Artificial Intelligence (AI) Accelerator

By

Ali Ahmad Altaf

Ahmad Shehroz Kayani

Muhammad Hasnain

Fatima Sheikh

Muhammad Abdul Rehman

Supervised by:

Dr. Hussain Ali

Submitted to the faculty of Department of Electrical Engineering,

Military College of Signals, National University of Sciences and Technology, Islamabad,

in partial fulfillment for the requirements of B.E Degree in Electrical (Telecom) Engineering.

June 2022

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to officially state that the thesis work contained in this report

"Artificial Intelligence Accelerator"

is carried out by

Ali Ahmad Altaf, Ahmad Shehroz Kayani, M Hasnain, Fatima Sheikh, M Abdul Rehman

under ***DR. HUSSAIN ALI*** *supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Electrical (Telecom.) Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.*

Approved by

[illegible]

Supervisor
Dr. Hussain Ali
Department of EE, MCS

Date: May 23rd , 2022

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan'Wa'Tala is the sole guidance in all domains.

Our parents, colleagues, and supervisor for their guidance.

The group members, who through all adversities worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has 9% similarity index. Turnitin report endorsed by Supervisor is attached.

[illegible]

Ali Ahmad Altaf

00000264480

[illegible]

Ahmad Shehroz Kayani

00000245481

[illegible]

Muhammad Hasnain

00000242819

[illegible]

Fatima Sheikh

00000246070

Abdul Rehman

Muhammad Abdul Rehman

00000251754

[illegible]

Signature of Supervisor

Abstract

In the computer idea and many prevalent machine learning tasks, such as language recognition and fraud face detection, there has been immense importance of Convolutional neural networks (CNNs). Artificial intelligence accelerator is used to run many Machine Learning algorithms. Various hardware platforms are used to support its processing. However, there are many challenges that need to be addressed for the successful computation of these algorithms such as high computational processing, cost efficiency and low power consumption at the same time. Field Programmable Gate Array (FPGA) technology can be customized to meet the specific requirements for the implementation of ML algorithms. The use of FPGA in deep learning has been increasingly significant due its capacity for maximizing parallelism and energy efficiency. Convolutional Neural Networks (CNNs) have become the benchmark in bringing high accuracy in many applications using machine learning or deep learning and speech recognition. For faster and speedy results, we need to accelerate CNN algorithms. FPGA has exceptional features which make it an achiever in accelerating deep learning algorithm. The prominent features are flexibility, low latency, and high-power efficiency. Flexibility allows us to customize hardware even down to its bit level. It becomes a competitive in its feature when precision and accuracy is needed in deep learning algorithm.

Table of Contents

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Project Overview

Artificial Intelligence (AI) is an emerging research field and has the objective to incorporate many Machines Learning (ML) algorithms. However, there are many challenges to be addressed to realize efficient AI implementations in hardware. For this a potentially strong hardware with AI implementation capabilities FPGA technology is being used.

1.1.1 Artificial Intelligence

Artificial intelligence is a field of computer science that deals with the modeling of human intelligence, that is programmed to be able to think like a human. It is the ability of computer or robot controlled by the computer that is programmed by the humans. the ability of a machine to behave intelligently in the same way as a human. Artificial intelligence covers the idea that human intelligence can be stipulated in a way that it can be easily reproduced by a machine. It performs the tasks goes from easy and basic to hard and complex. Google, an advanced web search recommends systems (e.g., YouTube, Amazon, and Netflix), understands human speech (e.g., Siri and Alexa) are AI. Similarly, the automation in decision-making with that playing at the top tier level in strategic gaming systems are all examples of AI applications.

1.1.2 Machine Learning

Machine learning is a subset of Artificial Intelligence. Machine learning is a concept of learning from experience and examples. In this, data of the complex problem is provided to the algorithm and computer can make logical predictions based on the given data. Moreover, computers use statistical analysis for the output values that fall within a specific desired range. In this way sample data and automate decision making process in building computer models is aided with the use of machine learning.

1.1.3 Hardware Accelerators

Hardware accelerators are configured to hasten computer science applications. To achieve high-efficiency embedded vision applications, runtime efficiency with power consumption must be limited [1]. The combination of embedded computer vision hardware accelerators (e.g., CPUs, GPUs and FPGAs), and their related vendor personalized visual libraries, developers have a challenge in directing this fragmented solution [2]. Hardware used for acceleration include:

Graphics Processing Unit (GPU)

They are designed to handle motion of the image. GPUs are increasingly utilized for large-scale data processing, speeding sections of an application while the rest executes on the CPU. Modern GPUs' extreme parallel processing enables a user to process billions of data and information.

Field Programmable Gate Array (FPGA)

FPGAs are integrated circuits that can be altered and reprogrammed an infinite number of times once they are manufactured. They are the basis for reconfigurable computing, a computer model that divides applications into parallel, application-specific pipelines. The advantage of reconfigurable computing is that it combines the speed of hardware with the flexibility of software, essentially combining the greatest features of both hardware and software [3]. An FPGA's basic design is made up of input/output blocks, customizable logic blocks (CLBs) and routing channels. Each CLB, which appears in a basic routing structure, is built of look-up tables and flip-flops that may be configured to perform combinational or sequential logic [3]. CLBs are surrounded by input/output blocks (IOBs) that allow them to connect with external devices. Because the universal routing structure allows for random wiring, designers can link the logic parts in any way they see fit. To implement the designs on an FPGA, a hardware description language (HDL), such as Verilog or VHDL is used.

1.2 Problem Statement

The challenge for high computational processing is therefore critical as AI algorithms and techniques are becoming increasingly sophisticated and advanced. To manage the challenging factors for emotion recognition in the world, Deep learning techniques have progressively been executed. This progress comes at the expense of a large computational cost as CNNs. As a result, specialized hardware is needed to accelerate their implementation. Graphics Processing Units (GPUs) are the universally used platform to implement CNNs as they offer optimal performance in terms of pure computational throughput but requires high power and energy consumption [4].

1.3 Proposed Solution

FPGA will be programmed to fit the ML algorithm and a fast data access efficiency will be received compared with regular GPUs. FPGA decreases power usage through ML algorithms on the hardware design.

RISC-V (Reduced Instruction set computer architecture) and open- source Instruction set Architecture (ISA). It allows system designers to add custom instructions according to their need. In this project we have added custom instruction and hardware unit to accelerate the ML algorithm.

Lastly, optimizing memory reads/writes overhead by using DMA, so the processor is free for other tasks

1.4 Objective

Our objective is to design an FPGA based AI Accelerator with:

- Capability of running optimized ML algorithms.
- Less computational complexity.
- Implement 32-bit RISC-V Architecture on FPGA
- Generic implementation of CNN accelerator on RISC-V.
- Optimizing memory reads/writes overhead.

1.5 Scope

Artificial Intelligence (AI) finds it market wherever ML algorithms are being applied. To serve the purpose, we will be testing an ML algorithm such as Convolutional Neural Network (CNN). The algorithm would be accelerated by our hardware accelerator. A CNN algorithm will be used and would be comparing the non-accelerated and accelerated outcomes We would observe the accelerated one being less complex in regard to computation.

1.6 Relevant Sustainable Development Goals

The project will provide us with the platform for acceleration of Artificial Intelligence (AI) algorithms. Advancements in AI has made every industry to shift towards it. With that there arises a problem of high computational complexity. Our project will aid in reducing such complexities. It will also enhance research and upgrade industrial technologies.

We are addressing the SDG's #9 which is "Industry, Innovation & Infrastructure".

CHAPTER 2: LITERATURE REVIEW

2.1 CNN MODEL

Convolution Neural Networks is the field of Deep Learning which works on the connectivity patterns of Neuron similar to the Human Brain and inspired by the organization of the visual cortex. CNN model as shown Figure 2.1 classify the input images by capturing their spatial and temporal dependencies through filters [5]. CNN picture sorting takes an input picture, processes it, and groups under some classes (like domestic and wild animals, human etc.). A series of convolution layers with filters (Kernels) is input picture, pooling, all connected layer associated, put a soft-max function to classify an object with probabilistic values between zero and one. The figure shows a flow of CNN to associate an enter image and classifies the object based on values.

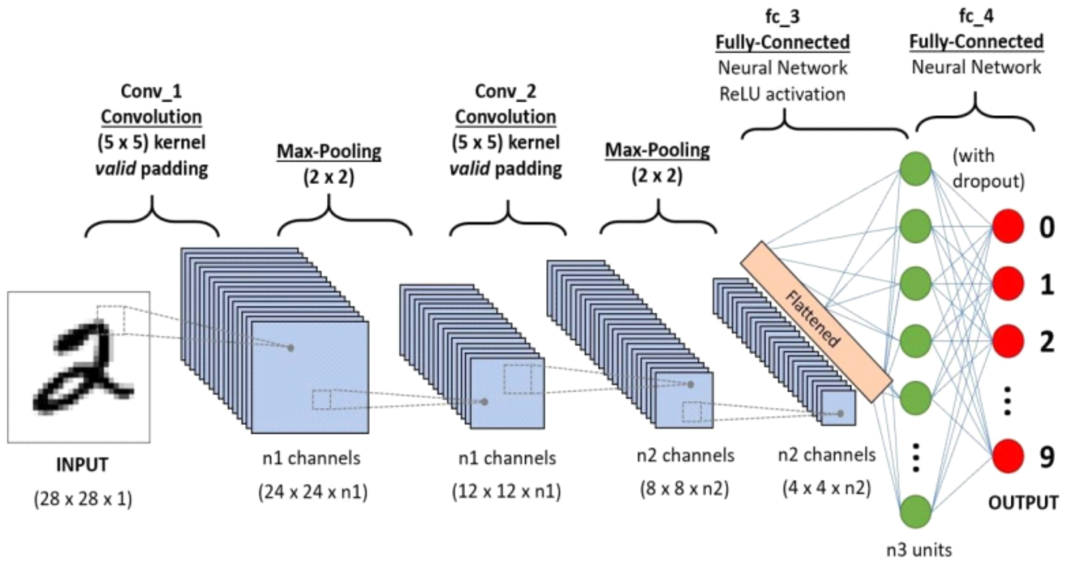


Figure 2.1 CNN Model Architecture [6]

Convolution Layer

In this layer, a numerical operation called convolution is done. It is used to reduce the size of the image [7] and extract specialized features according to kernel. Convolution operation simply passes the kernel over the image and performs point by point multiplication of filter with image pixels. Then the result is summed up to give output pixel. The process is repeated by moving kernel over complete image to give output image.

Pooling layer

The further down sampling and decreasing the matrix scale can be called as pooling. A filter now is the left-out results of the previous layer, and it selects a particular range out of every cluster of values. In this way the network is now able to train itself much quicker, while focusing on all the significant data in every aspect of the picture [8].

Fully connected layer

The input which represents the output of the prior layers can be a 1-D vector. The output will be a listing of probabilities for different potential labels attached to the picture (e.g., dog, cat, bird). The label that has the highest probability is that the classifications call [8].

DMA

"The characteristics of computer systems which allow some hardware subsystems to get their access to RAM (random access memory) or main memory without the CPU interference is called Direct Memory Access (DMA)".

It will completely be occupied by the read or write operation and therefore cannot be used for other tasks when we are DMA is not using. CPU firstly starts the transferring while using the DMA, then performs other tasks while the transfer is happening, and when the transfer is completed, the DMA gives an interrupt signal to CPU. When the CPU cannot maintain with the data transfer rate than this feature is helpful, or when the CPU needs to do work while waiting for comparatively slow I/O data transfer.

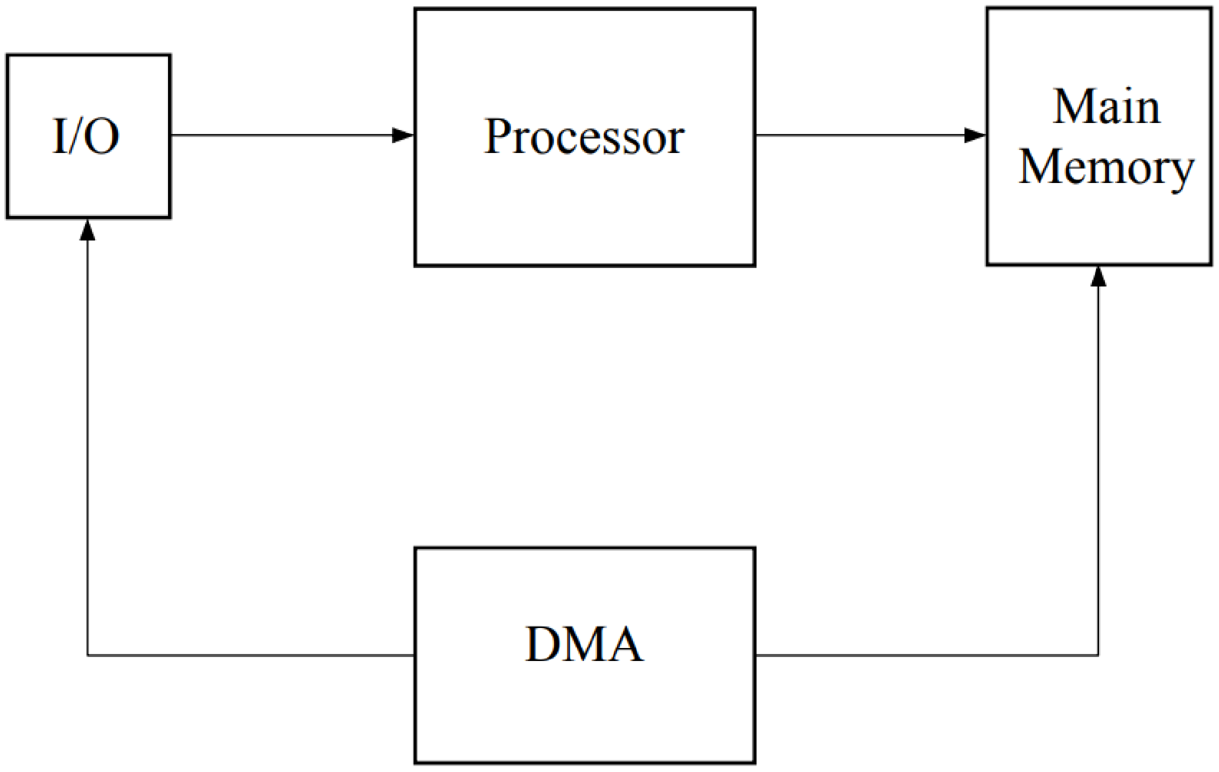


Figure 2.2 DMA

2.2 RISC-V Architecture

RISC-V is an open source, friendly, free ISA enabling is the latest period of processor innovation through public collaboration. It is structured as a small base ISA with a variety of extensions and freely available to both academia and industry. RISC-V has multiple phases as shown in Figure 2.3 [9]. The first phase is the fetch phase in this instruction loads from the memory. The other phase is the decode phase in this instruction is decoded i.e., which type of instruction is identified either it is load-store or ALU instruction. The next phase is the execution phase. In this phase required operation is performed on the decoded instruction. The next phase is the memory phase. In this phase if instruction is load or store instruction, memory is read or written. The last phase is a write-back phase. In this phase results are written into the register file. Memory is of two types‟ instruction memory and data memory. It has an arithmetic and logic unit (ALU). An arithmetic and logic unit may be a digital circuit wants to perform arithmetic and logic operations. Addition, subtraction, multiplication, and division are the examples of arithmetic operations [10]. AND, NOT, and OR are logical operations. All these operations are performed on values in the register file. It is a small amount of storage available. The Program counter (PC) is a register that has the address of the next instruction to be executed from the memory [11].

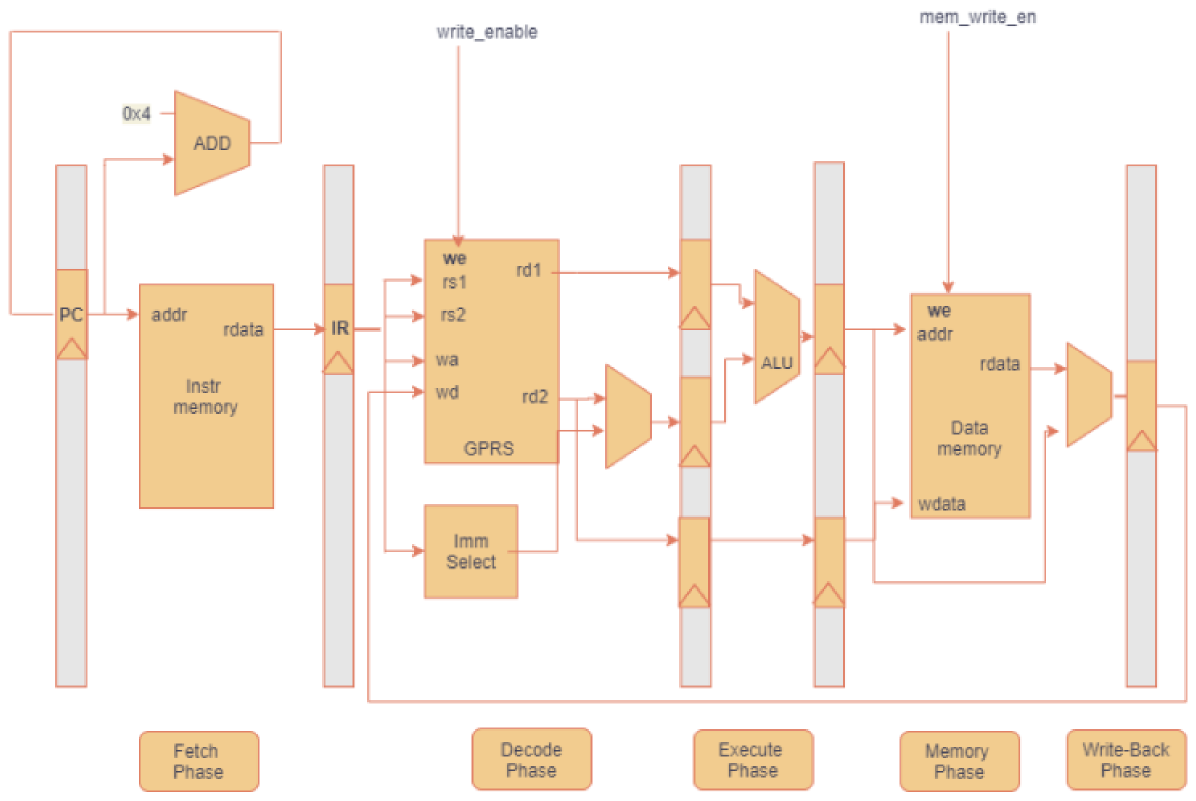


Figure 2.3 RISC-V Five Stage Pipeline

2.3 CNN Accelerator Strategies:

CNN has multiple layers that take much more time to execute. To accelerate the CNN we adopt two strategies, strategy per phase and vector block extensions. A detailed explanation of these strategies is given below. 2.3.1 Strategy per stages: This is the first strategy of the CNN accelerator [12]. In this strategy, author implemented the CNN accelerator on RISC-V and add Customize instruction to speed up the CNN process by including load vector, store vector, multiplication, division custom modules in architecture. Author implemented five-stage pipelines but different architecture from conventional RISC-V by adding a specific processing module to implement their boost up commands as shown in Figure 2.4 [13]. Properties and working of each stage are given below.

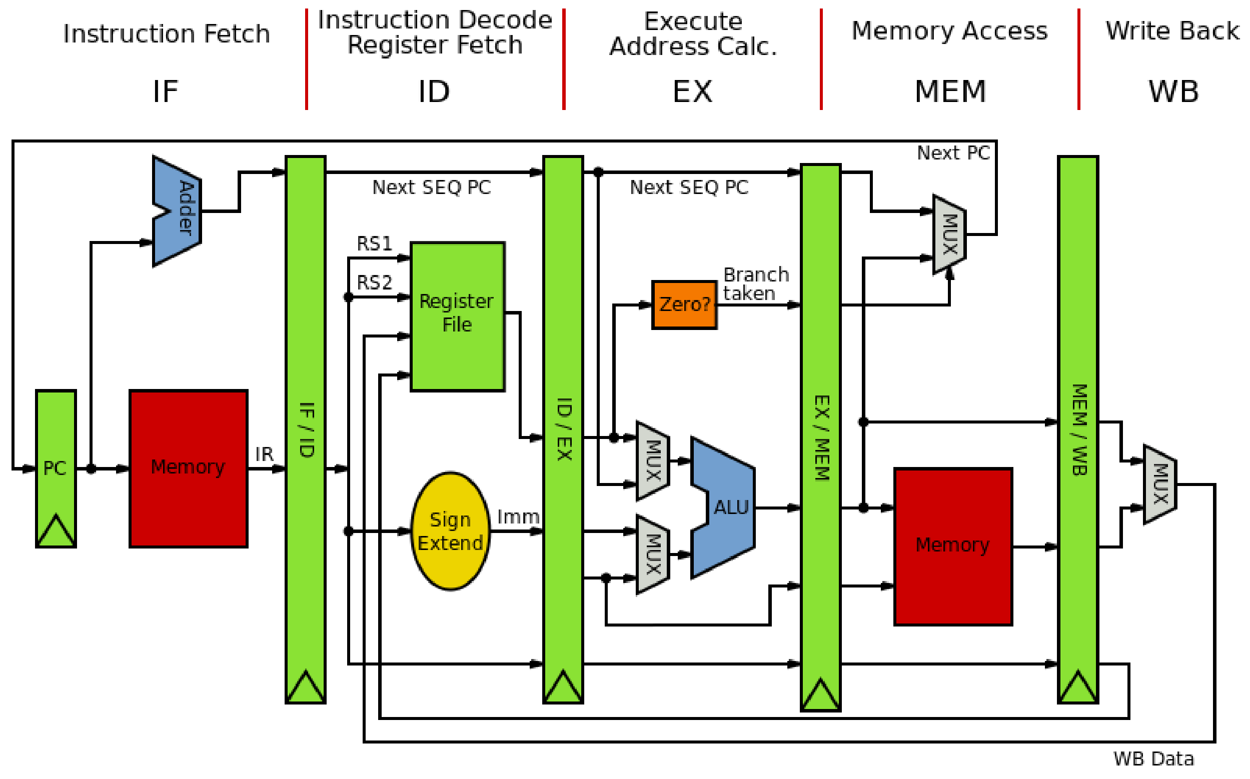


Figure 2.4: Custom Instructions on Every Stage CNN Accelerator [14]

2.4 FPGA-based Accelerators

There are many processors based on Intel, GPU and also work done on FPGA but FPGA process are not using generally in world because it is more challenging to process FPGA on ML algorithms [15]. Since it is an FPGA, the peak performance is equal to the DNN model, the performance peak is reported for using GoogLeNet source which ran at 900 fps [16].

CHAPTER 3: IMPLEMENTATION AND WORKING

3.1 CNN Implementation Steps

The implementation steps of CNN on RISC-V are shown in Figure 3.1. Python has become default platform for implementation of CNNs over last few years. Many existing implementations make it easier to make task of designing and implementing CNNs easier. We start with a python model we trained and validate the python model. Once a CNN is trained, for inference we require parameters and weights. In our design we are optimizing inference part of the CNN. So, we extract weight from the python model. Then we ported CNN to C language to port it on RISC-V.

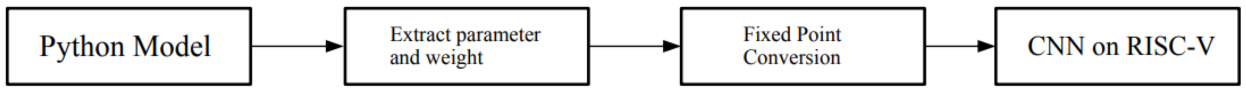


Figure 3.1 CNN Implementation Steps

3.1.1 Python model

We have the MNIST database. It is one of the common data sets used for image classification. It contains sixty thousand training images and ten thousand testing images. For the classification of the image, the image passes through the different layers of the model. The Layers of our model are shown in Figure 3.2.

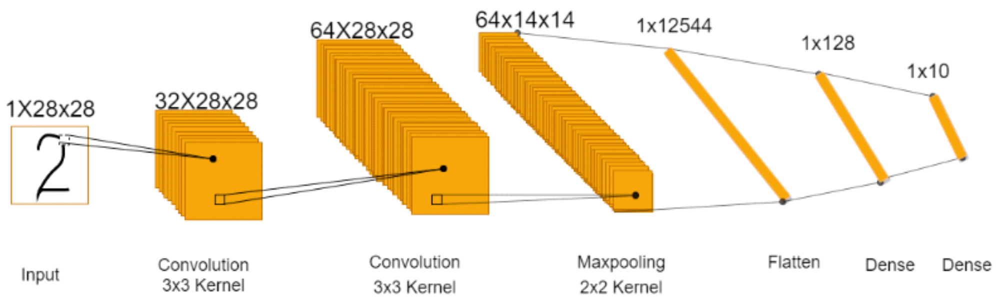


Figure 3.2 Python Model for RISC-V Implementation [17]

3.1.2 Extract Variables & Weights

In the second step, we extracted the weights from the python model. For this purpose, all the parameters are written in one file from the python model.

3.1.3 Fixed Point Conversion

The python code is converted into C code. All the weights are in the floating-point format are of size 32 bit but since we need to port part of it in FPGA, and fixed-point numbers are more suited for FPGAs. For this purpose, we convert data into fixed point 1.15 formats. The maximum and minimum range of the weights in floating-point format is between 0-1. That's why we convert into 1.15 formats. There is a special case of convolution addition where the format is changed is of 1.31 but when data c out the 1.15 format is retained.

3.1.4 RISC-V-32I

We explored some open-source RISC-V cores and a suitable core for CNN accelerator that we found was ORCA by vector blox. ORCA highlighted features are given as:

- ORCA is a completely open source
- Highly parameterized, ideally appropriate for FPGAs, transportable across FPGA vendors.
- It can be configured to either I or M extensions only. So, it has smart performance and space.
- It is often used as a standalone processor however was designed to be a host to vector blox proprietary light-weight vector extensions (LVE).
- It has optimum AXI3 and AXI4 instruction and information caches, a separate AXI4 Lite interface for uncached transactions, associated an auxiliary interface.

- Use dual-ported block rams on the FPGA for the register file.

As ORCA is compatible with the tool chain so we need to install tool chain for

Cross compilation of C code of CNN model

3.2 RISC-V GNU-Tool Chain Standard Installation

As orca is 32-bit RISC-V processor with integer and multiplication extensions. We installed RV32IM bare metal variant of RISC-V-GNU-Tool chain. The main step is downloading and install Tool chain in Ubuntu.

3.2.1 Profiling

The total time and number of cycles taken by each layer are shown in Table

Table 3.1 Profiling of CNN Layers

Sr.No	Layers	Time(ms)	No Of Cycles
1	Convolution1	150	15,034,049
2	Activation1	2	210,122
3	Convolution2	8,857	885,760,373
4	Activation2	4	457,291
5	Maxpooling1	13	1,361,133
6	Flat	1	195,435
7	Dense1	1,080	108,085,784
8	Activation3	0	1,865
9	Dense2	0	76,793
10	Activation4	0	557

Total		**10,107**	**1,010,783,402**

3.3 Technical Requirement

The Cora Z7 is very optimized SoC development board featuring the power and flexibility of Xilinx's Zynq-7000 SoC family [18]. The Zynq-7000 SoC family integrates the software programmability of an ARM-based processor with the hardware programmability of an FPGA, delivering optimized system integration [18].

Figure: Cora z7-7000 FPGA board [18]

3.4 CNN Layers Execution Timing Analysis

As it is shown from the above table the activation3, dense2, activation4 layer take lesser time than other layers i.e., 0ms. The flat layer takes more time layers i.e., 1ms. The activation1 layer takes more time i.e., 2ms.

The activation2 layer takes 4ms. The max-pooling layer takes 13ms, the convolution1 layer takes 150ms. The dense1 layer takes even more time i.e., 1080ms. After that convolution2 layer takes much more time than all layers i.e., 8857ms. The total time for both convolution1 and convolution2 is 9007ms and also the number of cycles is much more than other layers. So, we need to accelerate these layers to accelerate CNN. That’s why we make the accelerator for the convolution layer.

CHAPTER 4: HARDWARE DESIGNING

4.1 RISC-V Bring-Up

For bringing up the RISC-V core we have used Ubuntu 18.04 and Vivado version 2018.3. In this section, we will describe the steps required to bring up the RISC-V core. Prerequisites of using a RISC-V core include using Linux based machine and having Vivado Design Suite installed on the system. The bring-up of a RISC-V core comprises of the following steps: Building Tool chain, Building Vivado design project, and programming the RISC-V core. Below we have described each of these steps in detail.

4.1.2 Building Toolchain

In this step, we build the binary tools, gcc, and newlib library for 32IM-core. Numerous scripts are available online for carrying out these steps. Refer to Appendix A to see the script used by our team. For using the provided script, set "RSCV_INSTALL" variable in script.

4.1.3 Building Vivado Design Project

A script named "make file" is provided in Appendix B to make the Vivado Project for the RISC-V core. Just redirect to the file directory and run the "make" command in the terminal. This will generate the project from the terminal itself. Alternatively, run "make GUI" to build the whole project in the Vivado environment so user can use the GUI tools to manipulate the project according to his need. To make changes to Block Design created in Vivado, the user has to export and save the .tcl file. The "Make" command does this automatically for the user whereas, if the user has opted for the GUI option, then "make archived" can be used to save the changes to the design. If only RISC-V parameters are to be changed, arguments can be given to the make file to achieve it. Parameters can also be placed in an optional config.mk file before running the make command.

4.1.5 Programming the RISC-V Core

The make file includes the functionality of programming the FPGA and RISC-V core as well. To program the FPGA, use “make pgm‟ command. This will download the bitstream to the board. Then use “make run‟ to compile, build and load the software over JTAG to the instruction memory and start the program execution.

4.2 CNN Accelerator Design

In this section, we discuss the CNN accelerator design in detail. The CNN accelerator design can be broken down into two main parts: namely the hardware design and the software design.

4.2.1 Hardware Design

The crux of CNN accelerator hardware is mainly based upon two main components i.e. the processor and the accelerator itself. The processor runs the C code for CNN. It uses the DDR memory from Zynq system to store the program code and data. RISC-V uses the custom developed hardware accelerator to perform the convolution. This process is started when RISC-V writes the inputs to the accelerator. The hardware then runs the convolution process and produces results. These results are saved in the BRAM by the accelerator and the RISC-V is notified that the process has been completed. The Figure 4.1 below illustrates this process in a block diagram. In the subsequent subsections, each of these components is discussed in detail.

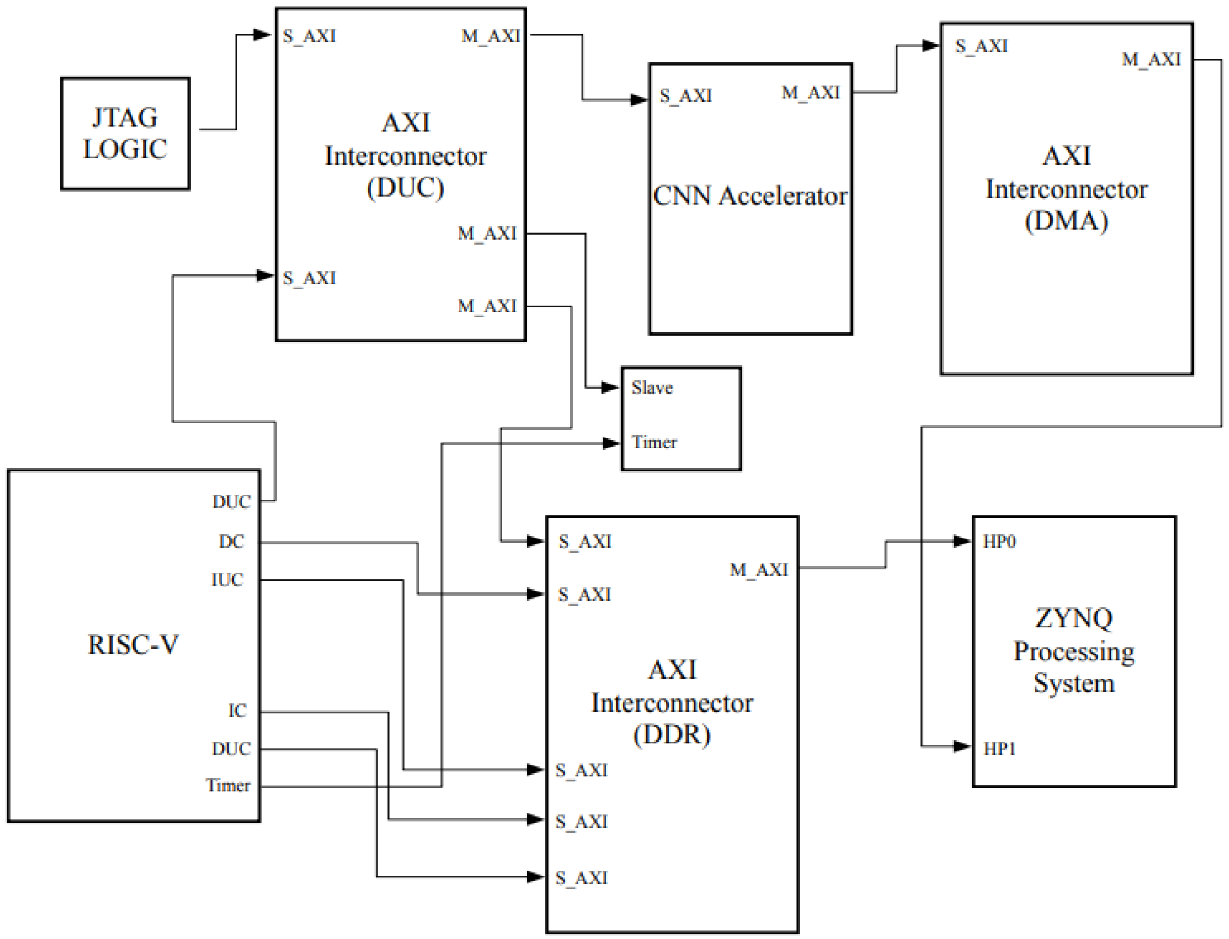


Figure 4.1 CNN Accelerator on RISC-V Block Design

4.2.1.1 RISC-V

In our project, 32-bit RISC-v with integer and multiplication is used to run C bare metal code for CNN model. RISC-V core is written in VHDL hardware descriptive language with a lot of configurable parameters. RISC-V used AXI3/4 interface to interact with peripheral and other blocks. A 32-bit timer is also attached with RISC-V core as a peripheral for time analysis. RISC-V has mainly two types of AXI ports; cached port and uncached ports. In RISC-V both data and instruction memory have both cached and uncached port. These ports are used to fetch instructions and data from the memory to RISC-V. An additional function of the uncached data port is to poll data available on external memories (including BRAM) so that it is constantly available to the processor. A timer AXI port is used to read the timer value.

Data Flow: The data and program section of C code generated by riscv-gnu-toolchain is written on zynq DDR through JTAG Logic for RISC-V processor. For the first time RISC-V fetches the data and instruction sections for c code using Data Uncached port "DUC" and instruction uncached port "IUC". Data and instruction data is stored in caches of RISC-V, and which is also a predefined section on Zynq DDR using Data cached "DC" and Instruction cached "IC" port. Then RISC-V operates on caches and uses "DUC" and "IUC" ports when it missed cache. "DUC" port also has a special purpose to interact with other peripheral of RISC-V and accelerator custom design. We use data uncached port for peripheral so we can directly change and access any variable/memory location by peripheral and accelerator. A RISC-V timer is also connected to the timer port for calculating the time and counting a specific value.

4.2.1.2 CNN Accelerator

The CNN Accelerator parallelizes the process of convolution by using a 3x1 multiplication strategy. It uses three different BRAMs to store the input row separately and one other BRAM to store the kernel. An address generation custom IP is used to generate address for each BRAM. Multiplication is performed on these inputs and the results are stored in yet another BRAM. RISC-V is notified about the completion of the process. The main components of CNN Accelerator are shown in the following Figure4.2: DMA, BRAMs, BRAM Address generator, Status registers, and convolution multipliers. The properties and functions of each of these components are discussed in Figure 4.2 below.

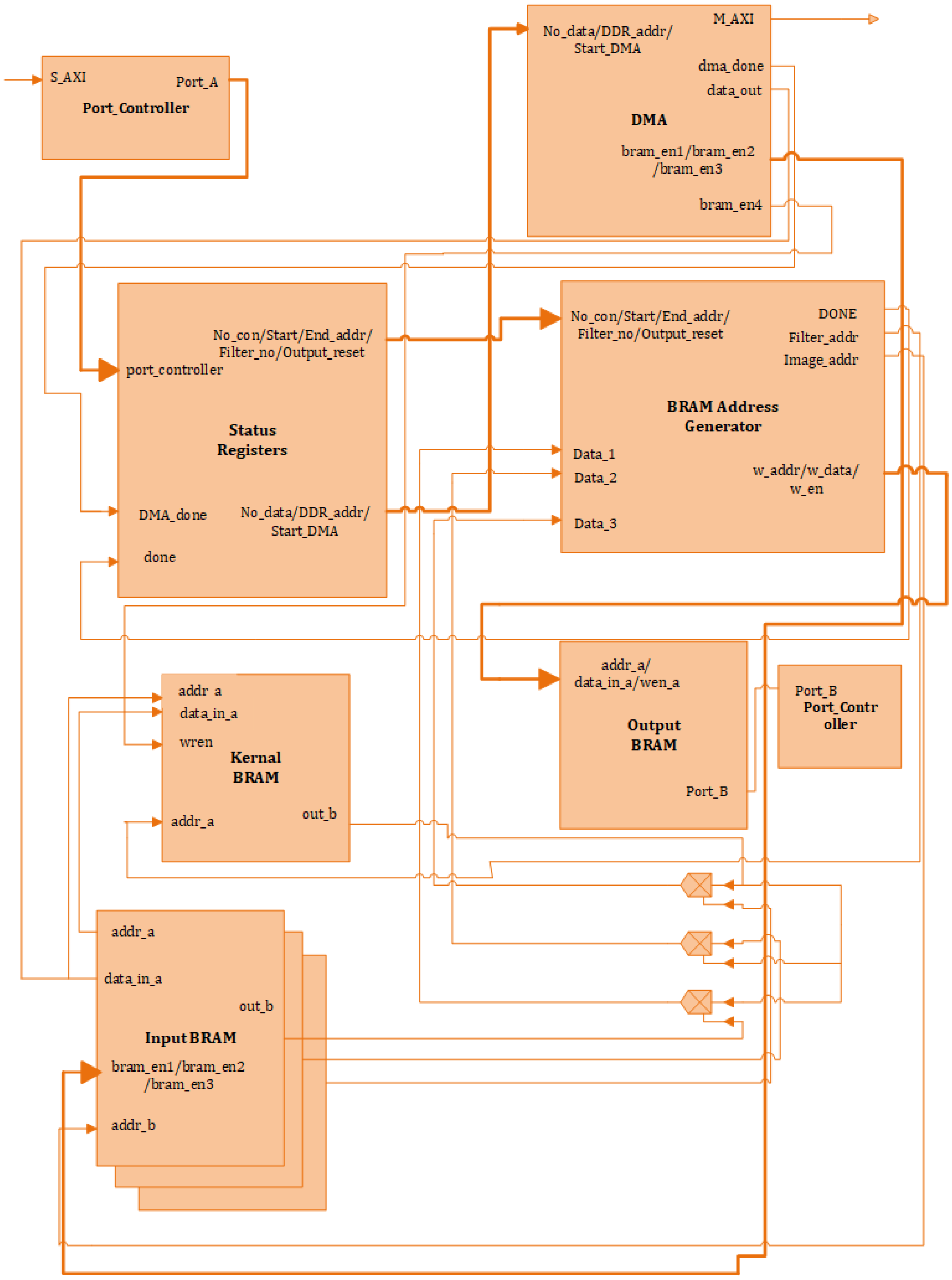


Figure 4.2 Accelerator Design

4.2.1.3 BRAMS

BRAM stands for Block Random Access memory. It is used in FPGA for storing a large amount of data. BRAM could be configured as standalone. It could be a single or dual port with different configurable parameters. We are using three BRAMs in our design, input, kernel, and output BRAM. All BRAMs are operating in Dual mode.

4.2.1.4 BRAM Address Generator

BRAM Address Generator is a custom IP designed by our team. This IP performs two main functions. Firstly, it receives the control signals from the status register IP core. When the BRAM Address Generator receives the start signal, last address, and filter number from the status register IP core, the IP begins its operation. This core produces the addresses for input BRAM and kernel BRAM corresponding to the provided filter number. The process is complete when the image address equals the provided last address as shown in Figure 4.3.

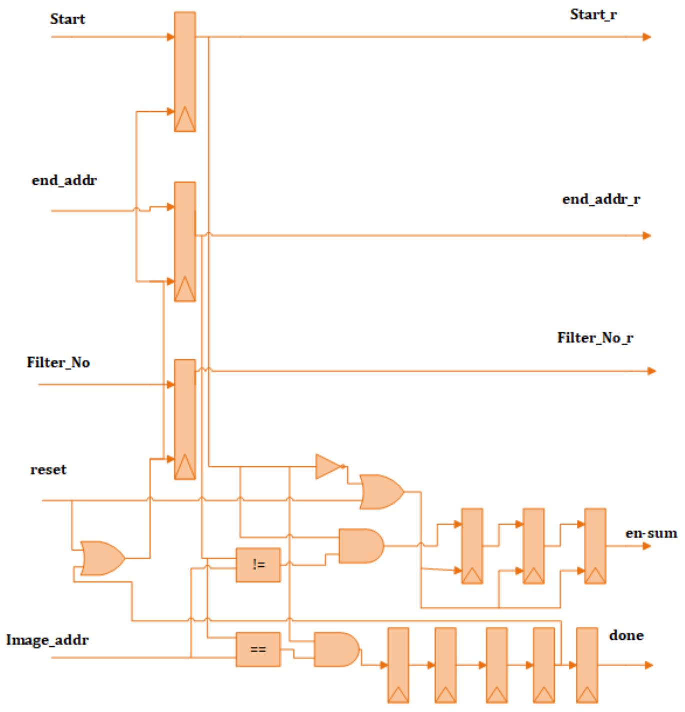


Figure 4.3 Address Generation Control Signals

Secondly, the main purpose of address generation IP is to generates a synchronized address for input, kernel, and output BRAM. The schematic for address generation and shifting process is shown in Figure 4.4.

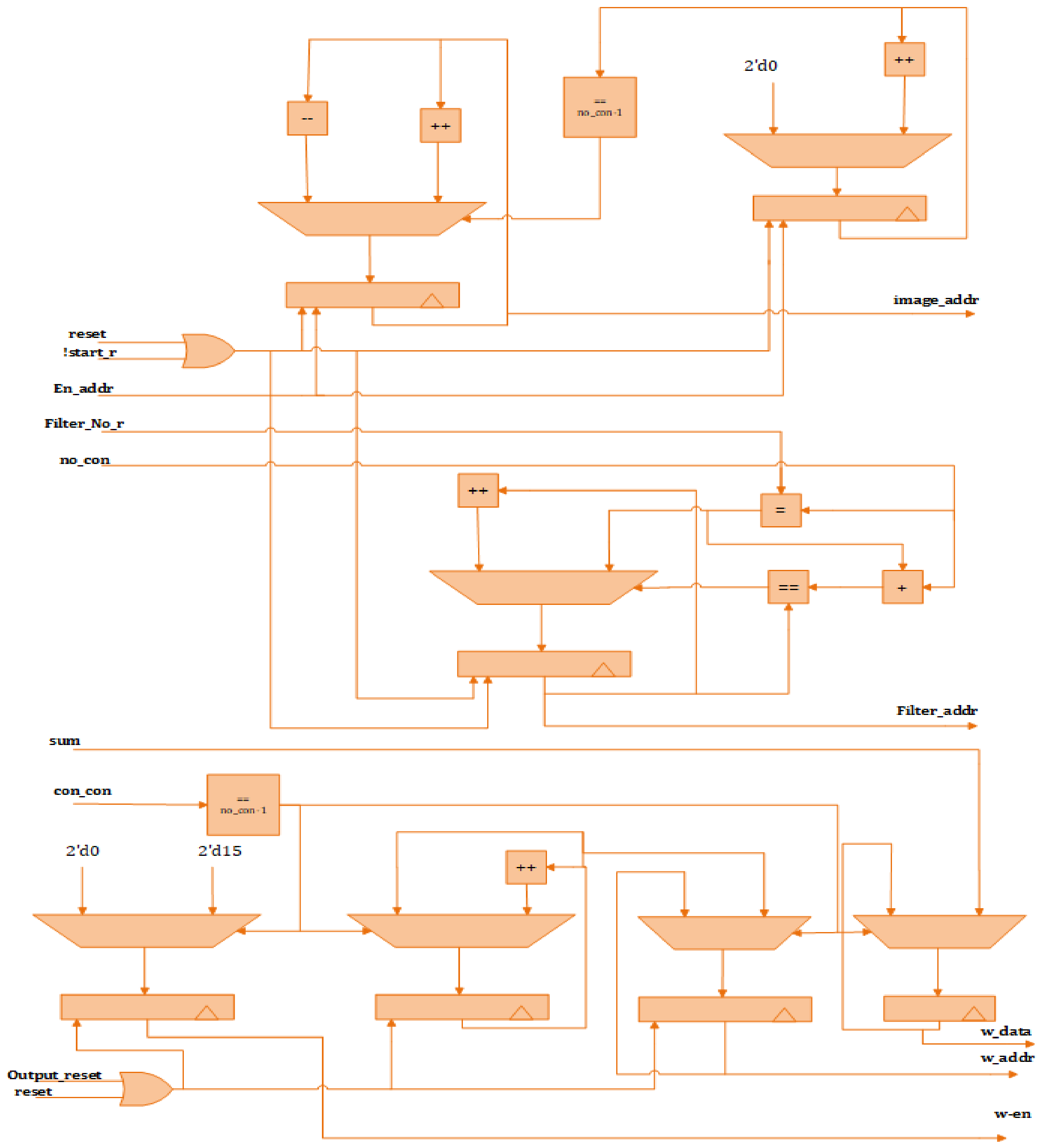


Figure 4.4 BRAMS Address generation RTL Design

The row shifting operation for convolution is also handled by the second module to generate a corresponding address for input BRAMs and kernel BRAM. Thirdly, the BRAM address generator IP performs addition over the results of the convolution multiplier. It also performs a cumulative sum for 3 cycles to generate convolution results. The schematic for this process is shown in Figure 4.5 below

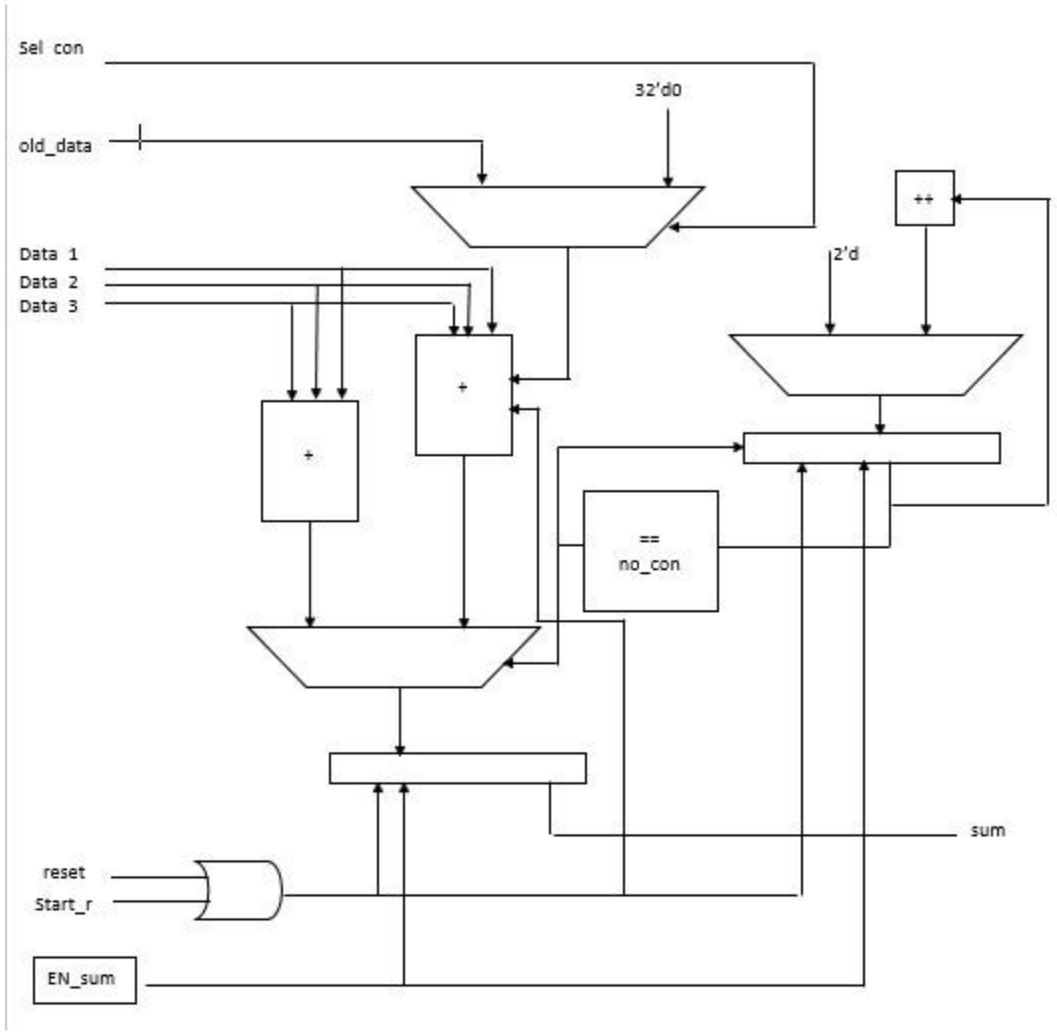


Figure 4.5 Cumulative Sum RTL Design

4.2.1.5 DMA

The main purpose of DMA is to further accelerate the design. The DMA is formed from AXI-MASTER IP. The IP was first tested using a verification IP **[19]**. AXI-MASTER has a state machine with the help of it first Write on a slave through counter and then Read from slave and at the end, the read data were Compared with write data. We have modified that state machine, which performs only one function at a time read or write depends upon the direction signal i.e. write_or_read given to it. It writes data when the direction is 1 and reads when the direction is 0. The modified state machine is shown in Fig 4.6

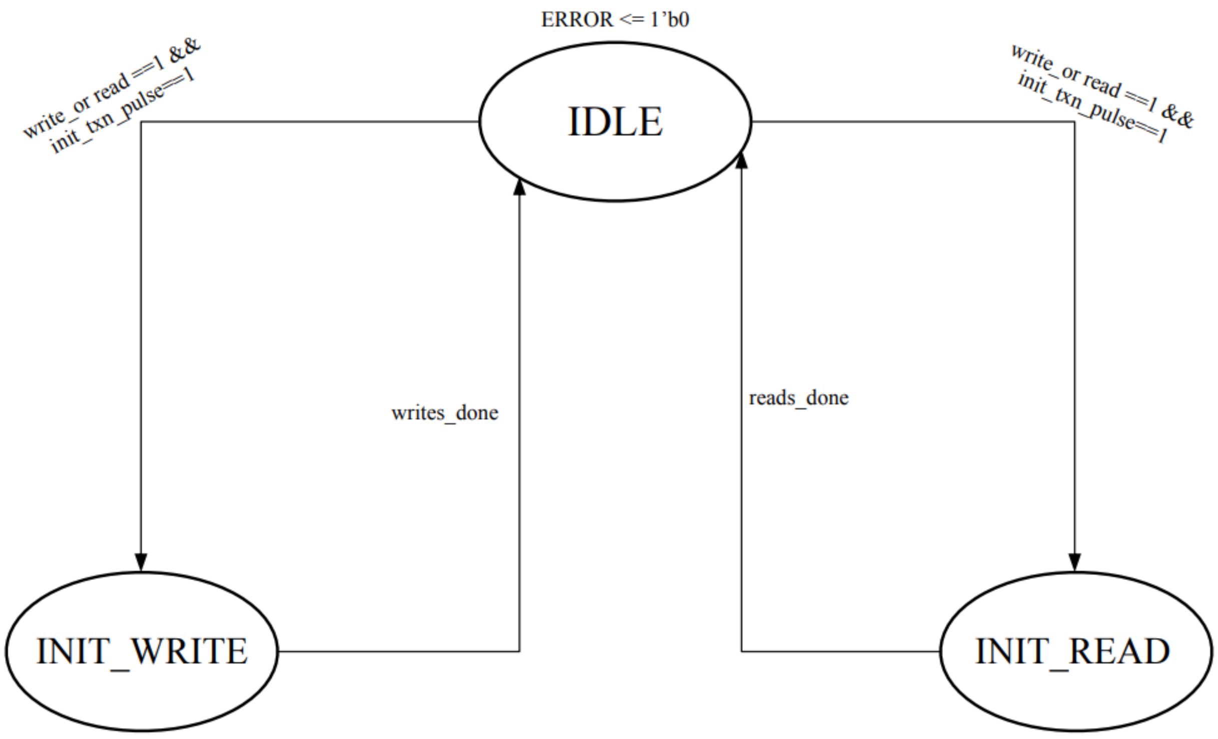


Figure 4.6 State machine of DMA

When write_or_read is equal to 1 DMA writes from BRAM to DDR. When write_or_read is equal to 0 DMA reads from DDR and writes it on BRAM. In our case, DMA is used only to reads from

DDR and writes on one of the four BRAMs depending upon write enable. DMA has the following ports as shown in Table 4.1

Table 4.1 DMA Ports

Sr. No	**Port**	**Description**
1	Direction	Read or write
2	DDR_address	Slave address
3	No_of_Data	No of data reads or write to DDR
4	Init_txn_pulse	Start DMA
5	BRAM_data_in	Input data from BRAM to write on DDR
6	BRAM_data_out	Output data from DDR to BRAM
7	write_enable(s)	This port may be one or more than one, depending upon attached BRAMs at output

4.2.1.6 Convolution Multipliers

Two staged pipelined multipliers are being used in 16x16 bit configuration to generate a 32-bit signed output.

Data Flow: The complete CNN accelerator consists of two paths, write and read from the CNN accelerator. Firstly, the input data and kernel for convolution are written on BRAMs using DMA. We are using three BRAMs for storing each input row. A kernel BRAM is used to store the filter for Convolution. After the loading process, a start bit from the status IP is used to start the convolution computation. Last address and filter-no are also selected by using status registers. The address generation IP generates the addresses for the input image and handles the column shifting process for convolution. Each Input BRAM gives the 16-bit data for convolution. Filter addresses are also generated by BRAM address generation IP. Kernel BRAM gives the 64-bit data where 48 least significant bits multiplied to corresponding 16 input data to produce 32-bit multiplication output BRAM address generation IP also accepts data from these three multipliers and sums them all. The second part of the Data flow is started where BRAM address generation IP generates the Output BRAM address, write enable, and write data. Write data is generated by three commutative addition of multiplication Sum. When the image address equals to last address the first-row convolution is done. The CNN accelerator sets the status registers done bit. This process will repeat for all rows of an input image. Whereas the Output address will auto-increment on write data unless reads data from Output BRAM using DMA and reset output address register through status register.

4.2.2 Software Design

The C code of CNN model executes the Convolution process using accelerator design. All other layers of Convolution Neural Network are running of RISC-V processor. The image is dumped on DDR using "dow" command on the following address *0x40000000***.** Run the image tcl file for image load to DDR and kernel to Accelerator BRAM is done by DMA. Convolution on Accelerator Software Flow. The software developer can access status register in their C code. The address for status register is *0x43C00000***.** The basic Flow design for performing computation on accelerator is shown in following figure.

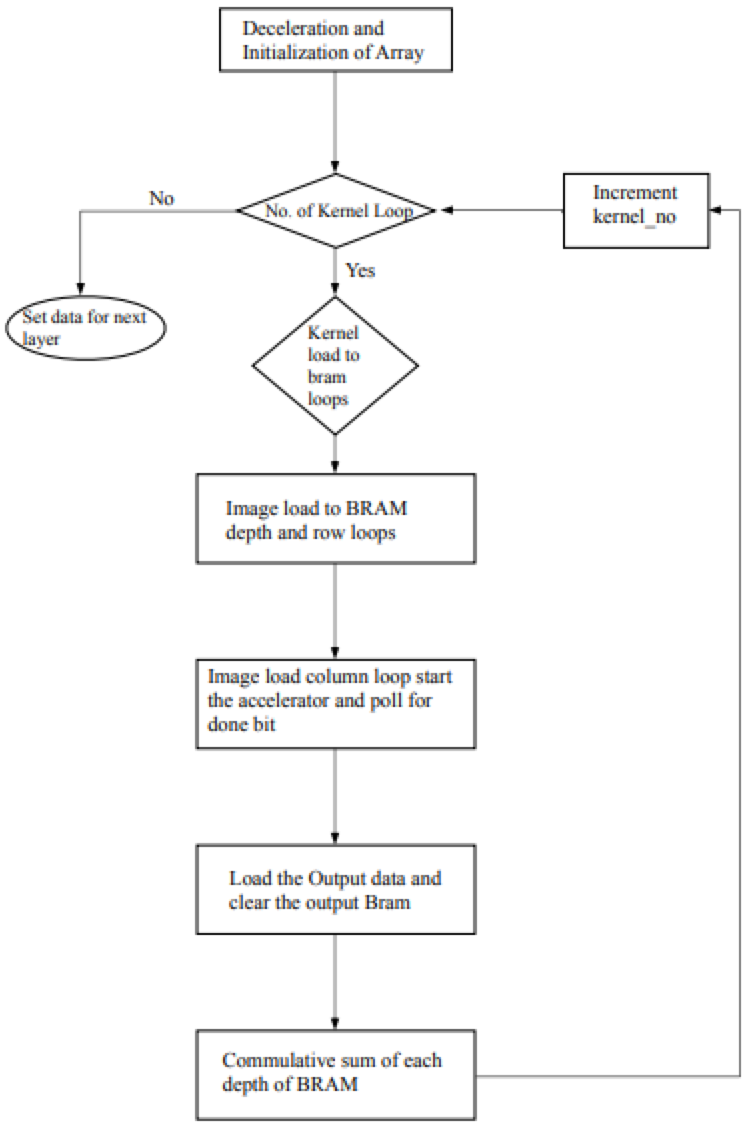

Deceleration and Initialization of Array
No. of Kernel Loop
No
Yes
Increment kernel_no
Set data for next layer
Kernel load to bram loops
Image load to BRAM depth and row loops
Image load column loop start the accelerator and poll for done bit
Load the Output data and clear the output Bram
Commulative sum of each depth of BRAM

Figure 4.7 Convolution on Accelerator Software Flow

The software developer can access all these BRAMs and status register in their C code. The address for each BRAM and status register is shown in following Table

Name	**origin**
Input_BRAM_Controller_0(Image Ram)	0x40000000
Input_BRAM_Controller_1(Kernel Ram)	0x42000000
Input_BRAM_Controller_2(Result RAM)	0x44000000
Status_reg_Controller	0x43C00000
axi_cdma_0	0x7E200000

CHAPTER 5: RESULTS

We have generalized the existing CNN architecture, now it supports every size of matrices, and can be used with any processor core. We can also change the image on run time. Lastly, we have optimized memory reads/writes overhead by using DMA, so it accelerates the old architecture which used to complete a MNIST image classification in 3084 ms to 2105 ms. The results are shown in following Table 5.1

Table 5.1 Comparison of existing CNN accelerator with modified

Sr. No	**Layers**	**CNN on RISC-V**	**CNN on RISC-V Based Accelerator without DMA**	**CNN on RISC-V Based Accelerator with DMA**
		Time(ms)	Time(ms)	Time(ms)
1	Convolution1	150	32	20
2	Activation1	2	2	2
3	convolution2	8,857	1827	871
4	Activation2	4	4	4
5	Maxpooling1	13	14	14
6	Flat	1	1	1
7	Dense1	1,080	1178	1181
8	Activation3	0	0	0
9	Dense2	0	0	0
10	Activation4	0	0	0

Total		**10,107**	**3084**	**2105**

CHAPTER 6: CONCLUSSION AND FUTURE WORK

6.1 Conclusion

We all know today's technology demands high computational speed in less time. As the technology grows people's patience is decreasing and they need a system that do their work in less time as possible. These days everything is being shifted to Machine Learning and Artificial Intelligence and these AI and ML algorithms usually take a lot of time for its computation. So, in order to eradicate these computational complexities a hardware is required that will accelerate this computational process. So, we designed an AI Accelerator based on FPGA that will help in speed up the AI algorithms that are being used in our daily life. In this way people don't have to wait long to do their basic daily life chores that involve AI computational.

6.2 Future Work

Currently we are not using the *write* side of DMA i.e., write on DDR through DMA, because it takes more time than usual. So, by shifting accumulation, bias addition, and max pooling to the hardware side, we will also be able to use the *write* side of DMA. As a result, this project will be more accelerated.

- DMA could use to load BRAMs instead of RISC-V and implements ping pong logics.
- BRAMS reload for every row computation that could be reused by making a state machine.
- Reconfigurable Accelerator for every python model.

BIBLIOGRAPHY

[1] N. Gupta, "Introduction to hardware accelerator systems for artificial intelligence and machine learning," Elsevier BV, 2020.

[2] "Heavy.AI," [Online]. Available: https://www.heavy.ai/.

[3] "Bucknell University," [Online]. Available: https://digitalcommons.bucknell.edu/.

[4] T. L.-T. S. N. Hanh Phan-Xuan, ""FPGA Platform applied for Facial," *Porcedia Computer Science,* vol. 151, pp. 651-658, 2019.

[5] C. L. G. A. C. T. A. D. B. Steve Lawrence, "Face Recognition: CNN approach," *Face Recognition: CNN approach,* p. 16, 1 January 1997.

[6] P. Ratan, "Analytics Vidhiya," Data Science Blogathon, 28 October 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.

[7] K. L. Hendrik Blockeel, "Computational Intelligence in Machine," *Machine LEarning,* vol. volume 111, no. Issue 5, May 2022, 2022.

[8] "IJARIIT," [Online]. Available: https://www.ijariit.com/.

[9] Michelogiannakis, 2016. [Online]. Available: http://inst.eecs.berkeley.edu/~cs152.

[10] "University of basrah," [Online]. Available: https://faculty.uobasrah.edu.iq/faculty/en.

[11] "Slide Player," [Online]. Available: https://slideplayer.com/.

[12] J. Y. H. a. Y. L. C. F. C. Poulet, " An fpga-based processor for CNN," *International Conference on FPGA,* pp. 32-37, 2009.

[13] B. M. a. H. C. M. P. A. Setio, "Memorycentric accelerator design for CNN," 2013.

[14] "Wikimedia commons," MIPS Architechture(Piplinined), 22 January 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:MIPS_Architecture_(Pipelined).svg.

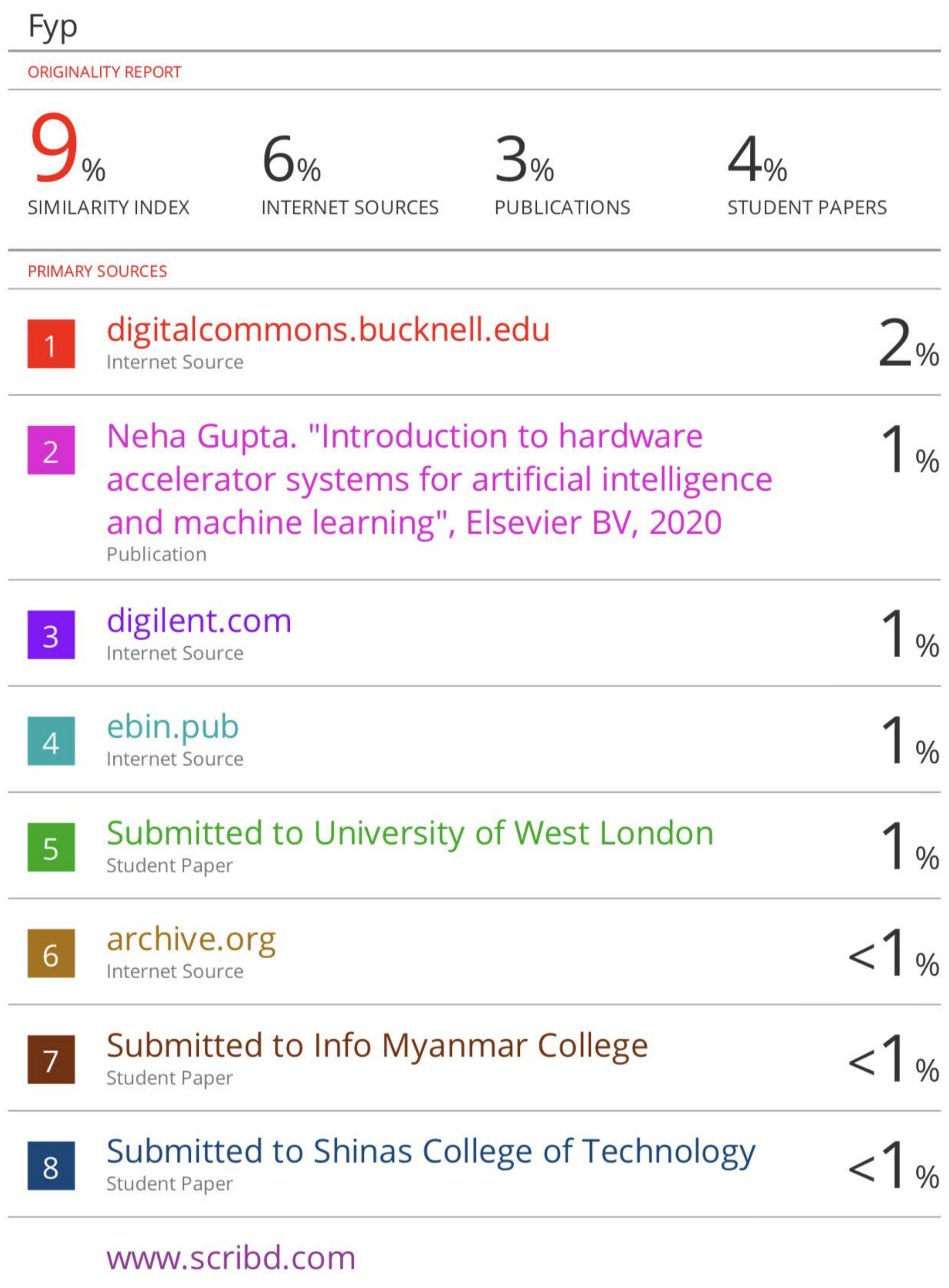
[15] "Internet Archive WayBack Machine," [Online]. Available: https://web.archive.org/.

[16] P. M. M. J. V. G. S. S. J. K. Albert Reuther, "Survey of Machine Learning Accelerators," *HPEC,* p. 6, 2019.

[17] "Data Camp," [Online]. Available: https://www.datacamp.com/tutorial/convolutional-neural-networks-python.

[18] "Digilent," [Online]. Available: https://digilent.com/shop/cora-z7-zynq-7000-single-core-and-dual-core-options-for-arm-fpga-soc-development/.

[19] "Xilinix Forums," 2020. [Online]. Available: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/AXI-Basics-4-Using-the-AXI-VIP-as-protocol-checker-for-an-AXI4/ba-p/1062002.



Fyp

ORIGINALITY REPORT

9%	6%	3%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	digitalcommons.bucknell.edu Internet Source	2%
2	Neha Gupta. "Introduction to hardware accelerator systems for artificial intelligence and machine learning", Elsevier BV, 2020 Publication	1%
3	digilent.com Internet Source	1%
4	ebin.pub Internet Source	1%
5	Submitted to University of West London Student Paper	1%
6	archive.org Internet Source	<1%
7	Submitted to Info Myanmar College Student Paper	<1%
8	Submitted to Shinas College of Technology Student Paper	<1%
	www.scribd.com	

9	Internet Source	<1%
10	Submitted to Teaching and Learning with Technology Student Paper	<1%
11	Submitted to Queen Mary and Westfield College Student Paper	<1%
12	www.edf-feph.org Internet Source	<1%
13	Submitted to Kaunas University of Technology Student Paper	<1%
14	Submitted to University Centre Peterborough Student Paper	<1%
15	mafiadoc.com Internet Source	<1%
16	F Hidayat, F Hamami, I A Dahlan, S H Supangkat, A Fadillah, A Hidayatuloh. "Real Time Video Analytics Based on Deep Learning and Big Data for Smart Station", Journal of Physics: Conference Series, 2020 Publication	<1%
17	Bo Peng, Jianguo Yao, Yaozu Dong, Haibing Guan. "MDev-NVMe: Mediated Pass-Through NVMe Virtualization Solution With Adaptive	<1%

Polling", IEEE Transactions on Computers, 2022
Publication

18	Sallar Ahmadi-Pour, Vladimir Herdt, Rolf Drechsler. "MircoRV32", Proceedings of the Workshop on Design Automation for CPS and IoT, 2021 Publication	<1 %
19	www.rawsound.com Internet Source	<1 %
20	open.library.ubc.ca Internet Source	<1 %
21	silo.pub Internet Source	<1 %
22	www.mdpi.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		