Real Time CNN-based Hardware/Software Co-Implementation for Object Detection in Autonomous Driving.

Abstract-This project is mainly for implementing an artificial intelligent neural network (A neural network is a type of machine learning which consists of a collection of building blocks each is called a neuron, it models itself after the human brain) it is designed for autonomous driving vehicles, the purpose of this project is to solve many problems that occur in our daily life this includes : traffic problems , car accidents , traffic overflow , avoiding high death rates due to traffic accidents etc.., we studied carefully how a basic neural network operates with the aid of different references and papers through the internet this includes : mathematics of the neural networks , different architectures, algorithms , types and the problems that may occur in any neural network, the results is that we are now aware of the main idea of implementing a neural network, how it works and the main requirements of the design , This will be the first time to implement faster RCNN on hardware as it was always used as software code not hardware components , in conclusion the final architecture to be implemented is typically the Faster RCNN(which is to be implemented on a hardware chip

Abbreviations and Acronyms.

- RELU: Rectified linear unit.
- Convnet: Convolutional neural network.
- R-CNN: Region based Convolutional neural network.
- ROI: Region of interest.

I. INTRODUCTION.

An autonomous car is a vehicle that can guide itself without human conduction. This kind of vehicle has become a concrete reality and may pave the way for future systems where computers take over the art of driving. An autonomous car is also known as a driverless car, robot car, self-driving car or autonomous vehicle [1].

A. Problem Definition.

It is worth asking at this point whether distrust of autonomous technology has a semblance of the truth and is it backed by some authoritative research. According to the WHO, approximately 1 million people die due to road accidents every year. And most of the accidents are caused due to avoidable reasons. Considering the fact that more than 90% of fatal car accidents are caused due to human negligence, the assertion that autonomous cars would not lead to a significant decline in the number of car crashes is simply absurd and defies even the most elementary logic.

B. Related Work.

- If there is no related work "For the best knowledge of the authors, this is the first Contribution in……."

Organization of the remaining chapters

C. Objective and Contribution.

In this work, we propose a real time CNN-based hardware/software Co-implementation for Object Detection in Autonomous Driving. Implementation is done using Tensorflow and High-level Synthesis methodology (C++ to Verilog). Critical parts in the design are implemented in hardware as it is faster than software. In object detection in autonomous cars, we need very fast response as very small delays may lead to hazardous events.

D. Organization of the Remaining Sections.

The rest of this paper is organized as follows.

In Section II we present the history and Background about Autonomous cars, Manufacturing Companies, an overview about neural networks and machine learning

In Section III we discuss our proposed architecture.

In Section IV we discuss our Implementation which includes both software and hardware Implementation.

In Section V, validation and results are presented.

II History and Background.

A. *Autonomous Car Components.*

The monitoring system of autonomous cars that consists of powerful sensors and LiDAR's view the objects ahead, with much more clarity, precision and a 360-degree view than the human eye and then chart the route of the car accordingly. While autonomous cars are good at identifying hurdles ahead, it cannot distinguish between pedestrians and inanimate objects.

vehicles? Nevertheless, the metrics of safety considerations cannot overrule autonomous cars because, in any given time and situation, they are statistically safer than human-driven cars, and intelligent AI-powered systems are immune to the foibles and temperamental indiscretions of humans.

What is needed is setting public expectations and gradually erasing doubts that the citizens have. Along with companies like Waymo, governments should also step in to allay the fears and disseminate the advantages of autonomous cars — that range from reduction in fuel consumption, less traffic congestion to the decreased probability of a collision. A framework conducive to emerging enterprises and minimal regulations that promote and incentivize innovation is needed, in addition to a dedicated task force and participatory approach between governments and private industry. No system is engineered to be free from the probability of error margin. The error margin can only be made negligible and then the cause of the error can be identified and rectified with a permanent effect. In autonomous cars, because of extensive research and cutting-edge technical equipment used, the error margin is already very low, and it is possible that it may be obliterated altogether once the test phase is over and the technology incorporates new innovations. Considering the enormous potential of autonomous car technology and the benefits it has in store for the society, to write it off and spell doom is akin to throwing the baby along with the bathwater.

Self-driving cars might seem like a recent thing, but experiments of this nature have been taking place for nearly 100 years. In fact, centuries earlier, Leonardo Da Vinci designed a self-propelling cart hailed by some as the world's first robot. At the World's Fair in 1939, a theatrical and industrial designer named Norman Bel Geddes put forth a ride-on exhibit called Futurama, which depicted a city of the future featuring automated highways. However, the first self-driving cars didn't arrive until a series of projects in the 80s undertaken by Carnegie Mellon University, Bundeswehr University, and Mercedes-Benz. The Eureka Prometheus Project proposed an automated road system in which cars moved independently, with cities linked by vast expressways. At the time, it was the largest R&D project ever in the field of driverless cars. Since 2013, US states Nevada, Florida, California, and Michigan have all passed laws permitting autonomous cars; more will surely follow. Autonomous vehicle components. Ouster's OS-1 3D LiDAR sensor captures point cloud data and camera-like images to help autonomous vehicles 'see' their environment How does an autonomous vehicle operate and make sense of what it sees? It comes down to a powerful combination of technologies, which can be roughly divided into hardware and software. Hardware allows the car to see, move and communicate through a series of cameras, sensors and V2V/V2I technology, while software processes information and informs moment-by-moment decisions, like whether to slow down. If hardware is the human body, software is the brain.

At Level Five Supplies, our tech is broadly categorized as follows:

- Data storage
- Drive-by-wire
- Positioning
- Power
- Processing
- Sensors
  - Camera
  - LiDAR
  - Radar
  - Ultrasonic
- Software

Autonomous vehicles rely on sophisticated algorithms running on powerful processors. These processors make second-by-second decisions based on real-time data coming from an assortment of sensors. Millions of test miles have refined the technology and driven considerable progress – but there is still a way to go. Driverless car technology companies the race to be the first self-driving car on the road is heating up. Level 5 autonomy is still some time away, but there is plenty else happening in the autonomous space, with aspects of driverless tech already making an appearance in today's mass-produced cars. Early adopters will enjoy benefits such as automatic parking and driving in steady, single-lane traffic.

B. *Manufacturing Companies.*

- Google – presently leading the charge via Waymo, its self-driving subsidiary
- Tesla – models are now being fitted with hardware designed to improve Tesla Autopilot
- Baidu – in the process of developing Level 4 automated vehicles
- General Motors – developed the first production-ready autonomous vehicle
- Toyota – working with ride-hailing service Uber to bring about autonomous ridesharing
- Nvidia – created the world's first commercially available Level 2+ system
- Ford – planning to have their fully autonomous vehicle in operation by 2021
- nuTonomy – first company to launch a fleet of self-driving taxis in Singapore
- BMW – has teamed up with Intel and Mobileye to release a fully driverless car by 2021
- Oxbotica – will begin trialing autonomous cars in London in December 2019

Challenges in autonomous vehicle testing and validation

As we can see, there are many different autonomous vehicle projects in various stages of development. But there's a big difference between creating a test vehicle that'll run in reasonably tame conditions and building a multi-million strong fleet of cars that can handle the volatility and randomness of the real world. One of the biggest challenges is putting the computer in charge of everything, including exception handling. By exceptions, we mean variable circumstances like bad weather, traffic violations and environmental hazards. Fog, snow, a deer leaping into the road – how does a fully autonomous vehicle interpret and react to these situations? When we take the driver out of the picture completely, automation complexity soars compared to lower-level systems. The software must handle everything. Rigorous testing is essential, but it won't be enough on its own. Alternative methods such as simulation, bootstrapping, field experience and human reviews will also be necessary to ensure the safety of the vehicle. For the time being, it means implementing each new capability carefully and gradually: hence why it will be a good few years before we see Level 5 vehicles on the road. Global perceptions of autonomous car technology How does the general public feel about the onset of autonomous cars? Generally, perception is positive, but there's still a way to go. Not everyone is convinced. The main bone of contention is safety. Autonomous car manufacturers must prove beyond doubt the safety of their vehicles before they can hope for widespread adoption. One survey conducted across China, Germany and the US found that drivers want to decide for themselves when to let a car drive autonomously and when to take over. The survey also found that trust in autonomous cars is nearly twice as high in China. A [2] discovered that the higher the level of automation, the higher the doubt. There are still reservations about giving the vehicle total control, despite having a positive view on autonomous vehicles generally. Another factor playing on people's minds is cybercrime: fear of personal data falling into the wrong hands. People are generally happy to share data for safety reasons, but less so when it ends up being sold to related service providers.

C. *Machine Learning and Neural Networks.*

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

While A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes Thus, a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems. The connections of the biological neuron are modelled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modelling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information. As shown in Figure 1.

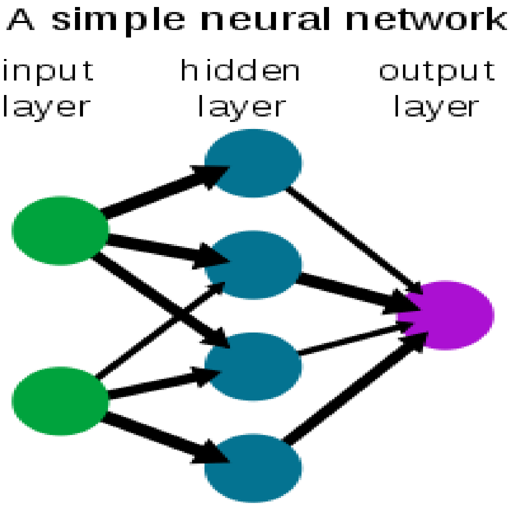


Figure 1 A simple neural network..

We used a dataset of common objects we see normally in our daily life in streets to train our neural network to be able to recognise these objects in any future image that would be our input to the neural network.

- The used classes are:

1. Cars
2. Human bodies
3. Buses
4. Motor Bikes
5. Trains

II. The Proposed architecture.

The proposed architecture is shown in Figure 2.

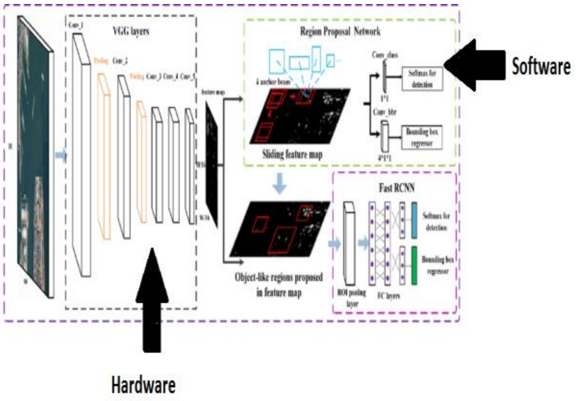
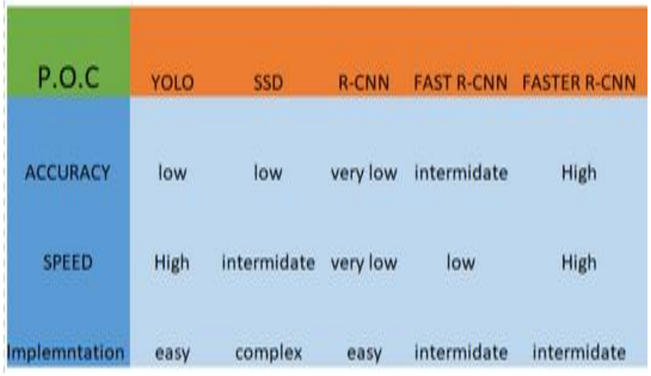


Figure 2 Block Diagram of the Project

Table 1 Comparison between Proposed CNN architectures



P.O.C	YOLO	SSD	R-CNN	FAST R-CNN	FASTER R-CNN
ACCURACY	low	low	very low	intermidate	High
SPEED	High	intermidate	very low	low	High
Implemntation	easy	complex	easy	intermidate	intermidate

In the literature, there are many architectures in the convolutional neural networks as shown in Table 1

In this work, we select the faster RCNN as the backbone for our implementation as it is the best according to autonomous driving constrains. Faster RCNN is the modified version of Fast RCNN. The major difference between them is that Fast RCNN uses selective search for generating Regions of Interest, while Faster RCNN uses "Region Proposal Network", aka RPN. RPN takes image feature maps as an input and generates a set of object proposals, each with an objectness score as output. The below steps are typically followed in a Faster RCNN approach: We take an image as input and pass it to the ConvNet which returns the feature map for that image. Region proposal network is applied on these feature maps. This returns the object proposals along with their objectness score. A RoI pooling layer is applied on these proposals to bring down all the proposals to the same size. Finally, the proposals are passed to a fully connected layer which has a SoftMax layer and a linear regression layer at its top, to classify and output the bounding boxes for objects. To begin with, Faster RCNN takes the feature maps from CNN and passes them on to the Region Proposal Network. RPN uses a sliding window over these feature maps, and at each window, it generates k Anchor boxes of different shapes and sizes. Anchor boxes are fixed sized boundary boxes that are placed throughout the image and have different shapes and sizes. For each anchor, RPN predicts two things: The first is the probability that an anchor is an object (it does not consider which class the object belongs to),Second is the bounding box regressor for adjusting the anchors to better fit the object. We now have bounding boxes of different shapes and sizes which are passed on to the RoI pooling layer. Now it might be possible that after the RPN step, there are proposals with no classes assigned to them. We can take each proposal and crop it so that each proposal contains an object. This is what the RoI pooling layer does. It extracts fixed sized feature maps for each anchor, then these feature maps are passed to a fully connected layer which has a SoftMax and a linear regression layer. It finally classifies the object and predicts the bounding boxes for the identified objects.

III. Implementation.

A. *Software Implementation.*

We know in the software industry that without an interface, libraries, and organized tools, software development proves a nightmare. When we combine these essentials, it becomes a framework or platform for easy, quick, and meaningful software development.ML frameworks help ML developers to define ML models in precise, transparent, and concise ways. ML frameworks used to provide pre-built and optimize components to help in model building and other tasks. There are famous frameworks commonly used as TensorFlow and PyTorch, we expected to use both, so we wanted to know the differences between them.

TensorFlow: TensorFlow is an open source deep learning framework created by developers at Google and released in 2015, It supports regressions, classifications, and neural networks like complicated tasks and algorithms. You can run it on CPUs & GPUs both. TensorFlow creates a static graph, you first must define the entire computation graph of the model and then run your ML model. It's more difficult than PyTorch to learn but it has larger community and easier to find solutions to your problem as it's older than PyTorch.

PyTorch: PyTorch is based on Torch and has been developed by Facebook, it's used by Facebook, IBM, Yandex, and Idiap Research Institute. PyTorch believes in a dynamic graph, you can define/manipulate your graph on-the-go. This is particularly helpful while using variable length inputs in RNNs. Torch is flexible and offers high-end efficiencies and speed. It offers a lot of pre-trained modules. It's easier to learn but it is new compared to TensorFlow, so it has smaller community than TensorFlow.

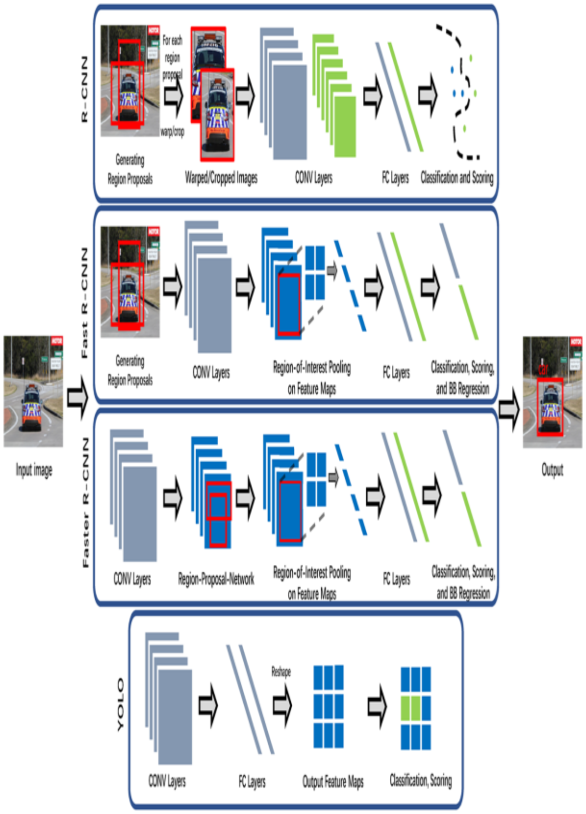


Figure 3 Block diagram for software identification of objects.

We will be simulating the Faster-RCNN architecture using google colabs as this will fasten the training using google servers

we will use a unique custom dataset which is a subset of the VOCtrainval_11-May-2012 that contains objects that is present on the road normally

This data set is found in the dataset folder where it is divided into 2 folders one which contains the images and the annotation files(XML format) and the other folder contains the tfrecord format that will be used in training (explained later)

Roboflow makes managing, preprocessing, augmenting, and versioning datasets for computer vision seamless. Developers reduce 50% of their code when using Roboflow's workflow, automate annotation quality assurance, save training time, and increase model reproducibility.

This website is used to change from one format to another to the annotation files and this will be used to generate the formats we will need in training in case you need to train on your custom dataset (note the dataset attached in the dataset folder already has the correct format) for the full tutorial on how to work with Roboflow with your custom dataset

We used the TensorFlow architecture in [4] to train our custom dataset and we extracted the weights in a pb file format that we will use to put in the architecture we will construct in hardware in inference mode

B. *Hardware implementation.*

Computations in neural networks are performed traditionally with floating point using either GPU or CPU but when it comes to hardware implementation, using floating point calculation doesn't just make it slower due to the difficulty of controlling the mantissa and the exponent for various operations but also consumes the resources, and to avoid this problem we use fixed point calculations that use much less resources and doesn't affect the performance[1].

Table 2: FPGA resource consumption comparison for multiplier and adder with different types of data.

	Xilinx Logic			
	Multiplier		Adder	
	LUT	FF	LUT	FF
Fp32	708	858	430	749
Fp16	221	303	211	337
Fixed32	1112	1143	32	32
Fixed16	289	301	16	16
Fixed8	75	80	8	8

we use the high-level synthesis based design approach that allows quick development. We use Vivado HLS tool and as shown in figure 4 the flow of the design is to write codes in high level language as C or C++ with considering some limitation and the tool generates synthesizable VHDL and Verilog codes and simulation wave forms can be obtained, we can target zynq FPGA directly, Specify the type of memory easily (FIFIO ,ROM ,BRAM , RAM)

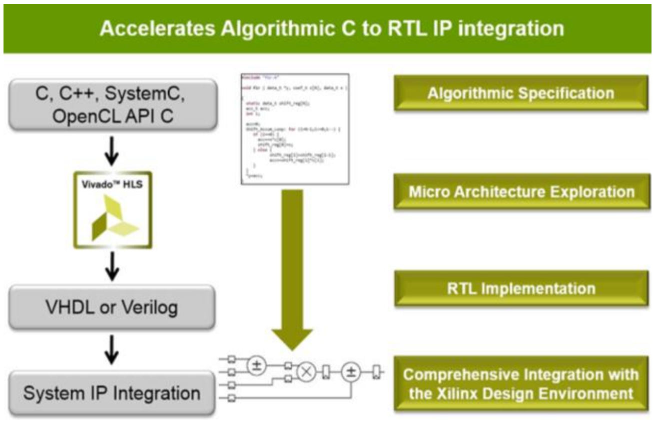

Accelerates Algorithmic C to RTL IP integration
C, C++, SystemC, OpenCL API C
Vivado HLS
VHDL or Verilog
System IP Integration
Algorithmic Specification
Micro Architecture Exploration
RTL Implementation
Comprehensive Integration with the Xilinx Design Environment

Figure 4 : Flow of the HLS design approach using (Vivado HLS)

Using directives, we can easily pipeline the codes, give estimations to frequency, clock cycles and resources utilization, Compare different solutions at time.[2]

Test the generated code against the C++ testbench and Useful flags inside the Verilog that help understanding the operation

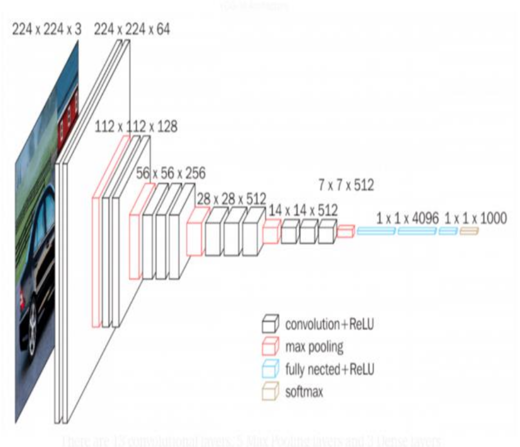


Figure 5 Block diagram of layers of VGG-16.

- As shown in figure 5 the architecture is divided into the following layers:

1) Convolution Layer.

Inputs:

Inputs of this layer are as follows:

three-dimensional array, two of them represents the size of the image or the feature map which depends on in which layer we are, and the third dimension is the number of channels, for example, the first layer always takes a 3 channels image(RGB).

four-dimensional array which represents number of kernels, number of channels, and two dimensions for the size.

One-dimensional array of the bias of every kernel applied.

Outputs:

The output is a three-dimensional array of the feature maps which is passed to Relu and Max-pooling layers afterwards. Figure 5 shows the simulation result of convolution layer.

2) Padding.

Inputs:

This is implemented before the convolution, so the input is either the image or the feature map and the main aim of this it is to preserve the image size after convolution so that we stick to the vgg16 CNN architecture.

Outputs:

The output is the padded image and then it will be passed to the convolution layer.

3) Max Pooling Layer.

Inputs:

Retrieve the output of the convolution layer after the Relu operation then halve each dimension by 2 which reduce the number of values to one quarter of the original number (by using a 2x2 window with stride of 2 to avoid any overlap)

Outputs:

The output is the feature maps that contain the maximum value of each four-pixel values in the original input, their number will be equal to the number of the filters and they will be passed to the next convolution layer.

4) RELU Layer.

Inputs:

This layer applies the Relu operation which is nonlinear operation used to reduce the number of deep layers in the CNN and it basically operates on each value and remove all negative values from the output feature map before Maxpooling layer.

5) Integration of (Padding-Convolution –Max pooling).

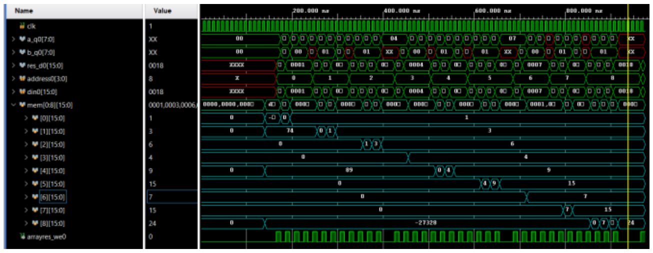
This layer is integration between the previous layers which work as first layer of vgg16 where all operations of Padding convolution and Maxpooling are applied after each other in this order

- Start the function by padding the input image so the feature map after convolution is same size of input image
- do the convolution with one image RGB , two filters with three channels each
- do max pooling 2x2 which reduce the output feature map size by 2.

C. An illustrative example

Matrix multiplication

Matrix multiplication is the first project we experimented using the Vivado tool as it is one of the main operations in the project.



Name	Value
clk	1
a_q0[7:0]	XX
b_q0[7:0]	XX
res_d0[15:0]	0018
address0[3:0]	8
din0[15:0]	0018
mem[0:8][15:0]	0001,0003,0006,
[0][15:0]	1
[1][15:0]	3
[2][15:0]	6
[3][15:0]	4
[4][15:0]	9
[5][15:0]	15
[6][15:0]	7
[7][15:0]	15
[8][15:0]	24
arrayres_we0	0

Figure 6 Matrix multiplication Simulation Result

In fig 6 the memory signals in the aqua color is output array that has the res_d0 saved in it. It shows clearly that it is the expected output. By tracing the wave form of inputs and the result the timing of everything goes as expected. This operation is basic one in the convolution layer and it gave us a good insight on how to use the tool power like pipelining, optimization techniques and memory implementation. In addition to that the report generated from tool about the resources needed on the FPGA to carry out this process also by referring to table 2 it shows clearly the estimated latency of multiplying 3x3 matrix and generating the output matrix and save it in a memory.

V. Validation and Results.

A. Software Results.

The results of the training in python in inference mode are shown is Figure 7 a,b and c

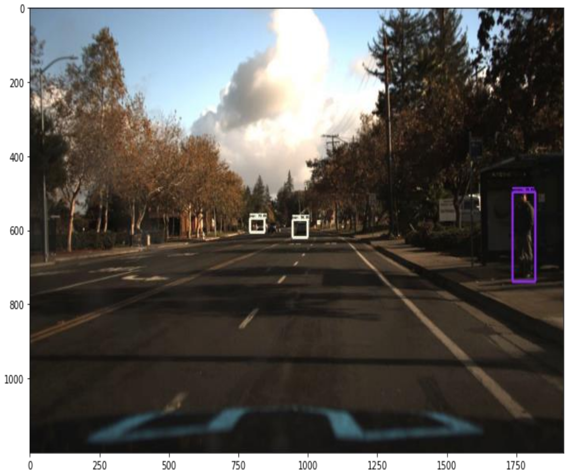


Figure 7 a) object detection

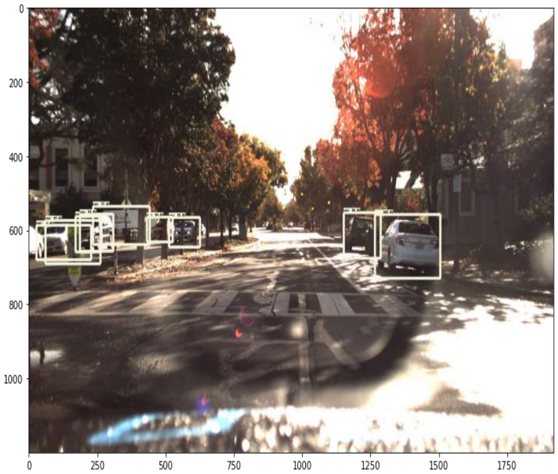


Figure 7 b) object detection

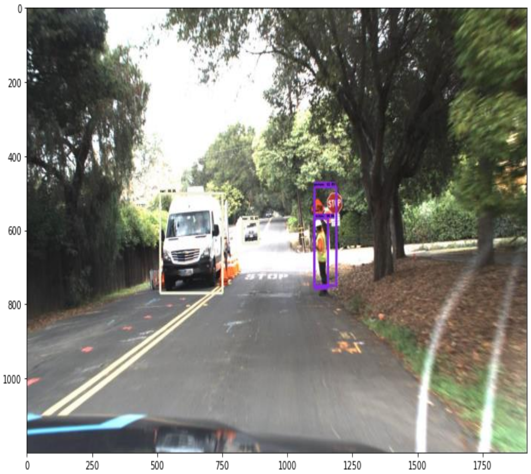


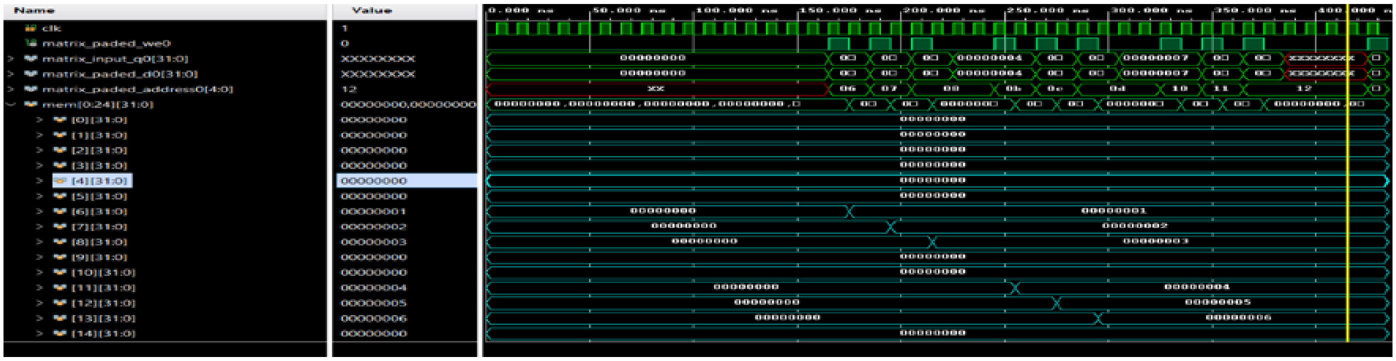
Figure 7 C) object detection

B. Hardware Results.

1) Padding.

By referring to the Figure 8 it shows that the function output is just as expected and after the required clock cycles the correct values are on the output .Furthermore the output is being saved in the memory once they are ready on the output and before the memory address changes to the next one the true value should be ready to write in the memory. Tracing this waveform enabled us to ensure that the function worked correct

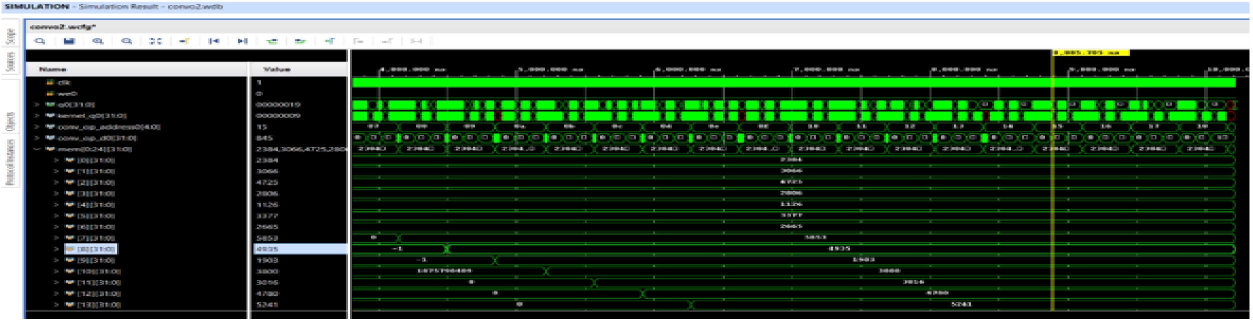
Figure 8 Padding wave form output



```
Name                                Value
  clk                               1
  matrix_paded_we0                  0
> matrix_input_q0[31:0]             XXXXXXXX
> matrix_paded_d0[31:0]             XXXXXXXX
> matrix_paded_address0[4:0]        12
v mem[0:24][31:0]                   00000000,00000000
    > [0][31:0]                     00000000
    > [1][31:0]                     00000000
    > [2][31:0]                     00000000
    > [3][31:0]                     00000000
    > [4][31:0]                     00000000
    > [5][31:0]                     00000000
    > [6][31:0]                     00000001
    > [7][31:0]                     00000002
    > [8][31:0]                     00000003
    > [9][31:0]                     00000000
    > [10][31:0]                    00000000
    > [11][31:0]                    00000004
    > [12][31:0]                    00000005
    > [13][31:0]                    00000006
    > [14][31:0]                    00000000
```

2) Convolution.

After the padding function comes the convolution which do the most calculations and need the largest time and resources. Figure 9 shows the simulation of this function on hardware. The write signal is only high when it is correct value is being set on the output


SIMULATION - Simulation Result - convo2.wdb

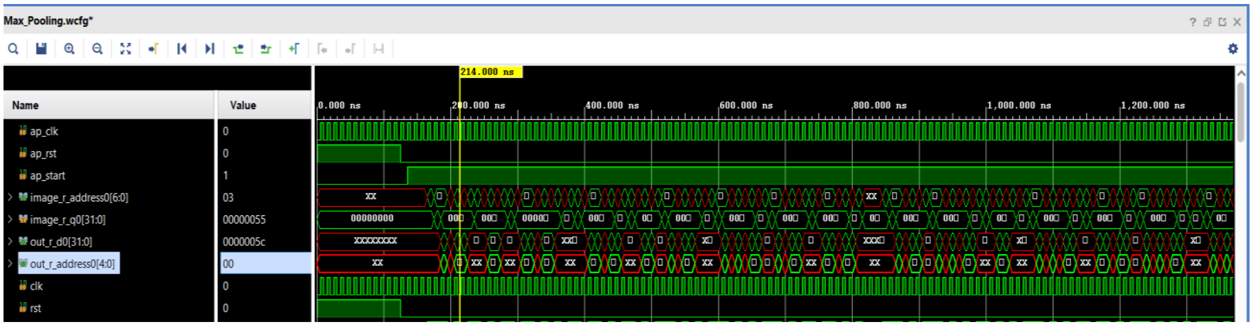
convo2.wcfg*

Name	Value
clk	1
we0	0
q0[31:0]	00000019
kernel_q0[31:0]	00000009
conv_op_address0[4:0]	15
conv_op_d0[31:0]	845
mem[0:24][31:0]	2384,3066,4725,280
[0][31:0]	2384
[1][31:0]	3066
[2][31:0]	4725
[3][31:0]	2806
[4][31:0]	1126
[5][31:0]	3377
[6][31:0]	2665
[7][31:0]	5853
[8][31:0]	4935
[9][31:0]	1503
[10][31:0]	3800
[11][31:0]	3056
[12][31:0]	4780
[13][31:0]	5241

Figure 9 Convolution wave form output

3) Max pooling.

The Maxpooling function differs from the previous functions as it doesn't go through the feature maps row by row, but it uses a window 2x2 which is dealing with two different rows at a time. So, in the wave form the output may go high impedance at some points but the output array is still acting as intended (Figure 10)

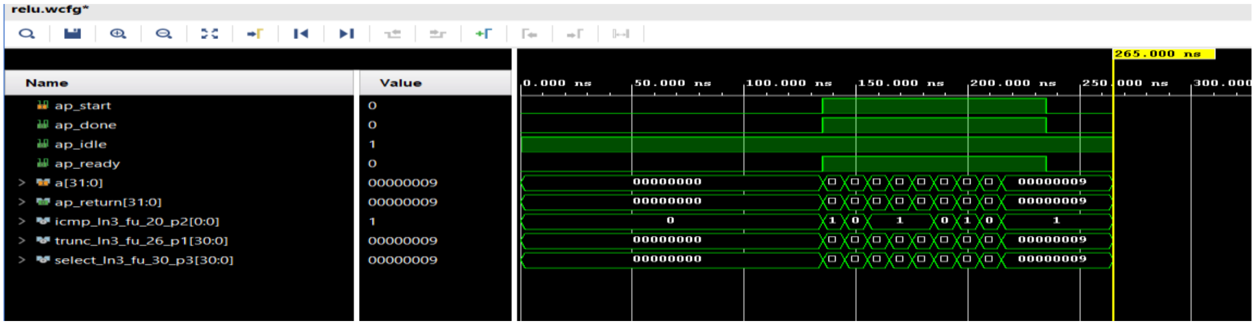

Max_Pooling.wcfg*

Name	Value
ap_clk	0
ap_rst	0
ap_start	1
image_r_address0[6:0]	03
image_r_q0[31:0]	00000055
out_r_d0[31:0]	0000005c
out_r_address0[4:0]	00
clk	0
rst	0

Figure 10 Max pooling wave form output

4) RELU.

The RELU simulation was the easiest as it only converting the negative values to zero and leave The other values as it is. Figure 11 shows the simulation of it on Vivado tool.

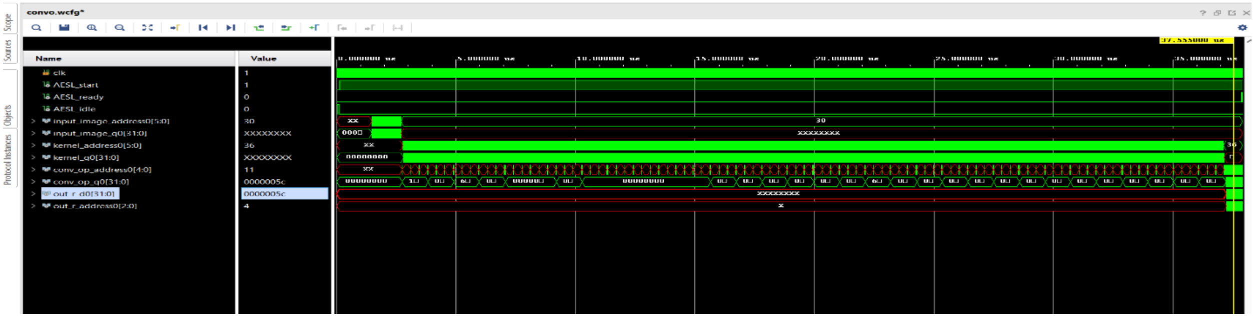

relu.wcfg*

Name	Value
ap_start	0
ap_done	0
ap_idle	1
ap_ready	0
a[31:0]	00000009
ap_return[31:0]	00000009
icmp_ln3_fu_20_p2[0:0]	1
trunc_ln3_fu_26_p1[30:0]	00000009
select_ln3_fu_30_p3[30:0]	00000009

Figure 11 RELU wave form output

5) Integrated layer.

After simulation of all these layers we needed to integrate them in one function to ensure that all of them are working smoothly with each other especially the interface between them .the simulation in figure 12 shows it take a lot of time before the output is being ready and calculated the explains the high impedance value at the output last this long



Name	Value
clk	1
AESL_start	1
AESL_ready	0
AESL_idle	0
input_image_address0[5:0]	30
input_image_q0[31:0]	XXXXXXXX
kernel_address0[5:0]	36
kernel_q0[31:0]	XXXXXXXX
conv_op_address0[4:0]	11
conv_op_q0[31:0]	0000005c
out_r_d0[31:0]	0000005c
out_r_address0[2:0]	4

Figure 12 Integrated layer wave form output

VI. Performance evaluation

A) performance metrics

Table 3 Latency of each Layer

	Latency(cycles)		Latency(absolute)	
	min	max	min	min
Convolution	869	869	8.690 us	8.690 us
Padding	79	79	0.790 us	0.790 us
Max pooling	206	206	206	206
RELU	0	0	0	0
Integration of padding & convolution & max pooling	3575	3789	37.570 us	37.890 us

The metrics of performance in our design are the speed mainly of performing the convolution calculations as it get much more time than other layers table 4 indicates the latency of different layers it shows clearly that convolution is the most time consuming .noting that cascading layers together and increasing the latency time that must be taken in consideration as we are not using the real image size yet nor the exact number of filters in VGG16

B) the utilization in FPGA

The utilization of the design when simulating it to know how much resources it will use from the available on the FPGA up till now the design is not complete, so we are using very small resources (table 5) .

Table 4 Comparison between Utilization of each Layer

	Total					Utilization(%)				
	BRAM_18	DSP48E	FF	LUT	URAM	BRAM_18K	DSP48E	FF	LUT	URAM
Convolution	0	3	476	757	0	0	0	0	0	0
Padding	0	0	59	277	0	0	0	0	0	0
Max pooling	0	0	471	1541	0	0	0	0	0	0
RELU	0	0	0	49	0	0	0	0	0	0
Integration of padding, convolution and max pooling	1	3	944	2186	0	0	0	0	0	0

4.6. Comparison with related work

4.7. Conventional versus proposed

4.8. Others work versus proposed

4.9. Limitations

4.10. Demo (if applicable) ~ 3 min

link

5 .Conclusions

References

[1] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2017. [DL] A Survey of FPGA-Based Neural Network Inference Accelerator

[2] J. Duarte et al., “Fast inference of deep neural networks in fpgas for particle physics,” arXiv preprint arXiv:1804.06913, 2018.

[3] J. Qiu et al., “Going deeper with embedded fpga platform for convolutional neural network,” in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA ’16. New York, NY, USA: ACM, 2016, pp. 26–35. [Online].

[4] Joseph Fiowa ,TensorFlow-object-detection-faster-rcnn,(2013),GitHub