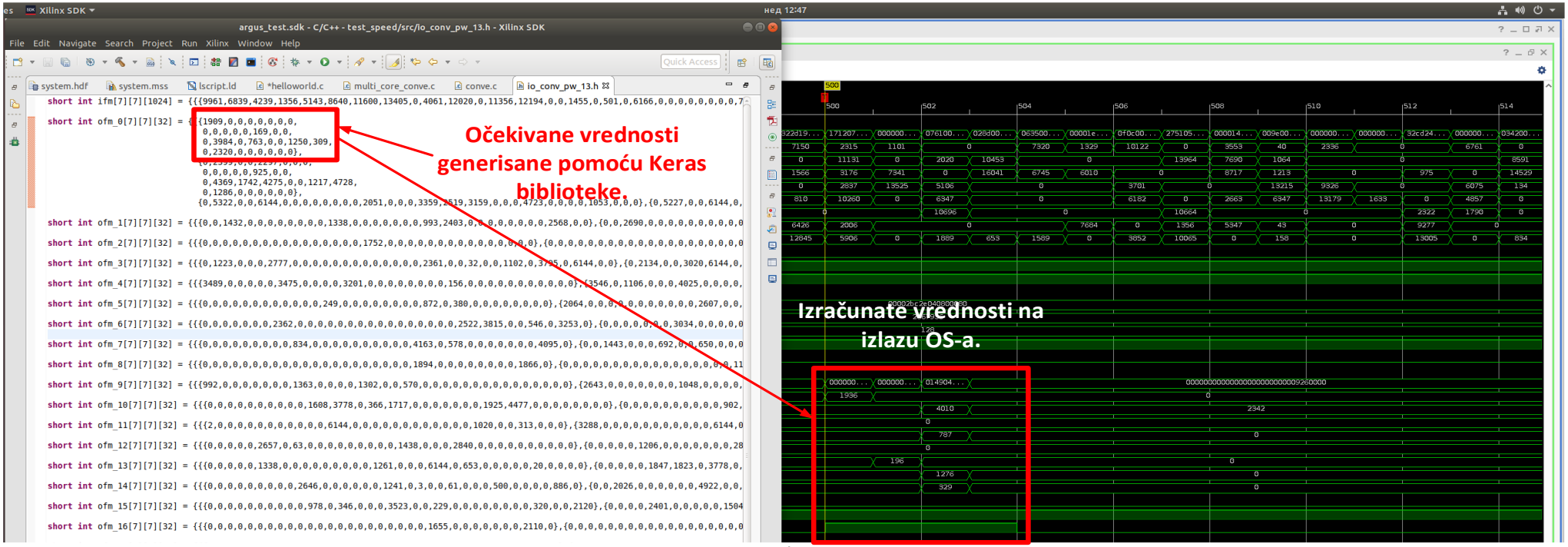


Slika 66 Detaljan prikaz povezanosti Argus akceleratora sa DM-om i logikom prekida.

Zbog kompleksnosti sistema, prikazavanje talasnih oblika na ovom nivou je zahtevan zadatak ukoliko se želi postići visok nivo razumevanja. Zbog navedenog u nastavku su prikazane dve slike koje prikazuju karakteristične delove procesiranja sloja, više ilustrativno, da bi se prikazao način testiranja, a manje da bi se prikazali detalji kao u slučaju simulacija na RTL nivou. Slika 67a) prikazuje početak procesiranja sloja koji je započet slanjem četiri uzastopne komande ka DM-u. Napomenimo da talasni oblici signala na naredne dve slike pripadaju *PointWise* sloju 13, MobileNet mreže. Pošto je u ovom tipu slojeva keširanje neaktivno, kompletan IFM je zatražen samo jednim zahtevom, poslednjim od četiri uokvirena crvenim pravougaonikom. Prva tri čine zahtev za *bias*, NZI i NZV. Na desnom delu slike 67a) može se videti odgovor DRAM-a koji posle, okvirno 70 taktova, počinje da dostavlja tražene podatke. Primetimo da se posle *bias*-a, javlja pauza od samo nekoliko taktova, u poređenju sa prvobitnih 70, pre nego se nastavi isporuka NZI vrednosti. Na drugom delu slike 67, može se videti odziv jezgra, to jest, prvi rezultati izračunavanja. Na slici 67b) je prikazan i izgled .h fajla koji sadrži ulaz/izlaz za pomenuti sloj i sa kojim se vrši poređenje. Crveni pravougaonik predstavlja očekivane vrednosti, dok su na talasnim oblicima uokvireni dobijeni rezultati. Može se uočiti mala razlika u vrednostima koja proizilazi iz toga što Keras biblioteka radi sa brojevima sa pokretnom tačkom, dok Argus akcelerator radi u 16-bitnoj aritmetici sa fiksnom tačkom. Prvi očekivani rezultat ima vrednost 1909, dok je njegova vrednost, izračunata od strane Argus akceleratora, 1936.



a)

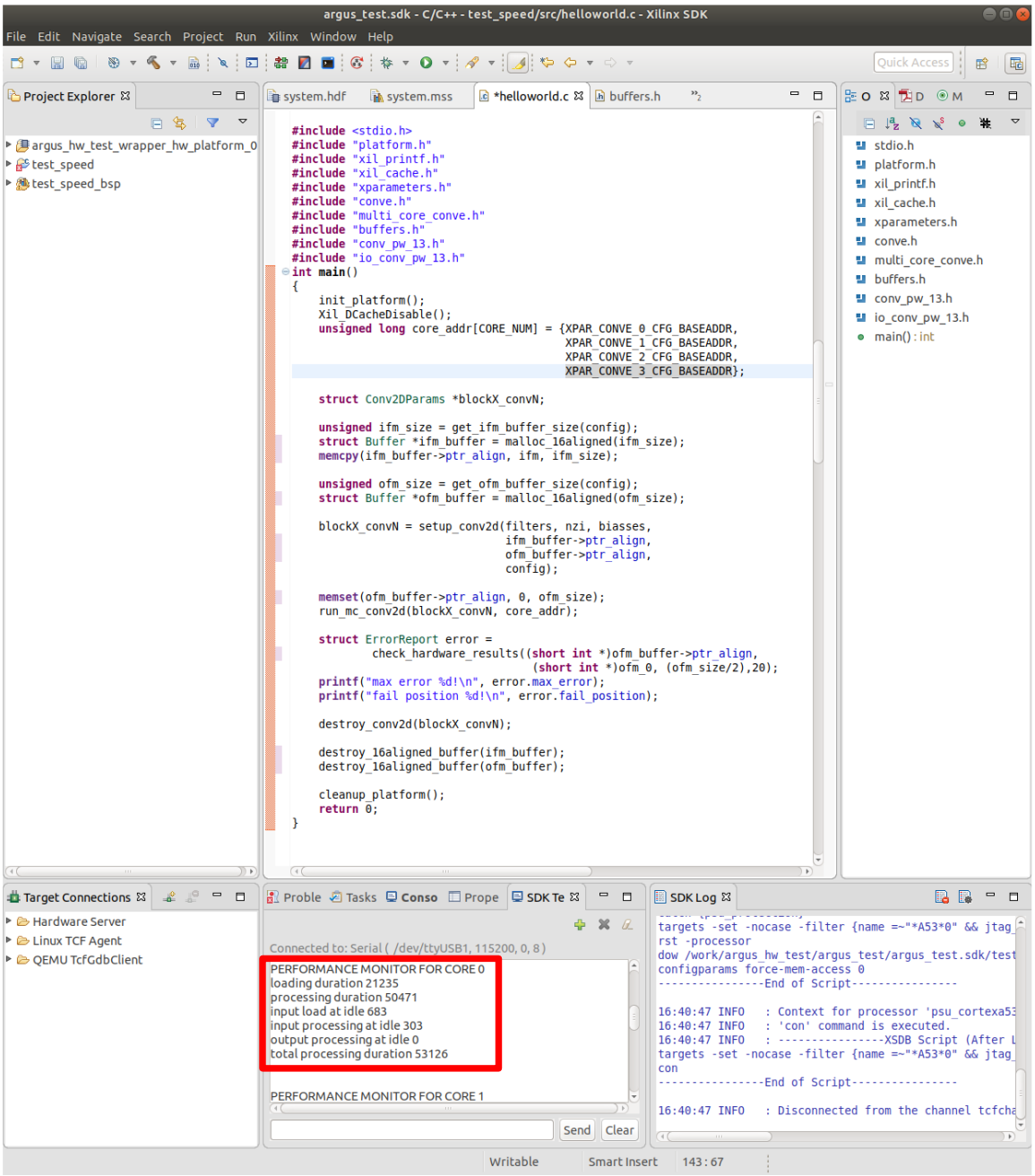


b)

Slika 67 Početak procesiranja sloja a), i preuzimanje prvih rezultata izračunavanja konvolucije b).

Primetimo da je ova razlika vrlo mala. Zbog robusnosti koju imaju CNN-ovi, ovakve razlike nemaju nikakav uticaj na konačni rezultat klasifikacije mreže. Ukoliko nastavimo porediti rezultate, primetićemo da na svakom postoje slična odstupanja, ali da vrednosti odgovaraju očekivanim u određenim granicama.

Kako bi se stekla slika i o softverskom delu verifikacionog okruženja na FPGA platformi, na slici 68 je prikazana *main* funkcija i rezultati očitavanja monitora performansi po završetku procesiranja osam grupa od po 32 kernela *Pointwise* 13 sloja, izračunate na CC modulu 0.



Slika 68 Prikaz softverskog okruženja i povratne informacije o performansama jednog konvolucionog jezgra.

Što se C koda tiče, na početku su uključene potrebne biblioteke i konfiguracija, odnosno IFM/OFM navedenog konvolucionog sloja. Dalje, inicijalizovan je SoC i isključeno je keširanje zato što akcelerator upisuje direktno u memoriju što se ne reflektuje u kešu. Ukoliko keš ne bi bio isključen, procesor bi na kraju sloja poredio prethodno keširane vrednosti te bi uvek prijavljivao grešku. Jasno je da se ovakve stvari mogu izbeći sofisticiranim rešenjima, međutim to nije tema ove doktorske disertacije. U nastavku se vrši inicijalizacija početnih adresa registarskih mapa CC jezgara u adresnom prostoru sistema. Takođe, vrši se alokacija za potrebne bafere i to poravnata na 16 bajta kako se ne bi dešavalo da određene grupe od 8 vrednosti budu podeljene na dva pristupa memoriji. Potom se vrši konfigurisanje jezgara pomoću funkcije *setup_conv2d* i resetovanje vrednosti prihvatnog bafera. Pozivom funkcije *run_mc_conv2d* (engl. *run multi-core convolution 2d*) započinje se izvršavanje konvolucionog sloja. Po završetku, proveravaju se rezultati pomoću funkcije *check_hardware_results* te se oslobađaju zauzeti baferi za IFM/OFM i konfiguraciju sloja pomoću funkcija *destroy_conv2d/16aligned_buffer*.

U donjem delu slike 68 je prikazan sadržaj monitora performansi posle procesiranja *Pointwise* 13 konvolucionog sloja. Navedeni rezultati su prikazani za slučaj kada je aktivan CC modul sa indeksom 0. Ovaj sloj čine 32 grupe od po 32 kernela, što znači da se na svakom CC-u izvrše 8 grupa od 32 kernela. Pošto IFM ima dubinu od 1024, a kernel dimenzije 1x1x1024, za svaku izlaznu tačku je potrebno 128 taktova da bi se izračunala. OFM ima oblik 7x7x1024, te je jasno da je za jednu grupu od 32 kernela potrebno 7x7x128 taktova, to jest, 6276 taktova u idealnom slučaju. Pošto svaki CC blok obrađuje osam ovakvih grupa, to je 6276x8 što je ukupno 50176 taktova za procesiranje. Na slici 68 se može videti da je ukupno vreme procesiranja (*total processing duration*) 53126 taktova sa svim zastojima koji potiču od nesavršenosti DRAM kontrolera. Ukupno za svih osam grupa, ograničenja DRAM kontrolera unose 2950 taktova zastoja, odnosno, u proseku oko 370 taktova po jednoj grupi. Ovo odgovara očitanoj vrednosti *input processing at idle* koji za poslednju grupu iznosi 303 takta. Pored navedenog, možemo videti da značajan deo procesiranja navedenog sloja uzima učitavanje parametara, ukupno 21235 taktova. Analiza performansi ovog sloja jasno pokazuje visoku efikasnost arhitekture akceleratora koji ne provodi više od 6% vremena u zastoju jednom kada procesiranje odpočne. Dodatno, treba primetiti da ovi zastoji potiču isključivo od DRAM kontrolera.

8 Eksperimentalni rezultati

U okviru eksperimentalne sekcije su predstavljeni rezultati implementacije kao i ostvarene performanse Argus akceleratora na FPGA razvojnom okruženju. Ovo poglavlje treba jasno da pokaže koje su prednosti Argus arhitekture u odnosu na publikovane akceleratore sa akcentom na rešenja koja su namenjena upotrebi u *embedded* aplikacijama. Prvi deo poglavlja donosi rezultate implementacije u smislu zauzeća raspoloživih resursa zajedno sa analizom u koje FPGA komponente je moguće instancirati odgovarajuću Argus verziju. Naravno, prikazano je i poređenje sa objavljenim FPGA rešenjima kako bi se demonstrirala fleksibilnost projektovanog akceleratora. U drugom delu su prikazane evaluirane performanse u smislu računskih kapaciteta Argus akceleratora. Takođe, neizostavno je i poređenje sa drugim rešenjima kako u smislu apsolutnih performansi tako i raznih metrika koje ističu efikasnost akceleratora.

8.1 Rezultati implementacije

8.1.1 Podešavanje alata

Pre početka implementacije, potrebno je kreirati *constraint* fajl kako bi se alat usmerio ka željenim rezultatima. Ovaj fajl sadrži instrukcije koje se najviše tiču vremenskih ograničenja. Tu se pre svega misli na periodu takt signala i na maksimalno vreme potrebno da se stabilizuju ulazno/izlazni signali. U slučaju kada se implementacija vrši na FPGA SoC-u i kada akcelerator nema interfejs ka eksternim pinovima na ploči, alat će sam generisati ova ograničenja. Dovoljno je povezati Argus akcelerator sa DRAM-om u okviru u okviru IP integratora, a alat će kreirati potrebna ograničenja. Uz navedena povezivanja, krucijalno je korektno napisati ograničenja u vezi sa periodama takt signala. I ovaj deo je automatski generisan od strane alaza zato što je za kreiranje takt signala korišćena slobodno dostupna komponenta *Clock Wizard*. Komponenta je razvijena od strane Xilinx kompanije i koristi se kao IP. Prilikom instanciranja moguće je u vizualnom okruženju podesiti željene parametre. U slučaju integracije Argus akceleratora, komponenta je konfigurisana tako da ima dva takt signala, jedan frekvencije 250 MHz, a drugi 500 MHz. Kao ulaz, *Clock Wizard* koristi jedan oscilator iz kog generiše oba takt signala što je potreban uslov da bi se koristila *multi-pumping* tehnika opisana u poglavlju koje se bavi mikro-arhitekturom PE-a. Pored takt signala, *Clock Wizard* generiše i reset signale prilagođene frekvenciji na kojoj rade flip-flopovi. To znači dva reset

signala, jedan za flip-flopove koji rade na 500 MHz, drugi za dvostruko sporiji takt signal. Posle instanciranja *Clock Wizard* IP-a i povezivanja Argus arhitekture na procesorski sistem, alat sam kreira *constraint* fajl i proces sinteze i implementacije može započeti.

Sinteze i implementacije je izvršen pomoću alata kompanije Xilinx, naziva Vivado Suite, verzije 2019.1. FPGA okruženje na kome je testiran akcelerator je takođe, od kompanije Xilinx, oznake ZCU104, na kojoj se nalazi čip ZU7 MPSoC. Korišćeno podešavanje sinteze je *Flow_PerfOptimized_high* koje je već dostupno u okviru alata. Što se tiče implementacije (*Place & Route* faze), ona je sprovedena korišćenjem takođe dostupne strategije naziva *Performance_NetDelay_high*.

8.1.2 Implementirane konfiguracije akceleratora – rezultati i analiza

Kako bi se pokazala fleksibilnost i mogućnost široke primene Argusa, akcelerator je implementira u tri različite varijante. Kompaktna verzija ima jednu instancu CC i DLP modula i kao takva može naći primenu u širokoj grupi dostupnih FPGA razvojnih okruženja. Argus CNN akcelerator sa jednim CC blokom je namenjen najpristupačnijim sistemima u kojima performanse nisu prioritet. Drugu varijantu predstavlja akcelerator sa dva CC i jednim DLP jezgrom. U ovoj konfiguraciji, akcelerator nudi skoro dvostruko bolje performanse u odnosu na instancu sa jednim CC modulom, po cenu nešto većeg zauzeća hardverskih resursa. Ciljana grupa FPGA okruženja nije značajno drugačija od one za koju je namenjena Argus arhitektura sa jednim CC blokom te je cilj analize ove konfiguracije da se predstavi kako se skaliraju performanse. Na kraju, za najsnažniju testiranu konfiguraciju je odabrana instanca sa četiri CC i jednim DLP modulom. U ovoj konfiguraciji, Argus akcelerator je namenjen FPGA okruženjima koja spadaju u srednji nivo složenosti u *embedded* svetu. Ova konfiguracija nudi visoke performanse po cenu većih zahteva za hardverskim resursima. Primetimo da u svakoj od tri konfiguracije, Argus akcelerator ima samo jednu DLP jedinicu. Ovo je posledica značajno većih računskih zahteva koje imaju konvolucioni slojevi u odnosu na ostale, te bi za testirane CNN modele bilo suvišno instancirati više DLP blokova. U tabeli 8 su prikazani rezultati implementacije navedene tri konfiguracije Argus akceleratora koristeći opisane strategije u prethodnom paragrafu.

Posle implementacije, potrebno je uporediti dobijene rezultate sa očekivanim. Ukoliko postoje drastična odstupanja od procene veličine akceleratora, postoji mogućnost da su neki

delovi RTL koda napisani tako da alat nije uspeo da ih protumači na željeni način. U ovom slučaju je potrebno prilagoditi stil kodovanja za kritične module. Druga mogućnost je da je došlo do previda prilikom inicijalne procene.

Konfiguracija	DSP	LUT	BRAM	Frekvencija
Argus 1CC	74	14k	55	250 MHz
Argus 2CC	140	25k	96.5	250 MHz
Argus 4CC	272	46k	218.5	250 MHz

Tabela 8 Zauzeće hardverskih resursa u slučaju tri različite konfiguracije Argus akceleratora.

Najjednostavnije je proveriti broj iskorišćenih DSP blokova. Jedan CC modul ima 32 PE-a što ukupno daje 64 DSP bloka po CC-u. Jedan DLP blok koristi 8 DSP blokova kako bi bio u stanju da procesira osam vrednosti u svakom taktu. DSP blokovi u DLP jezgru se koriste za množače pomoću kojih se računa, na primer, srednja vrednost u *Average pooling* slojevima. Ukupno 1 CC i 1 DLP modul zahtevaju 72 DSP bloka. Preostala dva DSP-a, u prvoj konfiguraciji tabele 8, se koriste za generisanje adresa za smeštanje podataka u DRAM, odnosno, za potraživanje podataka iz DRAM-a. Kao i u slučaju DLP jezgra, i za generisanje početnih adresa štapića se koristi operacija množenja te je alat procenio da je optimalan način implementacije množača pomoću DSP blokova. Ovo je najčešće slučaj kada su činioci brojevi čija širina prelazi 8 bita. Primenjujući identičnu analizu, jasno se može zaključiti da se rezultati poklapaju sa očekivanjima i u slučaju Argus akceleratora sa dva, odnosno, četiri CC bloka. Uz DSP blokove, analiza iskorišćenih BRAM-ova se takođe može vrlo jednostavno izvesti. Tako možemo zaključiti da je za čuvanje parametara kernela (NZI i NZV) potrebno 34 BRAM-a. Ostatak odlazi najviše na keš memoriju unutar IS modula. U zbiru IS, CC i DLP moduli zahtevaju 55 BRAM ćelija što je u skladu sa očekivanim. U slučaju izuzetno kompaktnih FPGA platformi, ovaj broj se može redukovati za 5 do 10 BRAM-ova smanjivanjem veličine keš memorije IS modula. Unutar CC modula nije moguće praviti uštede. Ovakvo smanjivanje keš memorije neće značajno uticati na performanse no ipak je za očekivati degradaciju. Napomenimo i to da je na nekim FPGA čipovima dostupan i Ultra-RAM [55] koji u celosti može preuzeti ulogu BRAM-a iz IS modula. Na kraju, analiza iskorišćenosti LUT-ova nije vršena jer su LUT-ovi osnovne gradivne komponente i sva logika je implementirana pomoću njih. Ovo znači da su korišćeni prilikom implementacije gotovo svake komponente te je samo provereno da su najveći blokovi korektno implementirani. Prevashodno se misli na logiku koja selektuje potrebne IFM

tačke jer je akcenat disertacije na smanjenju upravo ovog dela akceleratora. Kao i u slučaju DSP i BRAM blokova, i ovde dobijeno zauzeće resursa odgovara očekivanjima.

8.1.3 Poređenje sa prethodno publikovanim FPGA baziranim akceleratorima

Poređenje sa drugim rešenjima je urađeno na dva načina. Prvi segment poglavlja poredi krajnje rezultate implementacije prikazane u tabeli 9. Ovakvo poređenje često maskira krupne nedostatke arhitektura te je u te svrhe izvršena dodatna analiza predstavljena tabelom 10. Za razliku od apsolutnih brojeva prikazanih u tabeli 9, tabela 10 prikazuje u koje je trenutno dostupne FPGA SoC-eve moguće implementirati sve navedene akceleratore u datim konfiguracijama. Ovakav pristup je izabran iz više razloga od kojih je jedan taj što je arhitektura kreirana kako bi maksimizovala performanse akceleratora na trenutno dostupnim, a ne hipotetičkim FPGA čipovima. Na primer, postoje akceleratori koji koriste relativno mali broj raspoloživih DSP blokova, ali zahtevaju velike količine ostalih resursa te ih je teško ili nemoguće implementirati na većini kompaktnih FPGA platformi. Drugi razlog ovakvog poređenja je da se pokaže da je ravnomerno korišćenje dostupnih resursa od krucijalne važnosti za široku primenu akceleratora. Ova analiza se provlači kroz celo poglavlje te su podaci iz tabele 10 bitni u više metrika koje su korišćene u nastavku.

Akcelerator	Broj DSP blokova	LUT (FPGA)	BRAM (FPGA) Instanci/[kb]	Aritmetika (bita)
FpgaConvNet	900	218k	307/11.070	16
Snowflake	256	-	170/6.120	16
NullHop (FPGA)	128	229k	386/13.896	16
NEURAghe	864	88k	320/11.520	16
CoNNa C4	256	267k	596/21.456	16
Caffeine	1058	100k	391/14.094	16
Depthwise optimized accelerator [47]	3283	121k	-	8
Eyeriss v2	384	-	-	8
Thinker	1024	-	-	8/16
Envision	512	-	-	4/8/16
[38]	220	13.4	98/3528	8
[36]	1144	252	456/16.416	16
[37]	1350	178.1	730/26.280	8/16
NVDLA [56]	256	-	-	4/8/16
Argus 4 CC-a	272	46k	218.5/7.866	16
Argus 2 CC-a	140	25k	96.5/3.474	16
Argus 1 CC	74	14k	55/1.980	16

Tabela 9 Poređenje apsolutnog zauzeća hardverskih resursa.

Pre dalje analize rezultata iz tabele 9, bitno je napomenuti da su akceleratori prezentovani u radovima [36], [37], [38], kao i FpgaConvNet skalabilni te ih je moguće konfigurisati tako da zauzimaju manje resursa od navedenog. Ipak, navedeni brojevi su korišćeni iz razloga što su u analizi performansi korišćene baš navedene instance, to jest, rezultati koje iste ostvaruju. Ovakav pristup je odabran, jer ne možemo biti sigurni da će manje instance imati proporcionalne performanse. Pretpostavka je da su autori naveli rezultate za instance koje im najviše idu u prilog.

Arhitekture FpgaConvNet i Caffeine su implementirane koristeći HLS pristup (za razliku od RTL u slučaju Argus akceleratora) koji je izuzetno fleksibilan, ali nije efikasan u mapiranju koda na hardverske resurse. Takođe, HLS akceleratori najčešće nemaju fiksnu arhitekturu već se razlikuju od modela do modela CNN-a koji akceleriraju. Na primer, ukoliko je FPGA konfigurisan na jedan način, tako da je u mogućnosti da akcelerira MobileNet v1 CNN, neće biti u mogućnosti da akcelerira i ResNet50. Ukoliko je potrebno promeniti CNN model koji se izvršava, potrebno je ponovo proći kroz sintezu i implementaciju te ponovo konfigurisati FPGA kolo. Drugim rečima, ovo znači da ovako implementirani akceleratori najčešće nisu u stanju da menjaju CNN modele koje akceleriraju, u toku rada (engl. *on-the-fly*). Za razliku od HLS akceleratora, Argus je univerzalni akcelerator koji je nezavisan od CNN modela. Sa stanovišta zahteva za resursima, oba akceleratora iskorišćavaju veliki broj DSP blokova (ekvivalentno MAC jedinica ASIC tehnologiji) i BRAM-ova. Veliko iskorišćenje DSP blokova navodi da je reč o arhitekturama visokih performansi, međutim, u drugom delu ovog poglavlja ćemo videti da su performanse uporedive sa Argus CNN akceleratorom koji koristi i do 3 puta manje DSP blokova. Ovo je direktna posledica neefikasnog mapiranja operacija na DSP blokove od strane HLS kompajlera. Kao zaključaj, možemo reći da FpgaConvNet i Caffeine nije moguće instancirati na kompaktne FPGA platforme.

U prethodna dva pomenuta akceleratora, ograničavajući faktor je ukupan broj DSP blokova koji je potreban za implementaciju. Sa druge strane, CoNNa C4 i NullHop koriste značajno manji broj DSP blokova te je za očekivati da mogu naći primenu u FPGA sistemima ograničenih resursa. Ipak, ovo nije slučaj jer ove dve arhitekture imaju izraženu upotrebu LUT-ova. Tako ni CoNNa C4, a ni NullHop ne mogu biti implementirani u FPGA komponente malog kapaciteta. U slučaju CoNNa-e, velika upotreba LUT-ova je opravdana visokom efikasnošću koju ima akcelerator. Naime, jedino je CoNNa u stanju da preskače nepotrebne MAC operacije

prilikom procesiranja slojeva neovisno od korišćenog algoritma za orezivanje. Dodatno, CoNNa je u stanju da preskače i nule u IFM-u, čemu duguje i status najefikasnijeg nama poznatog akceleratora, kada se performanse skaliraju po iskorišćenom DSP bloku. Ovo nije situacija kod NullHop akceleratora, koji postiže izuzetno skromne rezultate u pogledu performansi. Za razliku od NullHop-a, NEURAghe ne zahteva veliki broj LUT-ova, ali iskorišćava 864 DSP bloka što je daleko izvan opsega pristupačnih FPGA čipova. Snowflake predstavlja suprotnost svemu do sada navedenom. Potreban broj MAC jedinica i BRAM-ova je zadovoljavajući za široku primenu akceleratora. Iako zauzeće LUT-ova nije jasno predstavljeno u radu, analizom arhitekture se može doći do zaključka da ni u slučaju LUT-ova Snowflake nema ekstremne zahteve. Ovo je odlika dobrih akceleratora koji ravnomerno koriste raspoložive resurse te ih je najčešće jednostavno skalirati u slučaju da su potrebne bolje performanse. Mana Snowflake akceleratora je što nije u mogućnosti da akcelerira orezane CNN-ove te su njegove performanse značajno skromnije u poređenju sa ostalim arhitekturama. Na kraju analize FPGA baziranih rešenja, ostaje akcelerator prikazan u [47]. Ova arhitektura je visoko optimizovana za rad sa CNN-ovima koji imaju Depthwise konvolucione slojeve. Jedna od njih je i MobileNet v1. Ono što se jasno ističe je ekstremno korišćenje DSP blokova (3283). Ova količina je dostupna samo na najvećim FPGA čipovima te akcelerator svakako nije pogodan za *embedded* aplikacije. Ono što nije jasno, je zašto su autori kreirali ovako kompleksan akcelerator za CNN model koji je prevashodno razvijen za *embedded* sisteme. Sa jedne strane, akcelerira izuzetno kompaktan CNN model, ali zahteva najsnažnije FPGA čipove. Povrh svega, performanse su beznačajno bolje u poređenju sa Argus arhitekturom (manje od 50% bolje) ako se uzme u obzir da akcelerator koristi 12 puta više DSP blokova. Ostale navedene arhitekture su implementirane kao ASIC te pravično poređenje nije moguće.

Kao što je rečeno u uvodnom delu, tabela 10 prikazuje u koje FPGA čipove je moguće smestiti analizirane FPGA akceleratore. Takođe, u tabeli 10 je prikazan i odnos potrebnih LUT-ova po jednom DSP bloku za svaku arhitekturu pojedinačno, ali i raspoloživi odnos za navedene čipove. Ovaj odnos je jedan od parametara koji nije često prikazan u publikacijama, ali pokazuje balansiranost iskorišćenja resursa. Kod akceleratora koji imaju mogućnost procesiranja orezanih CNN-ova, ovaj parametar najčešće oslikava kompleksnost logike za preskakanje nepotrebnih MAC operacija i iskazan je nekim većim brojem (obično većim od

200). Sa druge strane, kod arhitektura koje ne koriste benefite orezanih CNN-ova, ovaj odnos će obično biti neki značajno manji broj u odnosu na ove druge. Pošto znamo da procesiranje orezanih mreža značajno poboljšava performanse akceleratora, jasno je da je od velikog interesa pružiti podršku za preskakanje nepotrebnih MAC operacija. Sa druge strane, potrebno je zadržati kompleksnost logike u razumnim granicama kako zauzeće LUT-ova ne bi bilo prekomerno, što može drastično suziti odabir čipova na koje je moguće instancirati akcelerator sa željenim performansama.

Akcelerator	**Zahtevani odnos LUT/DSP**		**Zynq UltraSCALE+ MPSoC**					
		Zu2	**Zu3**	**Zu4**	**Zu5**	**Zu6**	**Zu7**	**Zu9**
Dostupni odnos LUT/DSP		196	196	120	93	109	133	109
[38]	60	•	•	•	•	•	•	•
Caffeine	94					•		•
Snowflake[(a)]	-		•	•	•	•	•	•
NEURAghe	101					•		•
[37]	131							•[(b)]
Argus 4 CC-a	168			•	•	•	•	•
Argus 2 CC-a	177	•	•	•	•	•	•	•
Argus 1 CC	195	•	•	•	•	•	•	•
[36]	200							•[(b)]
FpgaConvNet	242						•[(b)]	•[(b)]
CoNNa (C4)	1042							•
NullHop	1789							•

(a) Snowflake nema prijavljeno zauzeće LUT-ova te je mogućnost implementiranja na određeni FPGA čipa ocenjeno isključivo na osnovu broja DSP-eva i BRAM-ova.

(b) Akceleratori su skalabilni. Prikazani rezultati su za konfiguracije korišćenje prilikom analize performansi.

Tabela 10 Mogućnost instanciranja akceleratora u neke od dostupnih FPGA čipova.

Možemo primetiti da je odnos dostupnih LUT-ova po jednom DSP-u u opsegu od 100 do 200 LUT/DSP za sve razmatrane FPGA komponente. Detaljnije, kompaktni FPGA čipovi kao što su Zu2, Zu3 i Zu4 imaju nešto manju gustinu DSP-eva te je ovaj parametar i nešto veći. Primetimo da se sve tri konfiguracije Argus akceleratora nalaze u pomenutom opsegu. Uz Argus, dobar odnos imaju i NERAghe i akceleratori prezentovani u radovima [36] i [37]. Ipak Argus može da bude instanciran u najširu grupu čipovima, za razliku od preostala 3. Mana tri pomenuta rešenja se ogleda u činjenici da su izuzetno veliki akceleratori te ih je, bez obzira na dobar odnos LUT/DSP, nemoguće implementirati na kompaktnim FPGA čipovima. Napomenimo još jednom i to da su ova tri akceleratora skalabilna te postoji mogućnost da bi u nekoj

konfiguraciji mogli biti korišćeni na manjim čipovima. Ipak, da bi poređenje bilo pravedno, tabela 10 inkorporira one konfiguracije koje su u nastavku korišćenje u analizi performansi.

Izvan opsega od 100 do 200 LUT/DSP se nalaze Caffeine, arhitektura prikazana u radu [38], FpgaConvNet, CoNNa (C4) i NullHop. Prva dva imaju povoljan odnos, to jest, manji od 100 što je indicija da su rešenja bolja od ostalih. Međutim, pokazuje se da je smanjena upotreba LUT-ova direktna posledica nemogućnosti da akceleratori preskaču nule u parametrima/IFM-u. Ovo za rezultat dovodi do značajno lošijih performansi u poređenju sa ostalim arhitekturama. Uz navedeno, akcelerator prikazan u radu [38] je vezan za Xilinx-ove SoC-eve zato što koristi specifične skupove instrukcija dostupnih procesora unutar procesorskog podsistema. Drugim rečima, rešenje nije nezavisno od FPGA platforme u meri u kojoj druga jesu. FpgaConvNet, CoNNa (C4) i NullHop su rešenja čiji LUT/DSP parametar daleko prevazilazi dostupan odnos ovih elemenata na prikazanim FPGA čipovima. Sa jedne strane, ovakav odnos navodi na to da je reč o visoko efikasnim rešenjima koja koriste LUT-ove za implementaciju logike za preskakanje nepotrebnih MAC operacija, što i jeste slučaj posebno sa CoNNa (C4) akceleratorom. Sa druge strane, ekstremne vrednosti LUT/DSP parametra imaju za rezultat diskvalifikaciju pomenutih arhitektura od upotrebe na kompaktnim FPGA čipovima što se jasno vidi u tabeli 10.

Sumirano, može se reći da se odnos LUT/DSP u slučaju Argus akceleratora kreće u granicama koje su dostupne na postojećim FPGA čipovima što značajno dobrinosi skalabilnosti i mogućnosti da akcelerator bude implementiran na širokoj paleti FPGA čipova. Takođe, može se izvesti zaključak da Argus CNN akcelerator uspeva da uposli značajan broj DSP blokova za obradu podataka, uz dodatno korišćenje odmerenog broj LUT-ova za preskakanje nepotrebnih MAC operacija kako bi se dodatno ubrzalo procesiranje.

8.2 Analiza performansi

Kako bi se što bolje uporedile performanse Argus arhitekture sa postojećim rešenjima, rezultati su predstavljeni na tri različita načina. Prvi deo poglavlja prikazuje isključivo rezultate Argus arhitekture u tri različite konfiguracije. U ovom delu je akcenat na analizi kako skaliranje akceleratora utiče na poboljšanje performansi. Naravno, idealno bi bilo da je to linearna zavisnost što je ipak, gotovo nemoguće postići. Zbog navedenog, izvršeno je lociranje uzroka za nelinearno skaliranje performansi sa veličinom akceleratora. Drugi deo evaluacije donosi

direktnu komparaciju performansi između Argusa i odabranih akceleratora za pet različitih CNN modela. Može se primetiti da Argus akcelerator postiže dobre rezultate za skoro svaki od CNN modela, ali ne uvek najbolje. Naravno, ideja predstavljene arhitekture nije postizanje najboljih performansi u smislu apsolutnih brojeva već maksimizacija performansi u kompaktnim FPGA čipovima. Stoga uz svaki CNN model sledi kratka analiza koja ističe značajne prednosti Argus arhitekture u odnosu na ostale, a koje se ne vide direktno iz prikazanih brojeva. Poslednja sekcija poglavlja pokazuje efikasnost arhitektura. Efikasnost je izražena kao poređenje ostvarenih performansi sa idealizovanim, koje se mogu izračunati na osnovu broja MAC jedinica i frekvencije rada akceleratora.

8.2.1 Analiza performansi Argus akceleratora u više različitih konfiguracija

Tabela 11 donosi ostvarene rezultate Argus akceleratora u tri različite konfiguracije i prilikom akceleracije pet različitih CNN modela. Ovih pet CNN-ova je odabrano kao reprezentativni uzorak mreža sa drastično različitim računskim zahtevima i arhitekturama generalno. Analiza performansi na većoj grupi različitih CNN modela daje najbolji uvid u realne prednosti i mane arhitektura, što često nije slučaj u publikacijama. Naime, postoje arhitekture koje pokazuju sjajne rezultate na određenom tipu CNN-ova, dok za druge imaju vrlo skromne rezultate. Razlog za ovako nešto leži u činjenici da su akceleratori razvijani sa idejom da akceleriraju određene mreže, a ne kao univerzalni. Najkompaktniji predstavnik odabranih CNN modela je MobileNet v1 koji je ujedno jedan od najšire korišćenih CNN modela u *embedded* aplikacijama. MobileNet v1 je testiran za dve konfiguracije, najveću dostupnu verziju (ulazna slika dimenzija 224x224 i *width multiplier* 1.0) i jednu kompaktnu (ulazna slika dimenzija 128x128 i *width multiplier* 0.5). Zbog arhitekture konvolucionih slojeva u kompaktnim mrežama, performanse na ovakvim modelima najčešće govore o robusnosti akceleratora u pogledu kontrole i brzog menjanja konteksta koji se procesira. Sa druge strane, mreže kao što su AlexNet i VGG-16 čini manji broj, značajno većih slojeva, te kod njih ne postoji bojazan od potencijalne degradacije performansi usled neefikasnosti arhitektura. Ipak, ovako velike mreže imaju izuzetne računske zahteve te su uključene u poređenje kako bi se jasno komparirale sposobnosti akceleratora da izvrše veliki broj računskih operacija. Da bi analiza bila kompletna uvrštena je i ResNet50 mreža. ResNet50 odlikuju raznoliki slojevi, uključujući i sloj sabiranja.

U terminima apsolutnih performansi, u tabeli 11 možemo videti da Argus akcelerator sa jednim CC blokom uspeva da obradi 49.6 slika u sekundi (engl. *Frames per second*, fps) ukoliko radi sa najvećom konfiguracijom MobileNet v1 i 11.1 fps kada je CNN model ResNet50. Ove performanse su uglavnom dovoljne za većinu embedded aplikacija [57]. Sa druge strane akcelerator je izuzetno kompaktan te ga je moguće implementirati na najmanjim dostupnim FPGA baziranim SoC-evima. Naravno, uvećavanjem broja CC modula, na četiri, akcelerator postiže bolje rezultate i to 185 fps na MobileNet-u, odnosno, 36.5 fps u slučaju ResNet50 mreže. Kada je reč o starijim arhitekturama CNN-ova kao što su VGG-16 i AlexNet, rezultati variraju od 3.4 fps do 11 fps za VGG-16, odnosno, odnosno, 26.9 fps do 82.8 fps za AlexNet.

Argus	**MobileNet v1 128x128 (fps)**	**MobileNet v1 224x224 (fps)**	**ResNet50 (fps)**	**VGG-16 (fps)**	**AlexNet (fps)**
1 CC, 1 DLP	352.7	49.6	11.1	3.4	26.89
2 CC, 1 DLP	572.8	94.8	20.1	5.8	49.8
4 CC, 1 DLP	1090	185	36.5	11.02	82.8

Tabela 11 Performanse Argus akceleratora u tri različite konfiguracije za pet različitih CNN modela.

Iz tabele 11 se može izvesti da se faktor ubrzanja između Argus arhitekture sa jednim i četiri CC modula kreće u granicama od 3.08 do 3.73. Konkretno, za ResNet50 je ostvareno ubrzanje od oko 3.29 puta dok je u slučaju VGG-16 i AlexNet-a to 3.51, odnosno 3.08. U slučaju kompaktnih mreža, zadržava se navedeno ubrzanje te je u slučaju MobileNet mreže sa ulazom veličine 128x128 faktor ubrzanja 3.09 dok je u slučaju iste mreže sa ulaznom slikom veličine 224x224 faktor 3.73 puta. Glavni uzrok nelinearnog poboljšanja performansi je propusna moć memorijskog interfejsa između akceleratora i eksterne memorije. U pogledu opterećenja DRAM kontrolera se ističe MobileNet v1, a posebno u kompaktnijoj konfiguraciji. Tipičan predstavnik problematičnih slojeva je prvi *Pointwise* sloj, ali i nekoliko slojeva koji ga slede. Naime, dubina IFM-a prvog *Pointwise* sloja je 16. Ovo znači da ukoliko je mreža orezana, svaki PE će završiti po jedno izračunavanje za dva takta, osam po osam tačaka IFM-a. Pošto CC inkorporira 32 PE-a, to znači da će u proseku, u svakom taktu biti spremno 16 izlaznih tačaka OFM-a za smeštanje u DRAM. Sa druge strane, iz opisa arhitekture se može uočiti da je nemoguće proslediti sve podatke, jer je propusna moć OS bloka jednaka osam podataka u jednom taktu. Čak i da se performanse OS modula dupliraju, neće se dobiti nikakvo poboljšanje, jer su performanse OS-a limitirane širinom AXI magistrale između Argus arhitekture i DRAM-a (128 bita). Treba napomenuti da povećani zahtevi ka DRAM kontroleru nisu samo na strani upisa u DRAM već i na strani čitanja iz DRAM-a. Kako bi se PE-ovi držali

uposleni, potrebno je u svakom taktu dopremati osam tačaka IFM-a. Da situacija bude još kompleksnija, postoje slojevi u kojima je nemoguće deliti IFM između više CC blokova putem Link modula. Tipičan primer su slojevi sa 32 ili manje kernela kao što je pomenuti *Pointwise* sloj. Konačnom analizom ovog sloja dolazi se do zahtevane propusne moći DRAM kontrolera od 15GB/s na strani čitanja i oko 30GB/s na strani upisa u DRAM. Navedeni podaci su u slučaju najsnažnije Argus konfiguracije sa 4 CC modula. Svakako ovakva propusna moć memorijskog sistema nije ni približno dostupna na današnjim pristupačnim *embedded* sistemima. Pošto su ovakvi slojevi više izuzetak nego pravilo, jasno je da nije potrebno akcelerator prilagođavati istim kako se ne bi narušio postojeći dobar balans između performansi i zahteva za hardverskim resursima.

8.2.2 Poređenje apsolutnih performansi sa prethodno publikovanim akceleratorima

Apsolutne performanse su prikazane u naredne četiri tabele, po jedna za svaki testirani CNN model. Pored neizostavnog imena akceleratora, tabela sadrži i najbitnije karakteristike akceleratora kao što je frekvencija rada, korišćeni broj bita za reprezentaciju podataka, zauzeće LUT-ova u slučaju da je akcelerator implementiran na FPGA čipu i broj MAC jedinica. Podsećanja radi, broj MAC jedinica odgovara korišćenom broju DSP blokova, ukoliko je akcelerator baziran na FPGA tehnologiji. Što se tiče performansi, one su prikazane za izvršavanje celokupne mreže, ali i za procesiranje isključivo konvolucionih slojeva, kao najzahtevnijih. Jedan od razloga za dodatni prikaz performansi samo na konvolucionim slojevima je činjenica da za određene akceleratore postoje samo rezultati za konvolucione slojeve. Drugi razlog je razdvajanje konvolucionih slojeva od izuzetno velikih, zastarelih arhitektura FC slojeva koji se sreću kod AlexNet i VGG-16 mreže. Bitno je napomenuti da značajna degradacija krajnjih performansi potiče upravo od ovih slojeva koji uopšte nisu zahtevni u računskom smislu već u memorijskom. Više o ovome će biti reči u analizi koja sledi.

AlexNet se smatra prvom uspešnom CNN arhitekturom te je zbog toga i najrasprostranjenija u evaluacijama performansi prethodno publikovanih akceleratora. Tabela 12 donosi sumirano rezultate sedam akceleratora uključujući i najveću verziju Argus akceleratora.

AlexNet	**Fpga ConvNet**	**NVDLA**	**Eyeriss v2**	**Thinker**	**Envision**	**Snowflake**	**Argus**
Frekvencija (MHz)	125	250	200	200	200	250	250
Aritmetička preciznost	16	16	8	8/16	4/8/16	16	16
LUT (samo FPGA)	218k	-	-	-	-	-	46k

Broj MAC jedinica		900	256	384	1024	512	256	272
Slika/sekund	Konvolucija	-	-	287.4*	-	47	100.5	254
	Ukupno	121.53	68.6	234.1*	254.3	-	-	82.8

*Rezultati su preračunati za DRAM kapaciteta 25GB/s. Sa navedenim performansama autori tvrde da imaju 16% lošije performanse odnosu na simulacije u kojima DRAM kontroler ima neograničene performanse.

Tabela 12 Poređenje performansi za AlexNet.

Posmatrano isključivo krajnje rezultate Argus CNN akceleratora, jasno se može uočiti da su za značajnu degradaciju performansi odgovorni pomenuti FC slojevi. Može se izračunati da prilikom procesiranja, akcelerator troši oko 67% ukupnog vremena na procesiranje FC slojeva. Zbog svoje veličine i nemogućnosti ponovnog korišćenja parametara neurona, kao što je to slučaj sa konvolucionim slojevima, većinu vremena akcelerator provodi učitavajući parametre neurona. Stoga DSP blokovi ostaju neuposleni, čekajući parametre. Kako bi se maksimalno iskoristili računski potencijali DSP blokova u ovim slojevima, neophodna je izuzetano velika propusna moć memorijskog sistema koja se kreće i do 200GB/s [32] što ni približno nije moguće pronaći u modernim *embedded* sistemima. Zbog navedenog, ali i činjenice da je ovakav pristup arhitekturi CNN-ova napušten, mnoge publikacije izbegavaju analizu i evaluaciju performansi nad ovim slojevima ili smatraju da se raznim algoritmima kompresije problem može donekle prevazići [32]. Kako bi se sprovelo pravedno poređenje sa Snowflake i Envision akceleratorima, uporedićemo samo performanse za konvolucione slojeve sa Argus akceleratorom. Može se primetiti da su rezultati Argus arhitekture 5 puta bolji od Envision-a i 2.5 puta bolji od Snowflake akceleratora. Analizom Snowflake i Envision akceleratora može se doći do zaključka da je najveća prednost Argusa u tome što procesira orezane mreže što smanjuje količinu računskih operacija dva puta, idealno. Na ovakav zaključak navodi činjenica da su ostale karakteristike ova dva akceleratora vrlo slične (frekvencija, broj DSP blokova, aritmetika).

U poređenju sa FpgaConvNet-om, Argus je 30% sporiji, ali treba primetiti da FpgaConvNet koristi 3.5, odnosno 4 puta više DSP blokova i LUT-ova respektivno. Ovakvo korišćenje resursa svakako nije proporcionalno dobicima u pogledu performansi. Pomenimo još jednom i da je zauzeće resursa FpgaConvNet-a zaista veliko te ga nije moguće instancirati u kompaktne FPGA SoC platforme. Što se tiče Thinker-a, Argus CNN akcelerator je oko 3 puta sporiji, ali koristi 4 puta manje MAC jedinica. Treba napomenuti da su rezultati za ovaj akcelerator prijavljeni u terminima GOPs (engl. *Giga Operations per second*) bez razmatranja propusne moći memorijskog sistema kao jednog od potencijalno limitirajućih faktora u realnoj aplikaciji. Zbog ovoga može se pretpostaviti da bi realne performanse bile lošije od prijavljenih.

Na kraju, komparacija Argusa sa Eyeriss v2 akceleratorom pokazuje da Argus arhitektura ima nešto lošije performanse kada se posmatraju samo konvolucioni, slojevi što je posledica značajno većeg faktora orezivanja AlexNet mreže koju su autori koristili prilikom evaluacije performansi Eyeriss v2. Treba napomenuti i to da autori Eyeriss-a računaju na memorijski interfejs propusne moći od najmanje 25GB/s što je više od 3 puta više nego što je propusna moć DRAM kontrolera korišćenog za evaluaciju performansi Argus CNN akceleratora. Uporedive performanse koje Argus arhitektura postiže za konvolucione slojeve uz značajno slabiji DRAM kontoler i manji faktor orezivanja su direktna zasluga keširanja IFM-a unutar IS modula, čime su navedene prednosti Eyeriss-a gotovo poništene. Ipak, u slučaju FC slojeva ne postoji keširanje koje bi bilo od koristi kada je veličina *batch*-a jednaka 1 te u ovom delu mreže Eyeriss v2 pravi značajnu razliku u poređenju sa Argus arhitekturom.

Drugi CNN model nad kojim je izvršeno poređenje performansi je VGG-16 i rezultati su prikazani u tabeli 13. Kao što se može videti, Argus akcelerator zauzima treće mesto kada je reč o performansama. Ne računajući akceleratora prikazane u radovima [36] i [37], Argus pored toga što je brži, zahteva barem 4 puta manje LUT-ova u poređenju sa FpgaConvNet i NullHop akceleratorom, kao i 2 odnosno 1.8 puta manje LUT-ova od Caffeine-a i NEURAghe-a. U poređenju sa bržim akceleratorima, Argus zahteva 4.2 puta manje DSP blokova i skoro 5.5 puta manje LUT-ova od arhitekture prikazane u [36], pri tome ostvarujući performanse koje su za samo 1.81 puta lošije od onih koje koje ima navedeni akcelerator. Rešenje publikovano u radu [37], kao i prethodno analiziran akcelerator ne spadaju u embeded orijentisane arhitekture, ali ostvaruju zavidne performanse. Na kraju, sve FPGA bazirane arhitekture zahtevaju toliko hardverskih resursa da ih je nemoguće implementirati na pristupačne FPGA uređaje.

VGG-16		**Fpga ConvNet**	**NullHop (FPGA)**	**NEURAghe**	**Caffeine**	**CoNNa**	[36]	[37]	[38]	**Argus**
Frekvencija (MHz)		125	60	140	200	140	200	200	150	250
Aritmetička preciznost		16	16	16	16	16	16	8/16	8	16
LUT (samo FPGA)		218k	229k	88k	100k	267k	252k	178.1k	13.4k	46k
Broj MAC jedinica		900	128	864	1058	256	1144	1350	220	272
Slika/sekund	Konvolucija	-	-	-	-	-	-	-	2.66	15.8
	Ukupno	4.01	0.44	5.52	8.67*	7.83	19.97	48	2.36	11.02

*Broj slika u sekundi koje akcelerator obradi su računati kao količnik prijavljenih performansi u GOP/s podeljen sa 30GOP/s koliko je potrebno za VGG-16. Autori nisu prikazali performanse u terminima fps.

Tabela 13 Poređenje performansi za VGG-16.

Postignute performanse na MobileNet v1 mreži su prikazane u tabeli 14. Za poređenje su odabrana dva različita rešenja, jedno usko specijalizovano za procesiranje mreža koje sadrže *Deptwise* konvolucione slojeve [47] i Eyeriss v2 kao predstavnik najkvalitetnijih akceleratora današnjice. Za razliku od AlexNet-a, MobileNet v1 spada u grupu modernih CNN arhitektura koje ne koriste velike FC slojeve kao izlazne. To u mnogome doprinosi da performanse Eyeriss-a i Argus akceleratora budu gotovo identične za razliku od AlexNet-a u kome je Eyeriss imao prednost. Primetimo i to da Eyeriss ostvaruje identične performanse sa više od 3.5 puta većom propusnom moći DRAM kontrolera (25GB/s). Takođe, vredno je pomena da su autori naveli da i ovako velika propusna moć DRAM kontrolera degradira performanse MobileNet-a za 24% u poređenju sa evaluacijom u kojoj se DRAM kontroler smatra idealnim. Za pretpostaviti je da bi dodatna redukcija kvaliteta kontrolera, na nivo dostupan u kompaktnim FPGA baziranim *embedded* sistemima, dodatno umanjio postignute rezultate Eyeriss akceleratora. Pored Eyeriss-a, sa kojim je poređenje izvršeno na MobileNet-u sa ulazima veličine 128x128 i *width multiplier*-om postavljenim na 0.5, analiza je urađena i za najveću konfiguraciju MobileNet-a. Ukoliko se Argus uporedi sa akceleratorom prikazanim u radu [47] može se uočiti da postiže oko 20% lošije performanse. Ipak, lako je uočiti da akcelerator prezentovan u radu [47], za ovakav rezultat koristi 12 puta više MAC jedinica što ga uveliko diskvalifikuje od upotrebe čak i na srednje kompleksnim FPGA SoC-evima. Na kraju analize sprovedene za MobileNet v1, vredi istaći da se korisan potencijal arhitekture najbolje vidi na modernim i/ili kompaktnim mrežama. Ovakve mreže su u potpunosti potisnule prethodno evaluirane VGG-16 i AlexNet CNN modele iz upotrebe. Ovakav trend će se nastaviti, te prilikom poređenja, za merodavna treba najviše uzeti MobileNet i ResNet50, dok AlexNet i VGG-16 treba da ilustruju samo performanse akceleratora koje su bliske teoretskim maksimumima.

MobileNet v1		**Eyeriss v2**	**Argus**	**Depthwise optimized accelerator**	**Argus**
Veličina slike		128	128	224	224
Width multiplier		0.5	0.5	1.0	1.0
Frekvencija (MHz)		200	250	150	250
Aritmetička preciznost		8	16	8	16
LUT (samo FPGA)		-	46k	121k	46k
Broj MAC jedinica		384	272	3283	272
Slika/sekund	Konvolucija	-	-	-	-
	Ukupno	1117*	1090	231.2	185

Tabela 14 Poređenje performansi za MobileNet v1.

Na kraju, tabela 15 donosi rezultate za ResNet50 CNN model. Poređenje je izvršeno sa NVDLA [56] i Snowflake akceleratorima. Isto kao i u slučaju AlexNet-a, Snowflake nudi podatke samo za konvolucione slojeve i u ovom slučaju Argus CNN akcelerator postiže 2.27 puta bolje performanse. Ukoliko se uporedi sa NVDLA akceleratorom u konfiguraciji identičnoj Argusu, sa frekvencijom rada od 500 MHz, Argus akcelerator postiže oko 25% bolje rezultate.

ResNet50		**NVDLA**	**NVDLA**	**Snowflake**	**Argus**
Max frequency (MHz)		250	500	250	250
Bit Precision		16	16	16	16
LUT (FPGA only)		-	-	-	46k
Num. of MAC's		256	256	256	272
Inference/sec	Conv only	-	-	17.7	40.2
	Overall	17.45	29.1	-	36.5

Tabela 15 Poređenje performansi za ResNet50.

Kao zaključan analize apsolutnih performanse, može se primetiti da prilikom procesiranja modernih CNN modela koji imaju široku upotrebu u embeded aplikacijama, Argus akcelerator ima najbolje ili uporedive performanse sa najboljim prethodno publikovanim embeded akceleratorima. Kada je reč o starijim CNN modelima, kao što su AlexNet i VGG-16, Argus arhitektura ima bolje performanse od većine akceleratora, posebno ako se uzme u obzir zauzeće hardverskih resursa. Ipak, od nekih arhitektura je značajno sporiji, međutim, iste uveliko izlaze iz okvira kompaktnih, embeded FPGA čipova.

8.2.3 Analiza efikasnosti akceleratora

Analiza efikasnosti akceleratora se može sprovesti na više načina. Da bi se jasno istakli kvaliteti razvijene arhitekture evaluacija performansi je sprovedena koristeći dve najšire prihvaćene metode. Prva prikazuje broj operacija koje akcelerator uspeva da postigne prema iskorišćenom hardverskom resursu. Recimo da ovaj pristup prikazuje gustinu performansi po utrosenom resursu. Druga metoda prikazuje efikasnost kao količnik ostvarenog broja operacija u sekundi i teoretskog maksimuma akceleratora, odnosno, FPGA čipa koji se koristi.

U publikacijama je najrasprostranjeniji parametar GMAC/s/DSP, čime se jasno pokazuje koliko su dobro uposleni DSP blokovi akceleratora. Razlog zašto je ovo najrasprostranjenija metrika proizilazi iz činjenice da su DSP-evi (MAC-ovi) zaduženi za izvršavanje računskih operacija te je to sprega koja navodi na mišljenje da je ovo najbitniji parametar. Ipak, zbog nekoliko puta pomenute činjenice da dostupnost ostalih resursa nije bezgranična, te da akcelerator nekada nije moguće implementirati zbog nebalansiranog korišćenja ostalih resursa, u tabeli 16 su

prikazane i performanse prema iskorišćenim LUT-ovima, odnosno, BRAM-ovima. Primetimo da je kompletna analiza povezana sa akceleratorima koji su implementirani na FPGA čipovima jer je poređenje ASIC arhitektura nemoguće u ovom kontekstu. Tabela 16 prikazuje gustinu performansi po resursu za sve FPGA arhitekture analizirane u ovoj doktorskoj disertaciji. Metrike kao što su GMAC/s/DSP i GMAC/s/LUT su uveliko rasprostranjene za razliku od GMAC/s/BRAM metrike. Podsećanja radi, jedan BRAM je ekvivalent memoriji veličine 36 kb i nalazi se na FPGA čipu. Takođe, treba reći da su performanse akceleratora sposobnih da procesiraju orezane CNN-ove, skalirane kako bi se moglo napraviti pravedno poređenje sa arhitekturama koje nisu u mogućnosti da obrađuju orezane mreže. Skaliranje je izvršeno tako što je broj slika u sekundi koje akcelerator obrađuje procesirajući orezane mreže (fps), pomnožen sa ukupnim brojem operacija potrebnim za klasifikaciju ukoliko je mreža neorezana. Pre analize rezultata, pomenimo i to da su zauzeća LUT-ova, u originalnim radovima, za neke od akceleratora, izražena u logičkim ćelijama. Da bi poređenje bilo fer, ovi podaci su translirani u LUT-ove prema smernicama koje daju proizvođači korišćenih FPGA čipova.

VGG16	DSP	LUT (k)	BRAM	GMAC/s/DSP	GMAC/s/k LUT	GMAC/s/BRAM
NEURAghe [33]	1728	200	640	0.28	2.43	0.76
Caffeine [31]	1058	391	784	0.25	2.66	0.68
CoNNa [35]	256	267	596	0.95	0.91	0.41
NullHop [25]	128	229	386	0.14	0.08	0.05
[38]	220	13.4	98	0.43	7.13	0.97
[36]	1144	252	456	0.27	1.23	0.68
[37]	1350	178.1	730	1.1	**8.34**	2.03
Argus 1CC	74	14	51	**1.42**	7.51	**2.06**
Argus 4CC	272	46	202.5	1.25	7.41	1.68

Tabela 16 Gustina performansi prema iskorišćenim hardverskim resursima.

Kao što se u tabeli može videti, Argus akcelerator sa jednim CC blokom ima 30% bolje performanse od prve sledeće arhitekture predstavljene u radu [37]. Naravno, Argus sa jednim CC modulom ne ostvaruje visoke performanse kao akcelerator iz rada [37], ali možemo primetiti da je i Argus sa četiri CC bloka i dalje najbolji od svih navedenih arhitektura. Jedina arhitektura koja može da se pohvali visokom gustinom performansi po iskorišćenom DSP bloku je CoNNa C4, dok su ostali od 2 do 10 puta lošiji od Argusa.

Kada je reč o gustini performansi po LUT-u, Argus CNN akcelerator je drugi sa 10% lošijim performansama od najbolje arhitekture prikazane u radu [37]. Ono što se dodatno može uočiti je da vrednost parametra GMAC/s/kLUT gotovo i da ne opada čak i kada se broj CC modula

poveća sa 1 na 4. I u najvećoj verziji, Argus arhitektura zadržava drugo mesto kada je gustina performansi po LUT-u u pitanju. Analizirajući performanse po iskorišćenom BRAM-u, može se reći da da Argus akcelerator sa jednim CC blokom deli prvo mesto sa arhitekturom prikazanom u radu [37]. U poređenju sa ostalima, razlika je još veća i kreće se od 2 do 45 puta u korist Argusa. Kao zaključak, može se reći da Argus i akcelerator predstavljen u [37] imaju ubedljivo najbolje ukupne rezultate kada su navedene metrike u pitanju.

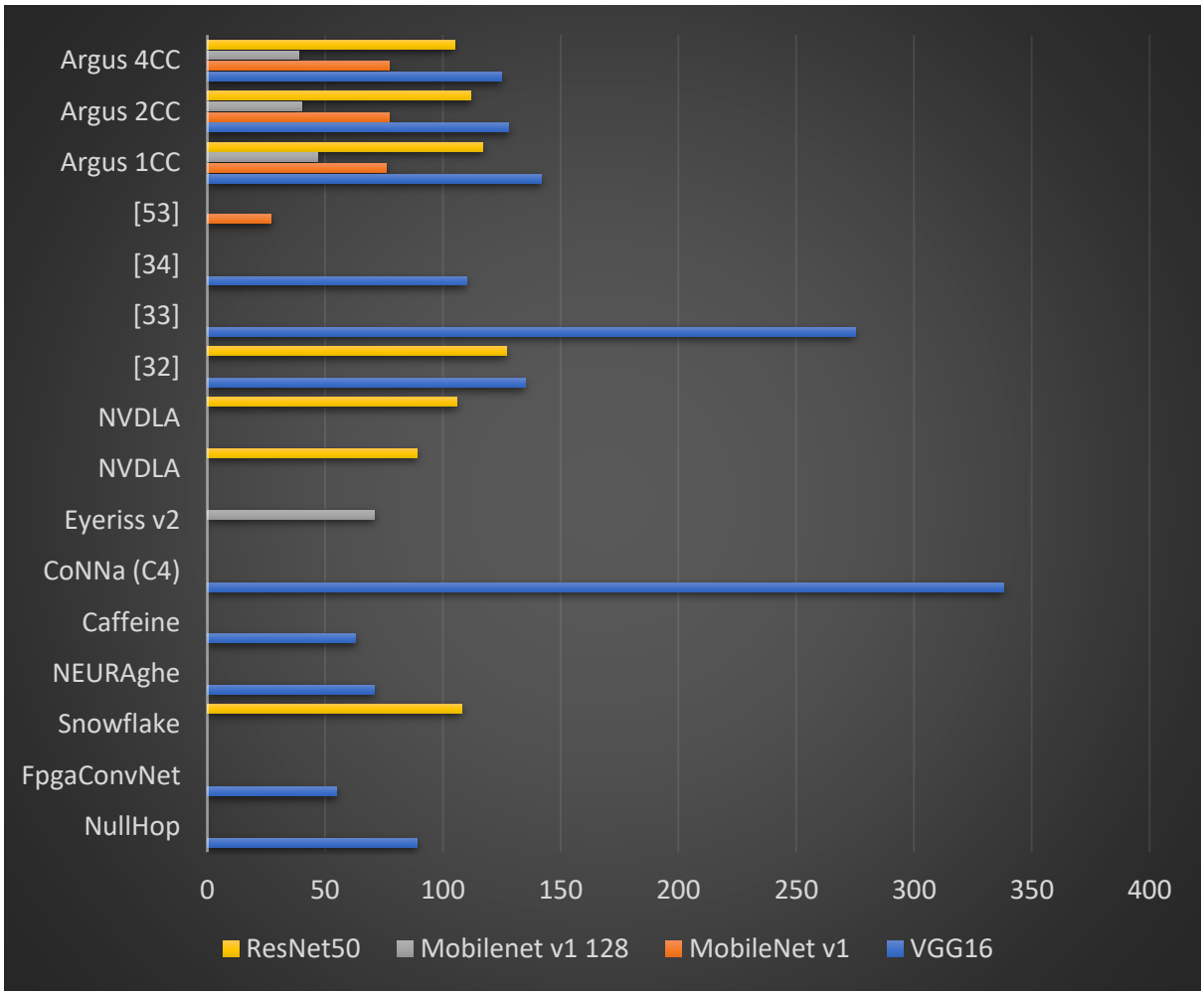
Na kraju, treba reći i da zavisnosti od algoritma koji se akcelerira, nekada neće biti moguće uravnoteženo trošiti resurse zbog prirode algoritma. Razmišljajući u tom pravcu može se dovesti u pitanje da li je ujednačeno trošenje resursa zaista bitno prilikom akceleriranja CNN-ova? Odgovor na ovo pitanje je moguće dati samo posmatrajući sva tri analizirana parametra zajedno. Iz tabele 16 se jasno vidi da je efikasnost Argus CNN akceleratora u vrhu za svaki od parametara. Visoku efikasnost prate i visoke performanse u kompaktnim FPGA čipovima što je argument koji potkrepljuje hipotezu da je u ovoj aplikaciji ravnomerno trošenje resursa bitno.

8.2.3.1 Poređenje efikasnosti kao količnik teoretskog maksimuma čipa i dobijenih rezultata

Efikasnost arhitektura je komparirana na dva načina. Prvo poređenje je izvedeno tako što je teoretski maksimum izračunat kao broj korišćenih MAC jedinica pomnožen sa frekvencijom na kojoj čip radi. Ova vrednost će biti imenilac dok će broilac biti ostvarene performanse akceleratora iskazane u GMAC/s. Drugi pristup jasno oslika ciljeve ove doktorske disertacije, a to je kreiranje arhitekture koja izuzetno efikasno koristi raspoložive performanse kompaktnih FPGA čipova. Kod ovog pristupa teoretske performanse su prikazane kao maksimum performansi FPGA čipa koji je proizvođač dao. Dakle, ukupne teoretske performanse koje uključuju iskorišćeni i neiskorišćeni deo čipa za razliku od prve metrike koja se fokusira isključivo na iskorišćene resurse. Da bi poređenje bilo pravedno, uzeti su najmanji čipovi u koje akceleratori mogu da se instanciraju.

Prvi pristup poredi arhitekture na široko prihvaćen način, posebno u ASIC tehnologijama. Prilikom implementacije akceleratora u nekoj od ASIC tehnologija, postoje značajno manja ograničenja u pogledu raspoloživosti različitih resursa. Takođe, množači se najčešće uzimaju za najkompleksnije komponente te je obično samo njihov broj iskazan, smatrajući da logičkih kola gotovo uvek ima dovoljno. Eventualno, iskazuje se i količina memorije koju akceleratori koriste. Zbog toga što broj logičkih kapija koje se ne koriste za MAC operacije često nije

ograničenje, ovakav pristup ima valjanost u ASIC svetu. Kako bi rezultati prikazani na grafikonu 1 bili korektno interpretirani, sledi primer kako je računata efikasnost u prvoj varijanti. Uzmimo najboljeg kandidata, a to je CoNNa (C4). Efikasnost akceleratora je iskazana u procentima i u slučaju CoNNa (C4) ona iznosi skoro 350% za VGG-16. Da bismo došli do ovog broja, potrebno je znati frekvenciju na kojoj radi CoNNa-a koja iznosi 140 MHz. Uz frekvenciju, potrebno je znati da akcelerator koristi 256 MAC jedinica (DSP blokova). Kada se pomnože ove dve vrednosti dolazi se do teoretskog maksimuma koji je moguće postići, ukoliko su MAC jedinice zauzete u svakom taktu, to jest, 100% vremena procesiranja. Maksimum MAC operacija koji CoNNa može da izvrši je skoro 36 GMAC/s. Izračunat broj GMAC/s, koji CoNNa ostvaruje kada se fps pomnoži sa potrebnim brojem GMAC-ova za procesiranje jedne slike, je oko 121 GMAC/s. Deljenjem 121 GMAC/s sa ~36 GMAC/s dobija se efikasnost koja je u granicama nešto ispod 350%.



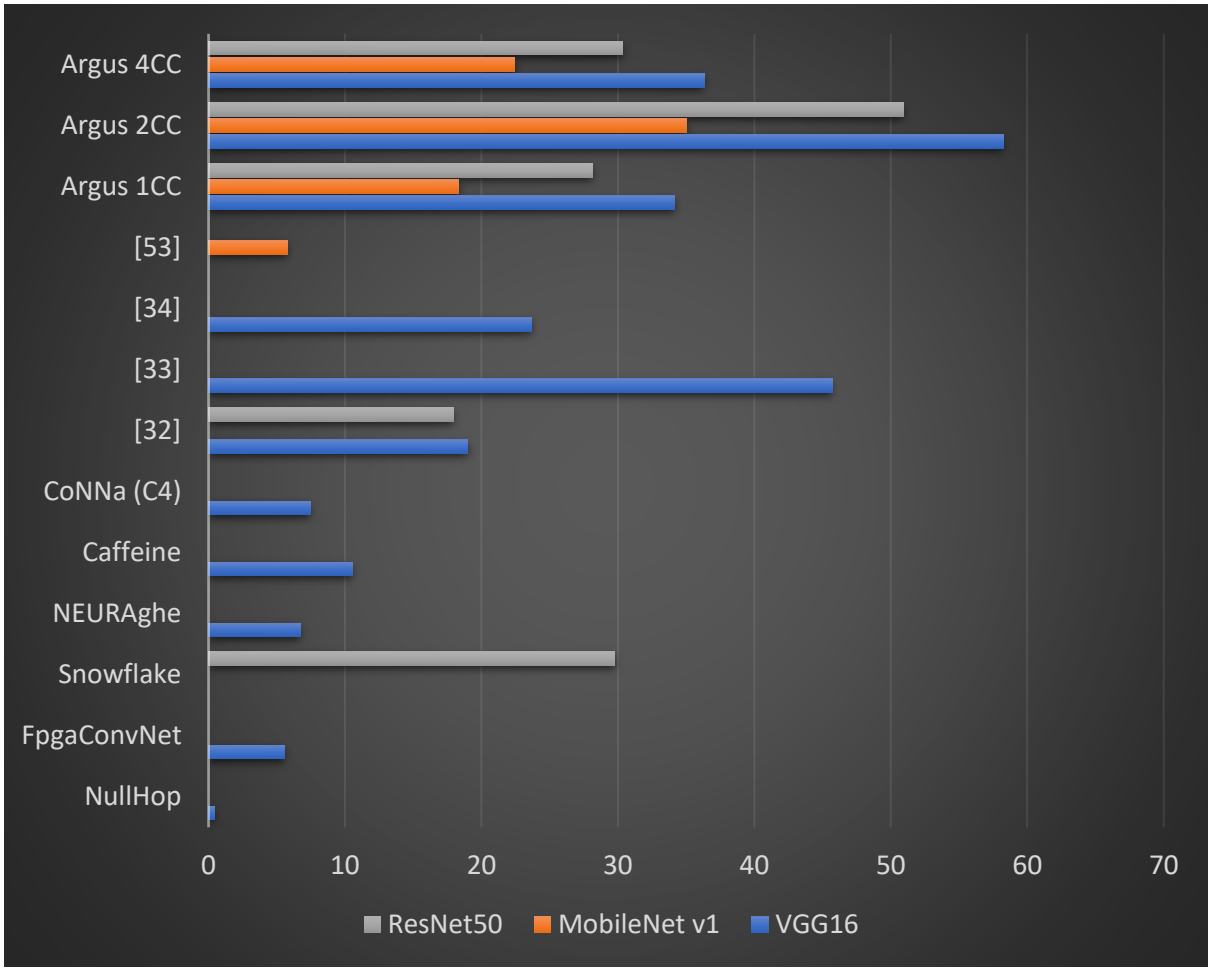
Grafikon 1 Efikasnost arhitektura uopšte, bez uticaja raspoloživih resursa.

Analizom grafikona 1 se može primetiti da arhitektura predstavljena u radu [37] takođe postiže visoku efikasnost, okvirno 275%. Kao i u slučaju CoNNa-e, i ova arhitektura ima mogućnost preskakanja nepotrebnih operacija čemu duguje visoke performanse. Sledeći je Argus CNN akcelerator sa jednim CC modulom koji ima efikasnost od skoro 150%. Ostali akceleratori, bilo da su ASIC ili FPGA bazirani, imaju lošije performanse na VGG-16 CNN-u. Što se tiče MobileNet v1 mreže u svojoj najvećoj konfiguraciji, Argus arhitektura ima najbolje performanse, dok nešto lošije u odnosu na konkurenciju ima kada je u pitanju MobileNet v1 sa ulazom veličine 128x128 piksela. U slučaju ResNet50, performanse su uporedive ili bolje od ostalih. Tako akcelerator prikazan u radu [36] ima marginalno bolje performanse dok ostali zaostaju, čak i kada se uporedi sa Snowflake-om koji akcelerira samo konvolucione slojeve.

Na osnovu prethodne analize, može se zaključiti da se arhitektura Argusa, kada se izuzmu ograničenja resursa, ima rezultate koji se nalaze u boljoj polovini od skupa koji je testiran u ovoj doktorskoj disertaciji. Dodatno, treba primetiti da ostale arhitekture oskudevaju u prikazu performansi za raznolike CNN-ove. Ova činjenica može navesti na više zaključaka od kojih su neki da je arhitektura razvijena imajući na umu jednu ili dve mreže te nisu u stanju da ostvare uporedive performanse na drugim. Drugi mogući razlog je da su izuzetno komplikovane te da mapiranje drugih CNN-ova zahteva značajno vreme potrebno da se prilagode konfiguracioni parametri. Svakako, za ozbiljnu analizu upotrebljivosti akceleratora u ovom trenutku, nije dovoljno predstaviti performanse za AlexNet i/ili VGG-16 jer su to arhitekture CNN-ova uveliko prevaziđenje, posebno u *embedded* svetu.

Posmatrajući isključivo grafikon 1, nije moguće dobiti sliku o tome koliko bi određeni akcelerator bio dobar u slučaju primene na u današnje vreme dostupnim FPGA sistemima. Ovo se jasno može videti na grafikonu 2, koji donosi rezultate efikasnosti arhitektura generisanih prema drugom pristupu pomenutom na početku ove sekcije. Detaljnije, za teoretske maksimume su uzeti maksimumi FPGA čipova na kojima su implementirani akceleratori. Do ovih brojeva se dolazi kada se ukupan broj DSP-eva koji postoje na odabranoj FPGA komponenti pomnoži sa maksimalnom frekvencijom na kojoj isti mogu da rade. Za maksimalnu frekvenciju je odabran brzinski indeks -1 koji donosi frekvencije od 645 MHz za svaki DSP [54]. Na primer, teoretske performanse Xilinx Ultrascale+ zu3 FPGA čipa u konfiguraciji 16-bitne aritmetike se mogu izračunati kao 645 MHz x 360 DSP blokova što iznosi oko 232 GMAC/s. Za svaki od čipova iz tabele 10 je na isti način moguće izračunati teoretske

maksimume. Sledeći korak je odrediti u koje FPGA čipove je moguće smestiti odgovarajući akcelerator. Takođe, tabela 10 sadrži ove rezultate. Na kraju je potrebno samo evaluirane performanse podeliti sa odgovarajućim performansama čipa na kome je moguće implementirati datu arhitekturu. Napomenimo i to da je u svrhe ove analize jedino merodavno porediti FPGA akceleratore.



Grafikon 2 Efikasnost FPGA arhitektura računata prema stvarnom FPGA čipu u koje ih je moguće instancionirati.

Grafikon 2 jasno pokazuje da Argus akcelerator postiže najbolje performanse prema raspoloživim performansama čipa. Podsećanja radi, Argus arhitektura sa jednim i dva CC bloka može da se implementira na Xilinx Ultrascale+ zu2 čipu. Pošto su i konfiguracija sa jednim i sa dva CC modula instancirane na istom FPGA čipu jasno je da će performanse konfiguracije sa dva CC bloka biti skoro dvostruko bolje. Prosto, imenilac, to jest, teoretski limit je identičan, a performanse akceleratora su skoro dvostruko bolje. Sa grafikona 2 možemo uočiti da Argus arhitektura uspeva da iskoristi skoro 60% teoretskog maksimuma zu2 čipa procesirajući VGG-16. Rezultati za ResNet50 su nešto bolji od 50%, dok je efikasnost prilikom procesiranja

MobileNet v1, u najvećoj konfiguraciji, oko 35%. Primetimo da svi ostali imaju značajno manje iskorišćenje performansi čipova na kojima su implementirani. Za visoku iskorišćenost performansi čipova su u najvećoj meri zaslužne:

1. Procesiranje orezanih mreža što smanjuje potreban broj operacija za oko 50% u odnosu na neorezanu mrežu.
2. Algoritam za orezivanje prilagođen resursima FPGA čipova koji ne zahteva veliki broj LUT-ova da bi se preskakanje nula implementiralo. Ovo dovodi do mogućnosti da se instancira akcelerator sa većim brojem MAC jedinica, to jest, da se iskoriste dostupni računarski resursi.
3. Arhitektura koja je u mogućnosti da na 250 MHz doprema podatke dvostruko širim magistralama ka DSP blokovima, kako bi DSP-evi bili uposleni na dvostruko većoj frekvenciji (500 MHz).

Prve dve stavke su do sada analizirane dok je uposlenost DSP-eva na visokim frekvencijama bila skrajnuta. Jasno je da frekvencija uz broj DSP-eva igra podjednako važnu ulogu. Frekvencija je stavka koja se u mnogome zanemaruje u svim do sada objavljenim radovima. Kada se uzme u obzir maksimalna frekvencija na kojoj ovi blokovi mogu da rade i frekvencija na kojoj se koriste, jasno je da velika većina arhitektura ne koristi ni 50% mogućnosti DSP-eva jer rade na učestanostima od 150 do 250 MHz. U poređenju sa 645 MHz, ove frekvencije su i do 4 puta niže te se može uočiti da se DSP blokovi ne iskorišćavaju u odgovarajućoj meri. Gledajući isključivo performanse, veliki broj arhitektura koje procesiraju orezane CNN-ove, ne postižu bolje rezulta nego hipotetička, naivna, arhitektura koja bi procesirala neorezane CNN-ove na visokim učestanostima sa velikim brojem iskorišćenih DSP blokova. Argus akcelerator i ovde pokazuje dobre strane jer blokovi rade na frekvenciji mnogo bližoj maksimalnoj nego što je to slučaj kod drugih.

Na kraju, treba reći da je Argus arhitektura u konfiguraciji sa 4 CC modula, konzervativno, implementiran na čipu oznake zu4. Uz vrlo malu redukciju keš memorije unutar IS modula, ili prebacivanje dela memorije u distribuirani RAM, moguće je isti smestiti i u zu3 čip te bi performanse Argusa sa 4 CC bloka bile još bolje od Argusa sa 2 CC-a. Ovakav pristup je, ipak, odabran kako bi olakšao posao kompajlerima koji mapiraju RTL na FPGA jer je poznato da u ograničenim uslovima, kakvi vladaju na FPGA čipovima, potrebno ostaviti određenu rezervu kako bi se implementacija izvršila na optimalan način i postigli željeni rezultati. Takođe, treba

reći da je u čip oznake zu2 moguće smestiti Argus akcelerator sa 3 CC modula kako bi se postigla još bolja efikasnost. Očekivati je da u ovoj konfiguraciji na navedenom čipu, Argus CNN akcelerator ostvari efikasnost od oko 80% za VGG-16 CNN. Ipak, pošto u konfiguraciji sa 3 CC bloka nije ni implementiran, grafikon 2 ne sadrži analizu performansi za ovu varijantu.

9 Zaključak

U ovoj doktorskoj disertaciji je prikazan novi algoritam za orezivanje CNN mreža i kompletan akcelerator, imena Argus, namenjen za upotrebu u FPGA baziranim, embeded sistemima. Cilj disertacije je da se pokaže da je paralelnim razvojem algoritma za orezivanje i arhitekture akceleratora moguće kreirati jezgro koje ima bolje performanse po ukupno raspoloživim resursima na FPGA čipovima od prethodno publikovanih rešenja. Algoritam je pažljivo skrojen imajući na umu raspoložive resurse u modernim FPGA čipovima. Kao baza je uzet postojeći algoritam koji rešava određene probleme do sada poznatih pristupa orezivanju mreža. Isti je kreiran za ASIC implementaciju te je pretrpeo poboljšanja kako bi se učinio pogodnim za upotrebu u FPGA baziranim akceleratorima. Logika za preskakanje nepotrebnih operacija je ovim poboljšanjem smanjena na polovinu originalne veličine. Dodatno, uvedeno je klasterovanje sličnih kernela kako bi se dodatno, sa faktorom 2, smanjila navedena logika. Da bi se prezentovao kvalitet algoritma, kreiran je kompletan akcelerator za procesiranje kako neorezanih tako i mreža orezanih pomoću predstavljenog algoritma. Akcelerator koristi sve navedene benefite algoritma te zbog toga balansirano koristi raspoložive hardverske resurse, a sve u cilju poboljšanja performansi na kompaktnim FPGA čipovima. Uz navedeno, korišćene su i napredne tehnike implementacije, kao što je *multi-pumping*.

Evaluacija je izvršena zasebno, algoritam za orezivanje je evaluiran na najčešće korišćenim CNN-ovima, dok je akcelerator upoređen sa prethodno publikovanim rešenjima na više relevantnih načina. Kako bi se demonstrirale performanse algoritma, orezana su tri arhitekturno značajno različita CNN modela, MobileNet v1, ResNet50 i VGG-16. Svaki od ovih modela ima neku specifičnost kao što je kompaktnost, raznolikost slojeva i visoke performanse i široka rasprostranjenost, respektivno. Orezivanje je sprovedeno na prethodno treniranim modelima na ImageNet skupu podataka, dok je za testiranje korišćen validacioni skup ImageNet-a. Rezultat je takav da, ukoliko i postoji degradacija kvaliteta CNN modela posle orezivanja, ona ne prelazi prihvatljive granice.

U nastavku disertacije je prikazan tok razvoja arhitekture polazeći od algoritma za procesiranje konvolucionih slojeva. Korak po korak su uvođene optimizacije, najčešće raspetljavanja petlji. Dalje, optimizovani algoritam za izračunavanje konvolucionih slojeva je transliran u hardversku implementaciju. Svaki hardverski blok je posebno prikazan u

zasebnom poglavlju sa akcentom na bitne detalje i tokove podataka kako bi se priložilo dovoljno informacija za reimpelmentaciju kompletne arhitekture. Naravno, posle implementacije izvršena je i funkcionalna verifikacija kojom je proverena korektnost implementacije akceleratora, ali i pretpostavke da u arhitekturi nema uskih grla ili nekih drugih tipičnih mana.

Na kraju su prikazani rezultati implementacije akceleratora u terminima zauzeća resursa kao i ostvarenih performansi. Sprovedena je analiza poboljšanja performansi sa skaliranjem akceleratora. Argus je konfigurisan u tri različite varijante. Prva uključuje jedno jezgro za procesiranje konvolucionih i jedno za procesiranje ostalih slojeva. Druga ima dve instance konvolucionog jezgra dok treća inkorporira četiri. Pokazano je da se performanse dobro skaliraju, ne linearno, ali približno linearno posebno u slučaju MobileNet v1, gde su performanse Argusa sa četiri konvoluciona jezgra 3.73 puta bolje od Argusa sa jednim konvolucionim jezgrom.

Po evaluaciji dobijenih rezultata, prikazano je detaljno poređenje Argusa sa prethodno publikovanim rešenjima. U prvom delu poređenja može se videti da je projektovani akcelerator kompaktan i da ga je moguće instancirati u pristupačnim FPGA čipova što nije slučaj sa velikom većinom ostalih. Zatim je usledi analiza apsolutnih performansi na osnovu koje se može zaključiti da su rezultati Argusa uporedivi sa najboljim arhitekturama ili bolji od svih kompaktnih akceleratora prethodno objavljenih. Ukoliko su performanse Argusa lošije od određenih akceleratora, jasno je naznačeno na račun čega je to postignuto. Ipak, metrika apsolutnih brojeva nije nešto što ističe kvalitete prikazane arhitekture. Da bi se pokazalo kako apsolutni brojevi ne govore sve o kvalitetu arhitekture, posebno ako ona treba da bude implementirana u embeded sistemima, data je tabela u kojoj su prikazane performanse iskazane u GOP/s po jedinici iskorišćenog resursa (DSP, LUT, BRAM). U ovom razmatranju možemo videti da se Argus nalazi u vrhu liste, zauzimajući prvo i drugo mesto u ovim metrikama. Na posletku, prikazana je i efikasnost arhitektura kao količnik broja ostvarenih operacija u sekundi i teoretskog maksimuma za odabranu FPGA komponentu. Ova analiza je sprovedena na dva načina, prvi koji evaluira kvalitet arhitektura uopšte i drugi koji pokazuje koliko od teoretski raspoloživih performansi na stvarnim FPGA čipovima, arhitekture koriste. Druga od dve analize jasno pokazuje prednosti Argusa u odnosu na sve nama poznate publikovane arhitekture. Argus uspeva da iskoristi više od 50% raspoloživih performansi FPGA

čipa u slučaju ResNet50 CNN-a, i skoro 60% u slučaju VGG-16 mreže. Ovaj rezultat je više od 10% bolji u slučaju VGG-16 u odnosu na prvog pratioca odnosno više od 20% bolje u slučaju ResNet50 mreže.

Reference

[1] Stephen Marsland, Machine Learning: An Algorithmic Perspective, Second Edition, Chapman & Hall/CRC, 2014.

[2] Richard Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks," u *International Conference on Neural Information Processing Systems - Volume 1*, New York, 2012.

[4] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, Clara I. Sanchez, „A survey on deep learning in medical image analysis," *Medical image analysis,* t. 42, pp. 60-88, 2017.

[5] Andre Luckow, Matthew Cook, Nathan Ashcraft, Edwin Weill, Emil Djerekarov, Bennie Vorster, „Deep learning in the automotive industry: Applications and tools," u *IEEE International Conference on Big Data (Big Data)*, 2016.

[6] Karen Simonyan, Andrew Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition," u *arXiv:1409.1556*, 2014.

[7] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz, „Pruning Convolutional Neural Networks for Resource Efficient Inference," u *arXiv:1611.06440*, 2017.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijum Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision," arXiv:1704.04861, 2017.

[9] Hyeon-Ju Kang, „Accelerator-Aware Pruning for Convolutional Neural Networks," *IEEE Transactions on Circuits and Systems for Video Technology,* t. 30, pp. 2093-2103, 2020.

[10] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, „Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach," u *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[11] Damjan Rakanovic, Andrea Erdeljan, Vuk Vranjkovic, Bogdan Vukobratovic, Predrag Teodorovic and Rastislav Struharik, „Reducing off-chip memory traffic in deep CNNs using stick buffer cache," u *2017 25th Telecommunication Forum (TELFOR)*, Belgrad, Serbia, 2017.

[12] Vuk Vranjkovic, Rastislav Struharik, „Stick Buffer Cache v2: Improved Input Feature Map Cache for Reducing off-chip Memory Traffic in CNN Accelerators," u *27th Telecommunications Forum (TELFOR)*, Belgrad, Serbia, 2019.

[13] Mariette Awad, Rahul Khanna, Machine Learning. In: Efficient Learning Machines, Apress, Berkeley, CA, 2015.

[14] Vranjković Vuk, Doktorska disertacija - Rekonfigurabilne arhitekture za hardversku akceleraciju prediktivnih modela mašinskog učenja, Novi Sad: Fakultet tehničkih nauka, 2015.

[15] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, „Deep learning," *Nature,* t. 512, pp. 436-444, 2015.

[16] Yann LeCun, Patrick Haffner, Léon Bottou, Yoshua Bengio, Object Recognition with Gradient-Based Learning, t. 1681, Berlin: Springer, Berlin, Heidelberg, 1999.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, „ImageNet Large Scale Visual Recognition Challenge," arXiv:1409.0575.

[18] Sergey Ioffe, Christian Szegedy, „Batch Normalization: Accelerating Deep Network Training by ReducingInternal Covariate Shif," u *in Proceedings of the32ndInternational Conference on MachineLearning*, Lille, France, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, „Deep Residual Learning for Image Recognition," u *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[20] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter, „Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," u *arXiv:1511.07289*, 2016.

[21] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, „Rethinking the Inception Architecture for Computer Vision," u *arXiv:1512.00567*, 2015.

[22] Chen Yu-Hsin, Emer Joel, Sze Vivienne, „Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,“ u *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, Korea (South), 2016.

[23] Chen Yu-Hsin, Yang Tien-Ju, Emer Joel, Sze Vivienne, „Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,“ *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* t. 9, br. 2, pp. 292-308, 2019.

[24] Zhang Shijin, Du Zidong, Zhang Lei, Lan Huiying, Liu Shaoli, Li Ling, Guo Qi, Chen Tianshi, Chen Yunji, „Cambricon-X: An accelerator for sparse neural networks,“ u *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Taipei, Taiwan, 2016.

[25] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, Tobi Delbruck, „NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps,“ *IEEE Transactions on Neural Networks and Learning Systems,* t. 30, br. 3, pp. 644-656, 2019.

[26] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, Olivier Temam, „DaDianNao: A Machine-Learning Supercomputer,“ u *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, UK, 2014.

[27] Jingyang Zhu, Jingbo Jiang, Xizi Chen, Chi-Ying Tsui, „SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity,“ u *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2018.

[28] Moons, Bert; Uytterhoeven, Roel; Dehaene, Wim; Verhelst, Marian, „Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI,“ u *IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, 2017.

[29] Yin Shouyi, Ouyang Peng, Tang Shibin, Tu Fengbin, Li Xiudong, Liu Leibo, Wei Shaojun, „A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications,“ u *2017 Symposium on VLSI Circuits*, Kyoto, Japan, 2017.

[30] Lee Jinmook, Kim Changhyeon, Kang Sanghoon, Shin Dongjoo, Kim Sangyeob, Yoo, Hoi-Jun, „UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," u *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, San Francisco, CA, USA, 2018.

[31] C. Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan and Jason Cong, „Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," u *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, USA, 2016.

[32] Vinayak Gokhale, Aliasger Zaidy, Andre Xian Ming Chang, Eugenio Culurciello, „Snowflake: A Model Agnostic Accelerator for Deep Convolutional Neural Networks," u *arXiv:1708.02579*, 2017.

[33] Paolo Meloni, Alessandro Capotondi, Gianfranco Deriu, Michele Brian, Francesco Conti, Davide Rossi, Luigi Raffo, Luca Benini, „NEURAghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs," arXiv:1712.00994, 2017.

[34] Stylianos I. Venieris, Christos-Savvas Bouganis, „Latency-Driven Design for FPGA-based," http://cas.ee.ic.ac.uk/people/sv1310/papers/sv2017fpl.pdf, 2017.

[35] Rastislav Struharik, Bogdan Vukobratović, Andrea Erdeljan, Damjan Rakanović, „CoNNA – Compressed CNN Hardware Accelerator," u *2018 21st Euromicro Conference on Digital System Design (DSD)*, Prague, Czech Republic, 2018.

[36] Lu Liqiang, Xie Jiaming, Huang Ruirui, Zhang Jiansong, Lin Wei, Liang Yun, „An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs," u *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, San Diego, CA, USA, 2019.

[37] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, Haibin Shen, „An Efficient Hardware Accelerator for Structured Sparse Convolutional Neural Networks on FPGAs," arXiv:2001.01955, 2020.

[38] Fanny Spagnolo, Stefania Perri,Fabio Frustaci, Pasquale Corsonello, „Energy-Efficient Architecture for CNNs Inference on Heterogeneous FPGA," *Journal of Low Power Electronics and Applications,* 2019.

[39] Song Han, Jeff Pool, John Tran, William J. Dally, „Learning both weights and connections for Efficient Neural Networks,“ u *NIPS*, 2015.

[40] J.A.Hartigan, M.A. Wong, „A K-Means Clustering Algorithm,“ *Journal of the Royal Statictical Society,* t. 28, br. 1, pp. 100-108, 197.

[41] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, J&g Sander, „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databeses with Noise,“ u *KDD-96*, 1996.

[42] B. W. Kernighan, S. Lin, „An Efficient Heuristic Procedure for Partitioning Graphs,“ *The Bell System Technical Journal,* t. 49, br. 2, pp. 291-307, 1970.

[43] [Na mreži]. Available: http://www.python.org.

[44] Chollet, F. & others, „Keras,“ https://github.com/fchollet/keras, 2015.

[45] Philip Colangelo, Nasibeh Nasiri, Eriko Nurvitadhi, Asit Mishra, Martin Margala, Kevin Nealis, „Exploration of low numeric precision deep learning inference,“ u *IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.

[46] Leslie N. Smith, „Cyclical Learning Rates for Training Neural Networks,“ arXiv:1506.01186, Washington, 2015.

[47] Ruizhe Zhao, Xinyu Niu, Wayne Luk, „Automatic,“ u *Proceedings of the 2018 ACM/SIGDA International Symposium on*, 2018.

[48] ARM, „AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite,“ 2011. [Na mreži]. Available: https://developer.arm.com/documentation/ihi0022/e/. [Poslednji pristup 11 Avgust 2021].

[49] „AXI DataMover v5.1 LogiCORE IP Product Guide,“ Xilinx, 5 April 2017. [Na mreži]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_datamover/v5_1/pg022_axi_datamover.pdf. [Poslednji pristup 11 Avgust 2021].

[50] Xilinx, „7 Series FPGAs Memory Resources - User Guide,“ 3 July 2019. [Na mreži]. Available:

https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf. [Poslednji pristup 11 Avgust 2021].

[51] Xilinx, „UltraScale Architecture DSP Slice - User Guide,“ 9 September 2020. [Na mreži]. Available: https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf. [Poslednji pristup 28 Avgust 2021].

[52] Reed P. Tidwell, „Alpha Blending Two Data Streams Using a DSP48 DDR Technique,“ 31 March 2005. [Na mreži]. Available: http://notes-application.abcelectronique.com/077/77-43047.pdf. [Poslednji pristup 28 Avgust 2021].

[53] Bajaj Ronak, Suhaib A. Fahmy, „Multipumping flexible DSP blocks for resource reduction on Xilinx FPGAs,“ u *IEEE Transactions on*, 2017.

[54] Xilinx, „Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics,“ 23 June 2021. [Na mreži]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds925-zynq-ultrascale-plus.pdf. [Poslednji pristup 29 Avgust 2021].

[55] Xilinx, „UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices,“ 14 Jun 2016. [Na mreži]. Available: https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf. [Poslednji pristup 18 Decembar 2021].

[56] NVIDIA, „NVDLA,“ NVIDIA, [Na mreži]. Available: http://nvdla.org. [Poslednji pristup 23 Maj 2020].

[57] Seyed Yahya Nikouei, Yu Chen, Sejun Song, Ronghua Xu, Baek-Young Choi, and Timothy R. Faughnan, „Real Time Human Detection as an Edge Service Enabled by a Lightweight CNN,“ u *International Conference on Edge Computing*, 2018.

[58] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams , „Learning representations by back-propagating errors,“ *Nature,* t. 323, pp. 533-536, 1986.

План третмана података

Назив пројекта/истраживања
Хардверска акцелерација конволуционих неуронских мрежа у ембедед системима
Назив институције/институција у оквиру којих се спроводи истраживање
Факултет техничких наука Универзитета у Новом Саду
Назив програма у оквиру ког се реализује истраживање
1. Опис података

1.1 Врста студије

Укратко описати тип студије у оквиру које се подаци прикупљају

____У овој студији нису прикупљани подаци. Истраживање је базирано на јавно доступним подацима чији су извори наведени у литератури дисертације._____

1.2 Врсте података

а) квантитативни

б) квалитативни

1.3. Начин прикупљања података

а) анкете, упитници, тестови

б) клиничке процене, медицински записи, електронски здравствени записи

в) генотипови: навести врсту _____

г) административни подаци: навести врсту _____

д) узорци ткива: навести врсту_____

ђ) снимци, фотографије: навести врсту_____

е) текст, навести врсту _____

ж) мапа, навести врсту _____

з) остало: описати _____

1.3 Формат података, употребљене скале, количина података

1.3.1 Употребљени софтвер и формат датотеке:

a) Excel фајл, датотека _____

b) SPSS фајл, датотека _____

c) PDF фајл, датотека _____

d) Текст фајл, датотека _____

e) JPG фајл, датотека _____

f) Остало, датотека _____

1.3.2. Број записа (код квантитативних података)

а) број варијабли _____

б) број мерења (испитаника, процена, снимака и сл.) _____

1.3.3. Поновљена мерења

а) да

б) не

Уколико је одговор да, одговорити на следећа питања:

а) временски размак између поновљених мера је _____

б) варијабле које се више пута мере односе се на _____

в) нове верзије фајлова који садрже поновљена мерења су именоване као _____

Напомене: _____

Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?

а) Да

б) Не

Ако је одговор не, образложити _____

_____

2. Прикупљање података

2.1 Методологија за прикупљање/генерисање података

2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?

а) експеримент, навести тип _____

б) корелационо истраживање, навести тип _____

ц) анализа текста, навести тип __ _____

д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).

_____

_____

2.2 Квалитет података и стандарди

2.2.1. Третман недостајућих података

а) Да ли матрица садржи недостајуће податке? Да Не

Ако је одговор да, одговорити на следећа питања:

а) Колики је број недостајућих података? _____

б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не

в) Ако је одговор да, навести сугестије за третман замене недостајућих података

_____

2.2.2. На који начин је контролисан квалитет података? Описати

_____

_____

2.2.3. На који начин је извршена контрола уноса података у матрицу?

_____

_____

3. Третман података и пратећа документација

3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у _____ *репозиторијум.*

3.1.2. URL адреса

_____

3.1.3. DOI

_____

3.1.4. Да ли ће подаци бити у отвореном приступу?

а) Да

б) Да, али после ембарга који ће трајати до _____

в) Не

Ако је одговор не, навести разлог _____

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани.

Образложење

_____

_____

3.2 Метаподаци и документација података

3.2.1. Који стандард за метаподатке ће бити примењен? _____

3.2.1. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.

_____

_____

Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.

_____

_____

_____

_____

3.3 Стратегија и стандарди за чување података

3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? _____

3.3.2. Да ли ће подаци бити депоновани под шифром? Да Не

3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? Да Не

3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена?

Да Не

Образложити

_____

_____

4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.

4.1 Формални стандарди за сигурност информација/података

Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности (*https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html*) и одговарајућег институционалног кодекса о академском интегритету.

4.1.2. Да ли је истраживање одобрено од стране етичке комисије? Да Не

Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање

_____

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да Не

Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:

а) Подаци нису у отвореном приступу

б) Подаци су анонимизирани

ц) Остало, навести шта

_____

_____

5. Доступност података

5.1. Подаци ће бити

а) јавно доступни

б) доступни само уском кругу истраживача у одређеној научној области

ц) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:

_____

_____

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима:

_____

_____

5.4. Навести лиценцу под којом ће прикупљени подаци бити архивирани.

_____

6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

_____

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

_____

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

_____