TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Computer Systems

Lembitu Valdmets 178205IASM

Running YOLO Artificial Neural Network on ZedBoard Using Xilinx DNNDK

Masters Thesis

Technical Supervisor
Madis Kerner
PhD student
Academic Supervisor
Kalle Tammemäe
assoc. professor

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

Lembitu Valdmets 178205IASM

YOLO tehisnärvivõrgu rakendamine ZedBoard arendusplaadil Xilinx'i DNNDK arenduskeskkonnas

Magistritöö

Juhendaja
Kalle Tammemäe
dotsent
Kaasjuhendaja:
Madis Kerner
doktorant

Tallinn 2020

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author:	Lembitu Valdmets	.... (signature)
Date:	Month Day, Year	

Annotatsioon

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 45 leheküljel, 10 peatükki, 17 joonist, 5 tabelit.

Viimastel aastatel on kasvanud tehisnärvivõrkudel baseevuvate algoritmide kasutamine objektide tuvastamisel piltidel ja videos. Kasutades erinevaid kiirendeid on algoritme võimalik kõrgelt paralleliseerida, tehes võimalikuks nende tuvastuskiiruse suurendamise. Käesolevas lõputöös kasutatakse Xilinx'i universaalset FPGA'l baseeruvat kiirendit, nimega "Deep learning Processing Unit", mille kasutamisel ei pea disainer omama kõrgel tasemel teadmisi riistvaralisest disainist.

Käesoleva lõputöö eesmärgiks on demonstreerida, kuidas saab Xilinx'i poolt välja antud tarkvaraarendustööriistadega teostada "You Only Look Once" (YOLO)-nimelist objektituvastus tarkvara kiirendamist FPGA abil, kasutades selleks Xilinx'i Deep Neural Network Development Kit (DNNDK) arenduskeskkonda.

Peamiseks motivaatoriks lõputöö valmimisel oli asjaolu, et mitmed ülikoolid, kaasaarvatud Tallinna Tehnikaülikool, kasutavad oma õppeprotsessides Avneti arenduskomplekte, nimega ZedBoard". Neid kasutatakse peamiselt riistvaralähedase programmeerimise ja kiipsüsteemide disainimise õpetamiseks.

Tulemuseks valmis süsteem, mis kasutab eelnevalt mainitud metoodikaid, et efektiivselt kiirendada YOLO tööd. Lisaks lahenduse kirjeldusele on välja toodud samm-sammuline juhend antud süsteemi koostamiseks ja käivitamiseks etteantud vahenditega. Oluliseks tulemuseks saadi, et süsteem töötas kiirendatult "ZedBoard" peal. Keskmiseks ühe pildi töötlusajaks oli 1.5 sekundit, mis ei lase seda lahendust kasutada enamiku reaalajaliste lahenduste juures. Peatükk 7 sisaldab arvutuslikku selgitust, mille abil saab ennustada, kui palju ujuvkomaarvutusi on vaja sooritada ühe pildi hindamiseks.

Käesolevat lahendust saab edasi arendada, suurendades selle töökindlust ja lisades tähendavaid funktsioone vastavalt kasutaja vajadustele ja soovidele. Täiendavad ettepanekud on välja toodud 9. peatükis.

Abstract

The thesis is in English and contains 45 pages of text, 10 chapters, 17 figures, 5 tables.

In recent years there has been an increase in using convolutional neural networks for solving object detection and classification problems. By using different accelerators it is possible to rapidly speed up inference and training with these algorithms. Xilinx has created an universal deep learning processing unit, called DPU, which is configurable hardware Intellectual Property (IP) used in multiple FPGA-based systems. The DPU offers high performance without the need for developers to have an expertise in hardware design.

This thesis is a proof of concept of running You Only Look Once (YOLO) artificial neural network based object detection algorithm on Avnet ZedBoard using Xilinx's Deep neural Network Development Kit (DNNDK).

One of the main motivations for this thesis is to provide useful material for educational purposes. This thesis also includes guide on replicating a proposed solution.

One of the big obstacles was to reconfigure many aspects to make the proposed solution more functional, because given algorithm is designed to be run on more computationally capable platforms, e.g. Zynq® Ultrascale and Zynq® Ultrascale+ platforms from Xilinx.

The main result is a working object detection algorithm running on Avned ZedBoard development platform, which are used in many universities. Solution uses programming tools provided by Xilinx to implement inference acceleration on artificial neural networks in different applications. This makes the proposed solution easy to modify, improve and deploy.

In order to implement the given solution for academic purposes there is some further development that can be done, mainly evolving system stabilisation and integrating different aspects into a single project. Ideas and proposals for further development and modifications are provided in the chapter 9.

List of abbreviations and terms

CPU	Central Processing Unit
DPU	Xilinx® Deep Learning Processing Unit
DNNDK	Deep Neural Network Development Kit
ARM	Advanced RISC Machines
YOLO	You Only Look Once
AI	Artificial Intelligence
mAp	Mean Average Precision
ACAP	Adaptive Compute Acceleration Platform
API	Application Programming Interface
IP	semiconductor Intellectual Property
SoC	System on Chip
DMA	Direct Memory Access
ReLu	Rectified Linear Unit
LeakyReLu	Leaky Rectified Linear Unit

Table of Contents

List of Figures

List of Tables

1. Introduction

This thesis is a proof of concept, including tutorial on running YOLOv3 object detection algorithm on Avnet ZedBoard.

Running artificial neural networks on different types of hardware-based accelerators is a topic extensively discussed in many years [1]. Artificial neural networks are widely used for object detection and classification tasks. Contests are being held where object detection algorithms compete against each other annually. One of the examples are COCO dataset [2] based competitions where in recent years a new object detection has been risen amongst others that delivers higher detection speed with comparable accuracy results to other widely used neural networks, called You Only Look Once (YOLO)[3]. To the best knowledge of the author this relatively new object detection algorithm has not been currently implemented in courses in Tallinn University of Technology.

1.1 Motivation

Many universities worldwide are using different development boards for rapid prototyping and as a practice tools for students to learn on. For example Tallinn University of Technology uses Avnet ZedBoards in course "System-on-chip design" [4] to educate students about basics of system-on-chip designs, architectures, internet integration and many more.

YOLOv3 [3] is relatively new and moderately discussed object detection algorithm used in many fields. There will be more information about its usages in chapter 2. The research amount of implementing YOLO on FPGA is lower considered to the rest of research with given algorithm. There is a lot more research that can be achieved, especially implementing YOLO on student-friendly platforms. YOLO algorithm on Avnet Zedboard could be used in educational purposes. The proposed solution would help students more efficiently understand how artificial neural networks can be accelerated on FPGA and therefore how different applications can be accelerated on hardware basis.

1.2 Problem formulation

The main objective of this thesis was to briefly analyze Xilinx's Vitis Unified Software Platform and use it to accelerate inference of YOLOv3 artificial neural network using Avnet Zedboard. If this can not be done as stated, an alternative solution must be searched to implement a result as similar as possible. Main aspect for alternative solution would be ease of use from widely used developing tools and environment. If solution is found, it should be analyzed and its results should be compared to similar solutions.

1.3 Contribution of Hypothesis

The initial hypothesis for proposed solution was as follows: Vitis AI can be used to easily deploy neural network applications accelerated by DPU without the need to modify hardware designs. Test will be done on YOLOv3 artificial neural network model on Zedboard to prove its ease of use and simplicity. If Vitis AI works as assumed it means accelerating inference of artificial networks has never been so easy and can be used to rapidly speed up prototyping of new embedded solutions which require real time image and video processing.

1.4 Thesis organization

This thesis is formulated in following chapters:

- Background chapter briefly lists some similar solutions that have been previously proposed. Each solution contains short analysis from author about feasibility of using those solutions to solve current problem. This provides a brief overview what has been done previously.
- Early experimental chapter describes workflow and early experimental part of thesis before final solution was found. There is also a list of tools that was meant to be used for finding the solution, but were neglected with reasons given.
- Used tools chapter lists and explains all the main tools that were used in the solution.
- Thesis solution explains the proposed solution and how it works and gives an overview of its structure.
- Guide chapter provides guide for everyone with required tools to have an ability to replicate given solution and implement YOLOv3 artificial neural network on Avnet Zedboard.
- The analysis chapter explains how proposed solution was analysed and what were the experimental results.

- Conclusion takes the main analysis results and provides a quick overview about these.
- Future work points out what could be improved in proposed solution.

2. Background

This chapter mainly lists and explains the earlier attempts running YOLO on similar platforms. The task was to find similar solutions to current problem description. The closest one was found where YOLOv2 CNN was implemented on ZedBoard using custom accelerator[5].

YOLO as an object detection algorithm can be widely used in various applications, e.g. automatic detection of Melanoma[6], fire hotpots detection on CCTV using given algorithm for high buildings evacuation [7], pedestrian detection [8], traffic counting systems [9] and real-time face detection [10]. These examples show that YOLO is widely capable object detection system that can also be used in resource-constrained environments, making idea of accelerating its inference on FPGA highly feasible.

In journal Article "Design and implementation of YOLOv2 Accelerator Based on Zynq7000 FPGA Hetereogeneous Platform" by Chen, Chenchai Zhilei, Xia Jun from Jiangnan University, China there was a successful attempt to implement custom accelerator for running YOLOv2 convolutional neural network on ZedBoard. Their experimental results show that the performance of 30.15GOP/s was obtained from ZedBoard. In given solution a YOLOv2's workload was rated 29.47GOP on ZedBoard and interference time for single pass was measured 1.13442897 seconds. [5]. This was until this day one of the few articles found in which any version of YOLO neural network is implemented on ZedBoard. This provides good source for comparison with Xilinx's solution. In comparison to current requirements this solution is not sufficient, because for YOLOv2 a custom accelerator was designed, making usability of this processing unit very limited: it only runs YOLOv2. Current solution needs to be more universal, making available to inference different convolutional neural networks in a more beginner friendly approach.

In article by Hiroki Nakahara and Masayuki Shimoda and Shimpei Sato there was similiar attempt accelerating YOLOv3 convolutional neural network on different hardware platforms. Tested solution was proposed mixed-precision YOLOv2 on the Xilinx Zynq UltraScale+ MPSoC zcu102 evaluation board, which has the Xilinx Zynq® UltraScale+ MPSoC FPGA. [11]. Compared to ZedBoard this system is more capable on computing performance (274,080 vs 53,200 LUTs; 2,520 vs 220 DSP). The results were successful and measured 28ms inference time on a single picture while consuming 4.5W of electrical power. Is was not shown what was computational performance of proposed solution so that could only be estimated from workload of implemented CNN model and execution time.

3. Early Experiments and Results

The current chapter explains early experimental stages of this thesis and explains why initial task description changed and how proposed solution was found. The main part consists of brief explanations about Vitis and Vitis AI and their influence on finding proposed solution.

3.1 Vitis

Vitis, also called Vitis Unified Software Platform is a software platform by Xilinx that enables the development of embedded software and accelerated applications on heterogenous Xilinx platforms including FPGA's, SoCs, and ACAPs. It provides unified programming model for accelerating Edge, Cloud and Hybrid computing applications. Providing embedded programming versatility never seen before, Vitis could become tool to improve development flow and speed of embedded and accelerated applications. It integrates high-level frameworks, making able to use C, C++ or Python to develop hardware accelerated applications. It also has low-level API's to more control over implementation if needed. [12]

Vitis was inspected because as of during development of thesis (April 2020) Vitis was released less than year ago and analysis, whether it can be used to simplify development flow on Avnet Zedboard, was not completed. It was found out that Vitis itself was not designed for AI inference on Xilinx hardware platforms. Nonetheless, it is still a viable platform to use while developing other applications. It must be noted that currently Vitis does not have "out-of-the-box" support for Avnet Zedboard and is meant to be operated on more modern platforms like Zynq$^{®}$ UltraScale+. There are workarounds around the web that enable to use Vitis with Zedboard, if needed. One of those workarounds are posted in Digilent Technical Forums [13].

3.2 Vitis AI

Vitis AI is a development platform for AI inference on Xilinx hardware platforms. It mainly consists of optimized IP (DPU), tools, libraries, models and example designs. As claimed in their website Vitis AI is designed with high efficiency and ease of use in mind[12].

Vitis AI uses dockers to deliver its tool to developers, making installing software more convenient compared to its previous version, DNNDK. Development flow in Vitis AI consists of running previously written shell scripts in predetermined succession.

Vitis AI is said to support Avnet Zedboard, but at launch (Vitis AI 1.0), it only supports ZCU102, ZCU104, U200, U250 platforms . There is a high probability that Vitis AI will have full support for Avnet Zedboard in upcoming year (2020-2021) [14]. For the time being, it is recommended to use Xilinx DNNDK to develop and run hardware-accelerated object classification applications on Zedboard [15]. Therefore, decision was made to use DNNDK to find proposed solution for the given task. More information about DNNDK can be read in the upcoming section 4.1.1.

3.3 Resnet-50 on ZedBoard

One of the first steps was to make sure DNNDK can be used to run DPU accelerated applications on Avnet ZedBoard. This is claimed to work by Xilinx in its long form answer record, named "73058 : Resnet-50 CNN application implemented on ZedBoard using Vivado and Petalinux 2019.2" [16]. After attempts of following the guide the result was successful and Resnet-50 worked as intended. This meant YOLOv3 could in theory be implemented as well.

DNNDK supports 2 deep learning frameworks: Tensorflow and Caffe[17]. Since YOLOv3 is not written in Caffe deep learning framework like Resnet-50 has been, it was necessary to convert it. Xilinx provides guide and tools to convert models from Darknet to Caffe, including different versions of YOLO [18]. This guide also provides tools to compile Caffe models to DPU, but lacks functionality to compile to ZedBoard. The problem was solved by using compiler that was previously proved to work with Resnet-50 provided by DNNDK. Information how to setup tools can be found at guide section 6.1.

3.3.1 YOLOv3 on Zedboard

To get YOLOv3 running on Zedboard with provided tools, emphasis was to modify as little as possible. In first attempts to convert YOLOv3 to Caffe and compile to application readable hardware configuration file (.elf) it was found that YOLO uses LeakyReLu as activation function in many of its layers and default hardware configuration of DPU on ZedBoard does not support it. LeakyReLu is a type of activation function used widely in different YOLO network models, more info in 4.2. LeakyReLu could be enabled by turning on its functionality in hardware configuration of DPU IP. This finally enabled the use of YOLOv3 on Zedboard as intended. More information of solution can be found in chapter 5.

4. Used Tools and Prototyping Environment

This chapter describes tools that were used to implement proposed solution. Under each tool there is a brief explanation about its background and how it was used in proposed solution.

4.1 Xilinx

Xilinx is a technology company offering wide range of computer hardware and software solutions and is primarily known for supplying programmable logic devices. [19] It was founded in Silicon Valley in 1984 and is currently supplying its products globally. Xilinx provides of most of the tools used in this solution.

4.1.1 Deep Neural Network Development Kit

Xilix Deep Neural Network Development Kit is a predecessor of currently released Vitis AI including tools to develop applications that are able to use hardware acceleration for convolutional neural network inference. It provides a unified solution for deep neural network inference applications by providing pruning, quantization, compilation, optimization, and runtime support. [17]. It includes optimized tool chains for speeding up development, lightweight programming API's. It is also advertised by Xilinx to be easy-to-use with gradual learning curve. There are currently 2 main frameworks that are natively supported by DNNDK: Tensorflow and Caffe. Models from other frameworks must be converted for use in DNNDK[17].

4.2 YOLO

YOLO is an object detector algorithm[3]. Compared to standard object detection algorithms YOLO being an artificial neural network detects and classifies all objects on a image in a single pass, hence the name YOLO "You only look once". This architecture claims to work really fast compared to competing neural networks like RetinaNet-50 while retaining similar precision as competitors[3]. YOLOv3 is an incremental step forward bringing new features to the algorithm and optimizing its performance [3].

YOLOv3 in most of its convolutional layers uses LeakyRelu functions as its activation layer. LeakyRelu "A unit employing the rectifier is also called a rectified linear unit ReLU" is an activation function used in artificial neural networks. It has a *raw* output when input is greater than 0 and *0.01input* when input is less than 0 [20].

YOLOv3 is a well studied and publicised object detection algorithm and therefore it was chosen for the current solution. In addition, it is open source and easily customisable. More information on usages of YOLO in chapter 2.

4.3 Zedboard

Avnet ZedBoard is a development kit developed for designers interested in exploring designs using the Xilinx Zynq-7000 all programmable SoC. The board contains all the necessary interfaces and supporting functions to enable wide range of applications. This board is efficient for rapid prototyping and proof-of-concept development [21].

Digilent offers Academic pricing of its products for many customers. Zedboard is widely used in academic studies and prototyping because of its comparatively low price and high availability for various institutions [22].

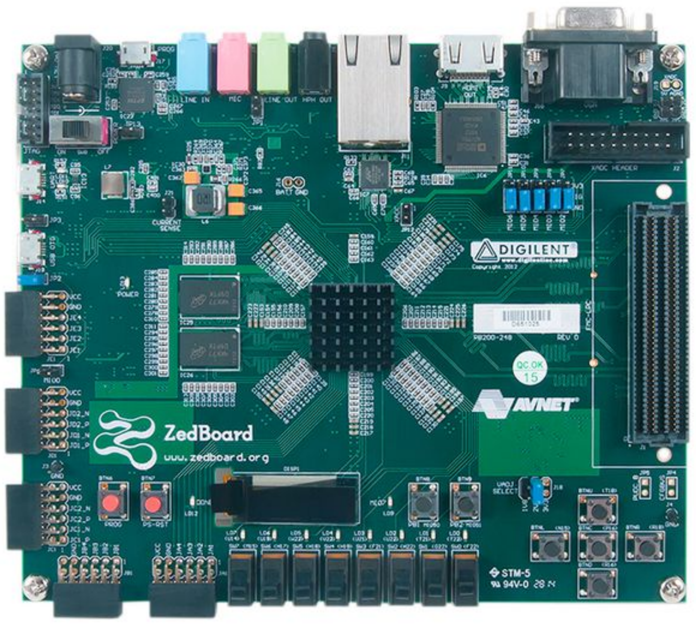


Figure 1. *Top down view of Avnet ZedBoard (source: Avnet)*

4.4 User hardware

This lists hardware that the thesis solution was developed and tested on. All of the necessary tools were also installed on this configuration. This information can be used for reference when porting to another system.

4.4.1 Host PC

Host PC used in this process was desktop PC. Its main specifications include:

- Processor: Intel I7-7700K
- RAM: 16GB DDR4
- Storage device: 1TB Samsung EVO 860 SATA SSD
- Operating system: Ubuntu 18.4.3 LTS

5. Thesis solution

This chapter describes overall behaviour and structural characteristics of proposed solution. Most of the work is based on code samples provided by Xilinx, manuals by Xilinx and community forum posts at Xilinx Community Forums.

The proposed solution is a complex system based on structure provided by Xilinx Deep Neural Network Development Kit (DNNDK) [17] meant to be used on Avnet Zedboard [23]. This description further explains proposed system in hardware, software and application aspects.

The framework used in this solution is based on DNNDK framework. To illustrate, here is figure of DNNDK toolchain.

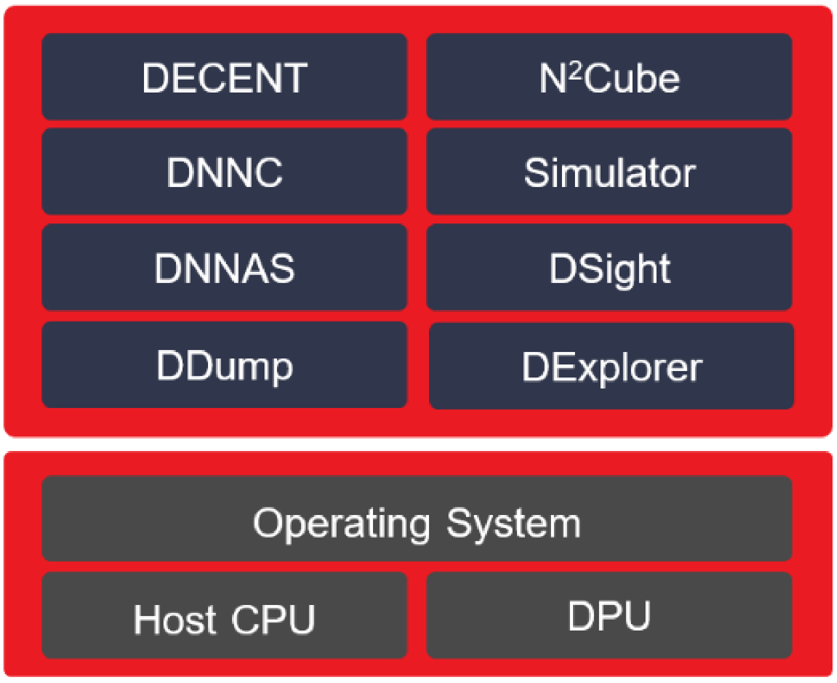


Figure 2. *DNNDK Toolchain (Owner: Xilinx)*

5.1 Hardware

Solution uses Avnet Zedboard to run YOLOv3 artificial neural network on. Zedboard has Zynq® 7020 System-on-chip in addition Dual-Core ARM Cortex-A9 processing system, 512MB DDR3 (32 x 128), 256 Mb Qspi Flash, 53200 LUT's and 106400 Flip-Flops [23]. FPGA is using mostly standard connections regarding processing system to essential inputs and outputs of system. The most notable difference is to use Xilinx Deep Processing Unit (DPU) to accelerate inference of YOLOv3 artificial neural network. The DPU used in this project is from sample system provided with Xilinx's guide 73058: "ResNet-50 CNN application implemented on a ZedBoard using Vivado and PetaLinux 2019.2" with small modifications [16]. The Parameters of DPU must be changed using Vivado design suite to use this with YOLOv3. "To use this design a Tcl script has been made that generates the Block diagram, the wrapper and the constraints file in Vivado design suite" [16]. It is also notable that DPU clock frequency is not a theoretical maximum of Z7020, which is 200MHz, but is 90MHz, because of excessive power demand that is not properly provided by the Zedboard. The 90MHz DPU frequency is recommended by Xilinx guide 73058 [16].

Here are listed parameters of DPU configuration used in this project:

Table 1. *DPU parameters in Vivado*

Parameter	**Value**
DPU Cores	1
Arch of DPU	B1152
Usage	low
Channel augmentation	enabled
AveragePool	enabled
Conv:	Relu Relu6 LeakyRelu
Softmax cores:	0

5.2 Software

Solution is built by following guidelines and structures provided by Xilinx DNNDK, mainly in guide 73058 [16, p. 3], running on Petalinux configured for this application. Petalinux is configured in a way that compiling applications on Zedboard is made possible. Petalinux configuration is explained further in these steps:

- Initial configuration. Project is initialized by .bsp file provided by Xilinx's guide 73058 [16, p. 3] which configures initial parameters, for example configuring device tree to include DPU and adding the DNNDK library.
- "Configuring the Petalinux project with hardware design"[16, p. 3]. This can be done with argument *–get-hw-description* more information can be read in guide section.
- Configuring Petalinux project by modifying list of packages that will be installed on Petalinux. This gives an ability to compile applications on Petalinux itself.
- Configuring *rootfs*. Petalinux now has many additional packages in it. This makes it running on RAMDisk not feasible because of the low amount of overall system memory on Avnet Zedboard. Therefore *rootfs* is located on a separate partition on SD card making possible to store additional packages, applications and even images under test.

After successful configuration, Petalinux can be built and exported to SD card. SD card can be plugged in Zedboard and tested for successful build of Petalinux. If configuration is working as intended, the Petalinux configured project can be saved to .bsp file making sharing and reusing Petalinux projects much easier [16, p. 8].

There are some additional configurations that must be done if applications that require acceleration from DPU are needed to be compiled on target. Although DNNDK as a library is installed via Petalinux, the tools for compiling applications are not, making incomplete installation of DNNDK on Zedboard. Runtime libraries work and precompiled applications work as intended. Additional libraries must be copied from DNNDK tools to target for compiling. More information about this is found in chapter 6.2.

5.3 Application

Using YOLOv3 in this solution was mainly accomplished by running the application that uses DPU for accelerating inference of a neural network. The application is based on the sample that was provided by Vitis AI guide which was about converting Darknet models to Caffe to Xilinx DNNDK. In this solution user application was written in C++ and uses common and specialized libraries to accomplish its tasks. One of the most specialized and important library is dnndk.h which adds functionality to communicate with DPU. DPU initialization, image acquisition, preprocessing, postprocessing and recording of the results is done in application and can be easily modified by changing source files on target device and recompiling application on Zedboard [18]. It is also possible to cross compile applications [17]. Inference of YOLOv3 is run on DPU, meanwhile application waits for DPU for an output of output layers. Output signal is sent by DPU as an *interrupt* to main processing system. *Interrupt* handling is done by DPU driver on Petalinux. To visualize image classification an output text was added to the application which prints a name of a detected object class onto a result image, located currently in the top left corner of detection - this functionality was not present in the sample solution.

Application uses .elf file that is specifically compiled for running YOLOv3 on DPU with previously mentioned configuration. This .elf file contains information of a neural network to simulate and how a neural network should behave. The file is sent to DPU for initialisation and is compiled by Deep Neural Network Compiler [17].

The sample code from Vitis Edge-AI tutorial was initially designed to process both pictures and video files. Since current objective was to focus on image recognition part of the application the video part of it was unedited. As the video part was untested on Zedboard with current configuration, it may not perform as it was supposed to. Nevertheless, image recognition part of the application is tested and its results are analyzed in analysis part of this thesis.

5.4 Behavioural notes during Experimentation

During experimentation with larger sets of images a behaviour was noted in which device freezes (becomes unable to respond) and resets itself. After restart system behaves normally again. This is probably caused because of the memory remapping in Petalinux configuration: there might appear a situation in which encapsulated applications share the same overlapping memory space, making possible to rewrite each other's data. If this occurs, kernel detects error and reboots. During experimental stage this problem had many repair attempts by author, but none successful. Therefore, as in current solution this behaviour can occur, the probability of the error was measured around 3% during each image pass.

6. Guide

This chapter contains mostly step-by-step guide on how to get YOLOv3 up and running on Avnet Zedboard using Xilinx DNNDK. This tutorial is divided into 2 main parts: tools and system. Copying large amounts of other tutorials into here is a decision based on combining all existing tutorials into single document that can be used from setting up the environment to running application on Zedboard.

6.1 Getting tools to run

This section is shorter than the later one, because most of the documentation for installing development tools are documented by the providers. It is possible to install and start using these tools without the use of this guide, however, here are some remarks author has made while installing named tools for the first time. To see what hardware platform solution was made and tested on, refer to 4.4.1.

6.1.1 Vivado 2019.2 HLx edition

Installation of Vivado HLx edition 2019.2 is well documented and installation instructions can be found at [24]. In this sample only Vivado is required when selecting which software to install.

6.1.2 Deep Neural Network Development kit

Deep Neural Network Development Kit (DNNDK) can be installed using quick start guide included with DNNDK guide (UG1327) [17]. This guide covers most of aspects installing DNNDK on host PC successfully, however, there are come remarks author points out to viewer which might make installation process easier. In quick start chapter, only "Setting up Host" part is necessary. Setting up ZedBoard will be done in next section of this guide 6.2.

6.1.3 Petalinux

Petalinux installation guide is written in "Petalinux Tools Documentation Reference Guide" [25]. Refer to this guide for installation: guide is well written and also has tips on troubleshooting while encountering any common issues while installing the Petalinux tools.

Setting up Petalinux working environment as guide suggests mainly launching a *settings.sh* or *settings.csh* script which also checks for Petalinux tools installation. Author points out that this script applies Petalinux tools path variables to only current terminal session and will not work in separate terminal window. In order to permanently add Petalinux Tools, these paths must be included in $PATH variable in Linux environment.

6.1.4 Edge AI tools for converting YOLOv3 to Caffe and DNNDK

This subsection is composed of many software packages that are needed in order to successfully convert YOLOv3 to Caffe and use shell scripts written to ease this process for developer.

It is important for the author to emphasise that the instructions may not be absolute for this installation. The hardware configuration 4.4.1 used in this project was prone to multiple errors due to the lack on instructions given officially by Xilinx [18]. Therefore, it is recommended to install the additional dependencies which might be missing in the current version of official installation guide. This might change as the official version improves.

Here is the list of official perquisites for the converter:

- Ubuntu OS 14.04 or 16.04. For more information, see chapter 1 in the DNNDK User Guide UG1327[17].
- DNNDK tools and image for evaluation boards (zcu102 used in this example). For more information, see Xilinx AI Developer Hub.
- Python 2.7 and its virtual environments for Ubuntu OS.
- The official YOLOv3 network model trained with COCO dataset is available in YOLO homepage [26]. Download the yolov3.weights file (around 248 MB) and place it in the 0_model_darknet folder.

Ubuntu OS was mentioned 16.04 in Official list, but Ubuntu 18.04 works as well in this application. Some additional dependencies that author noted were missing when trying to compile Darknet and Caffe:

- libprotobuf-dev profobuf compiler. These can be installed with following terminal commands:

  ```
  sudo apt-get install libprotobuf-dev protobuf-
      ↪ compiler
  sudo apt-get install libatlas-base-dev
  ```

  These are necessary for handling larger data amounts.

For further installation, refer to guide provided by Xilinx [18]. If encountered with problems during installation it is recommended to search for answers in Xilinx community forums at `https://forums.xilinx.com`.

6.2 Getting YOLO onto ZedBoard

This section and its subsections describe typical development flow on how to set up ZedBoard and how to get YOLOv3 running on ZedBoard. This guide is an extension of Xilinx's guide 73058 [16] with some additional steps noted. The Xilinx's guide 73058 itself is also copied here as stated in the introduction of this chapter. Each subsection has its perquisites indicating what user must have done or obtained prior following this guide.

Below is a simplified illustration of development workflow of getting YOLOv3 onto ZedBoard.

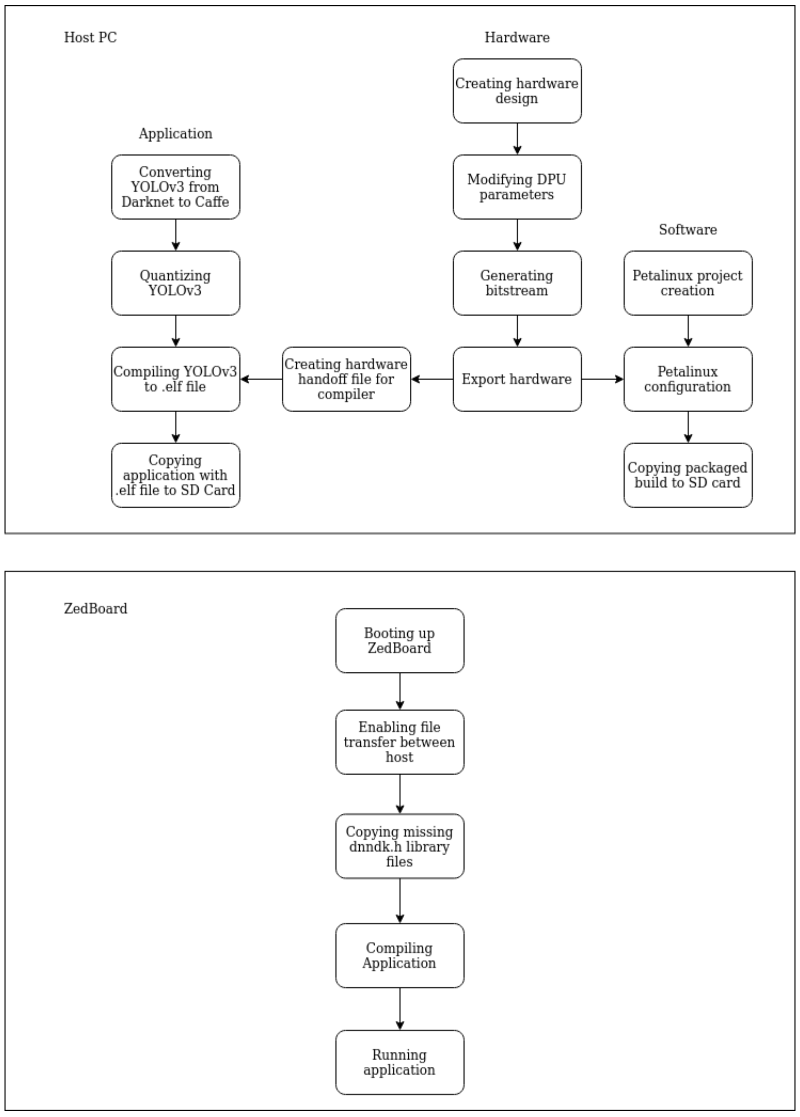

Host PC
Hardware
Creating hardware design
Modifying DPU parameters
Generating bitstream
Export hardware
Application
Converting YOLOv3 from Darknet to Caffe
Quantizing YOLOv3
Compiling YOLOv3 to .elf file
Copying application with .elf file to SD Card
Creating hardware handoff file for compiler
Software
Petalinux project creation
Petalinux configuration
Copying packaged build to SD card
ZedBoard
Booting up ZedBoard
Enabling file transfer between host
Copying missing dnndk.h library files
Compiling Application
Running application

Figure 3. *Simplified view of development flow in proposed solution*

6.2.1 Hardware

Hardware design subsection uses tutorial from Xilinx's guide 73058. here will be copies of original guide with additions if needed. Perquisites of this subsection:

- Vivado Design suite 2019.2 HLx Edition (any edition works)

The Hardware design is simplified by a Tcl script that generates the Block Diagram, the wrapper, and the constraints file. Finally, it generates the bitfile.

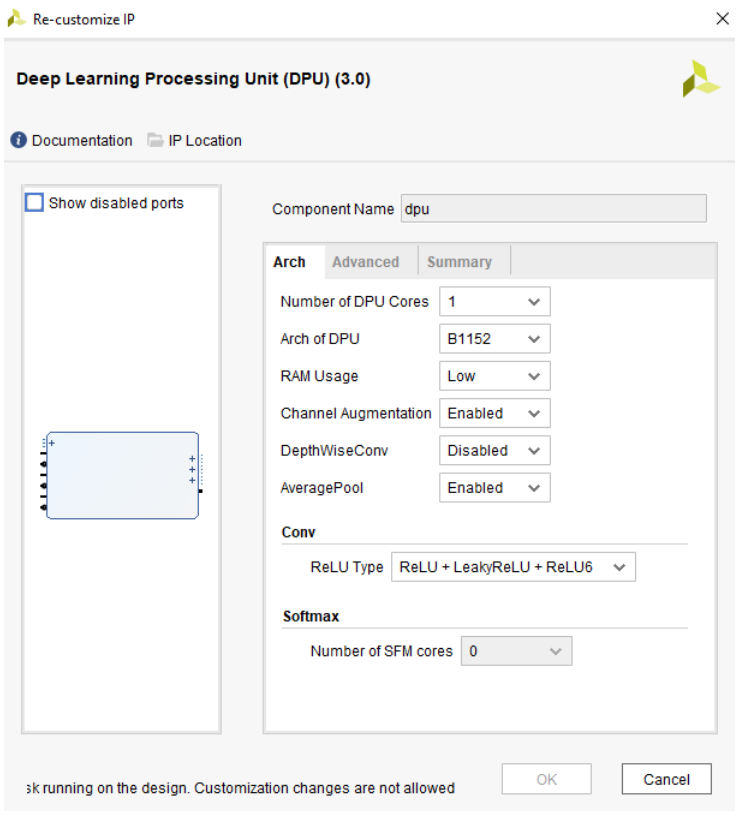
1. Download the file *Resnet50_ZedBoard_2019_2.zip*. This contains the Tcl script and the DPU IP.
2. Extract the archive

```
tree -L 2
```

3. Open Vivado 2019.2 and change the directory to HW_ZedBoard. This will be the new working directory.
4. In the Vivado Tclconsole run the following:

   ```
   cd project_path/Resnet50_ZedBoard/pl
   source ./scripts/Resnet50_ZedBoard.tcl
   ```

   The Script will open a new project, create the Block Diagram and configure the Zynq microprocessor.
5. Before continuing with original guide, Modification to DPU must be made for it to run YOLOv3. Select DPU IP by double clicking on it. On the active window (Re-customize IP) make sure the DPU configuration is as seen on following illustration.



Component Name: dpu

Number of DPU Cores: 1
Arch of DPU: B1152
RAM Usage: Low
Channel Augmentation: Enabled
DepthWiseConv: Disabled
AveragePool: Enabled

Conv
ReLU Type: ReLU + LeakyReLU + ReLU6

Softmax
Number of SFM cores: 0

sk running on the design. Customization changes are not allowed

Figure 4. *Screenshot of DPU configuration used in this project.*

6. Run “Generate Bitstream” to run the synthesis, then run the implementation and generate the configuration file. If you open the BD you should see the following as is in figure 5.
7. It is now possible to export the Hardware (with bitfile included) and procees with Petalinux flow.

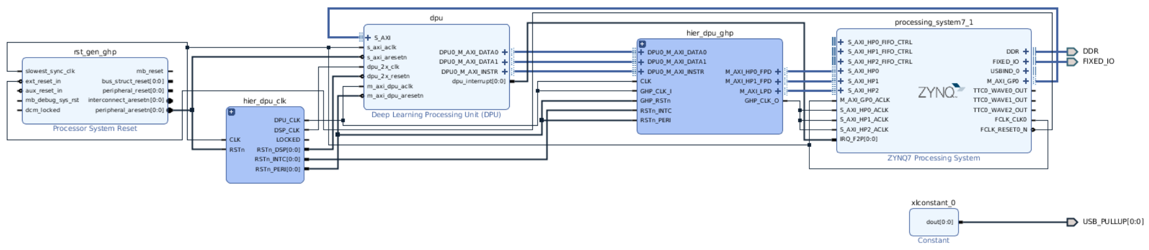


Figure 5. *Screenshot of block diagram of hardware configuration used in this project*

6.2.2 Software

This subsection covers Petalinux installation, SD card preparation, setting up Zedboard and starting Petalinux to check if the installation was successful. By the end of this subsection the user should get Petalinux booted up and running on Zedboard successfully.

Perquisites for this subsection:

- Petalinux and its perquisites are installed and configured on host computer. For installation, refer to 6.1
- User has knowledge of basic operations with Zedboard. If not, refer to Zedboard Getting Started Guide [27].

The following section is mostly copied from Xilinx's guide 73058 with additional comments and additions from author. For reference, please refer to Xilinx's guide 73058 and its references [16]. For overall explanation of software section, refer to 5.2.

Petalinux project creation from resnet50_zedboard.bsp

- Download Resnet-50 package from Xilinx homepage and extract it to desirable location on host PC. Package can be found in following link: `https://www.xilinx.com/support/answers/73058.html` For this solution a package, named "Resnet50_ZedBoard_2019_2.zip" was used.
- Setting up two useful variables:

```
cd <path_to_working_directory>
export TRD_HOME=$(pwd)
export PET_PROJ="resnet50_zedboard"
```

- Creating the Petalinux project:

```
cd $TRD_HOME/apu/resnet50_zedboard_bsp/
petalinux-create -t project -s resnet50_zedboard.bsp
    ↪ -n $PET_PROJ --force
```

- Configure the PetaLinux project with the HW design

```
cd $TRD_HOME/apu/resnet50_zedboard_bsp/$PET_PROJ/
petalinux-config --get-hw-description=$TRD_HOME/pl/
    ↪ prj/zedboard --silentconfig
```

NOTE! Before continuing with official guide from Xilinx an additional configuring must be done to reconfigure Petalinux to run YOLOv3.
Adding additional packages to Petalinux:

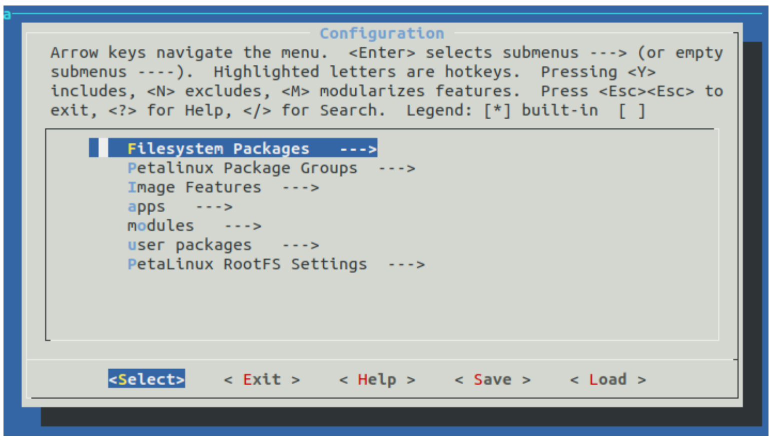
– In current terminal, type

```
petalinux-config -c rootfs
```

A window is opened where user can choose what packages can be installed on Petalinux. All necessary and desired packages must be selected before building Petalinux project. Recommended packages for running and compiling YOLOv3 application in Petalinux:

∗ filesystem packages -> devel –> make
∗ filesystem packages -> libs -> gtk+, gtk3+, opencv
∗ filesystem packages -> misc -> packagegroup-core-buildessential, packagegroup-self-hosted, xserver-common

Save changes and close Petalinux rootfs config window.



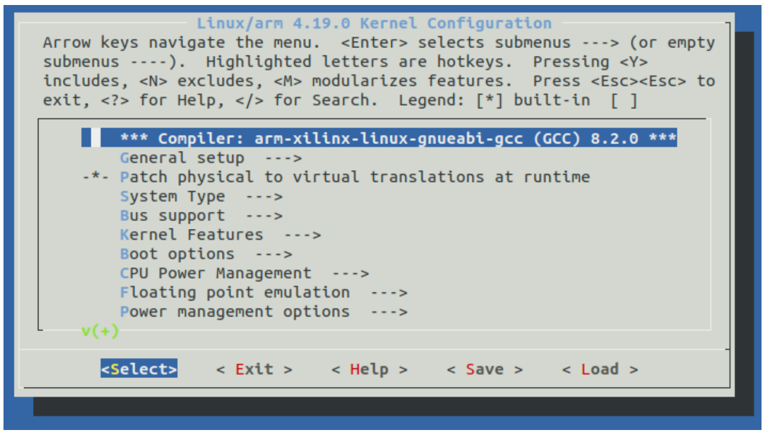
```
Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

    Filesystem Packages   --->
    Petalinux Package Groups  --->
    Image Features  --->
    apps   --->
    modules   --->
    user packages   --->
    PetaLinux RootFS Settings  --->

<Select>   < Exit >   < Help >   < Save >   < Load >
```

Figure 6. *Petalinux Root filesystem configuration window.*

– During the experimentation with various configurations of YOLOv3 on Zedboard it was found that YOLOv3 required more DMA memory than provided by default. So it is necessary to increase DMA contiguous memory allocator. For this, insert:

```
petalinux-config -c kernel
```

After that a Petalinux kernel configuration menu will appear in a separate terminal window or tab, as shown in figure 7.



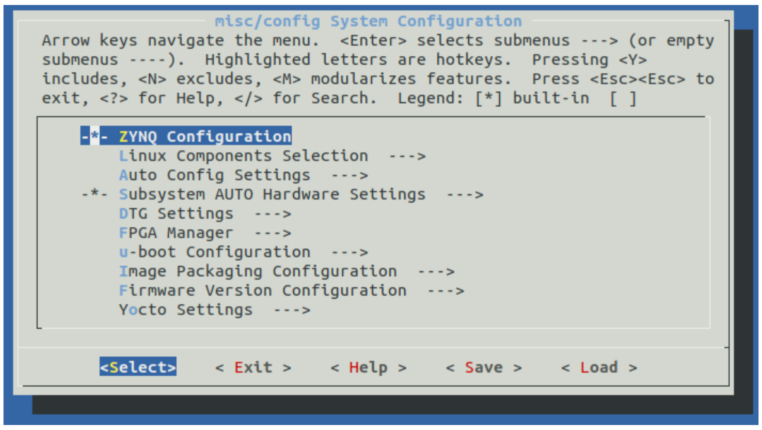
```
                    Linux/arm 4.19.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        *** Compiler: arm-xilinx-linux-gnueabi-gcc (GCC) 8.2.0 ***
            General setup  --->
        -*- Patch physical to virtual translations at runtime
            System Type  --->
            Bus support  --->
            Kernel Features  --->
            Boot options  --->
            CPU Power Management  --->
            Floating point emulation  --->
            Power management options  --->
        v(+)

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 7. *Petalinux kernel configuration main menu.*

– Navigate Petalinux kernel configuration into section device drivers -> generic driver options -> DMA contiguous Memory Allocator and make sure mentioned option is selected with "*".
  In next section: Size in Mega Bytes select it and set number to 128. In this demo 128MB is sufficient to store all necessary temporary information that YOLOv3 requires.
– Now Petalinux installation will be so large in size that making it to boot on RAMDisk becomes unfeasible. Alternative solution is to configure Petalinux *rootfs* to mount onto SD card itself. For that, insert:

```
petalinux-config
```

A Petalinux main configuration screen will appear, like shown in figure 8.

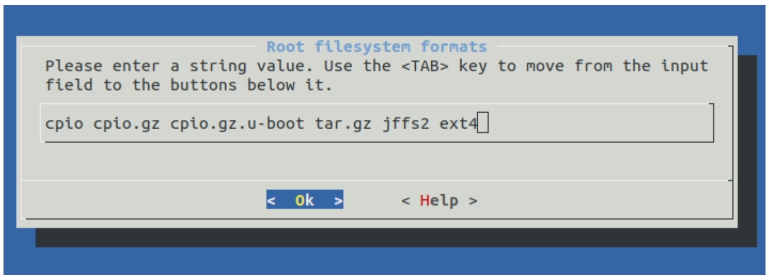


```
                    misc/config System Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        -*- ZYNQ Configuration
            Linux Components Selection  --->
            Auto Config Settings  --->
        -*- Subsystem AUTO Hardware Settings  --->
            DTG Settings  --->
            FPGA Manager  --->
            u-boot Configuration  --->
            Image Packaging Configuration  --->
            Firmware Version Configuration  --->
            Yocto Settings  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 8. *Petalinux configuration window.*

– Navigate in menu and select *EXT (SD/eMMC/QSPI/SATA/USB)* in Image packaging configuration -> root filesystem type.

- In order to create filesystem as *ext4* type we will need to add *ext4* into root filesystem formats in Image packing configuration -> root filesystem formats. Type "ext4" at the end of existing line as shown in figure 9.



```
Root filesystem formats
Please enter a string value. Use the <TAB> key to move from the input
field to the buttons below it.

cpio cpio.gz cpio.gz.u-boot tar.gz jffs2 ext4
```

Figure 9. *Petalinux Root filesystem configuration formats window.*

- Build the Petalinux project. Insert:

  ```
  petalinux-build
  ```

  This process could take several minutes.
- Create BOOT.BIN and image.ub for the SD card:

  ```
  cd $TRD_HOME/apu/resnet50_zedboard_bsp/$PET_PROJ/
      ↪ images/linux
  petalinux-package --boot --fsbl zynq_fsbl.elf --u-
      ↪ boot u-boot.elf --fpga system.bit --force
  cp BOOT.BIN image.ub $TRD_HOME/SDcard
  ```

- Preparing SD card. Guide on preparing SD card for booting Petalinux is copied from Chatura Niroshan's web article "Installing Ubuntu on Xilinx ZYNQ-7000 AP SoC Using Petalinux". For reading straight from the source, please visit the address according to citation [28]. For same purposes as why most of 73058 is copied here, the mentioned tutorial will also be written here. All credit for this refers to the original author.

  "This can be easily done with *GParted* application for Linux. In Ubuntu, open a terminal and type following command to start *GParted*.

  ```
  sudo gparted
  ```

  If gparted is not installed, you can install it using the following command.

  ```
  sudo apt-get install \emph{GParted}
  ```

  Connect the SD card to the PC using SD card reader and start *GParted* using above command. Follow the steps below to prepare the SD card.
  - Select the SD card in *GParted*.
  - Make sure its unmounted and delete the partition of the SD card so that it displays 'unallocated' in *GParted*.

- Right click the unallocated space and create a new partition with following settings. **Free Space Proceeding (MiB):** 4, **New Size (MiB):** 512, **File System:** FAT32, **Label:** BOOT. Don't change other settings and click *Add* to finish.
- Right click the remaining unallocated space and create a new partition with following settings. **Free Space Proceeding (MiB):** 0, **Free Space Following(MiB):** 0, **File System:** ext4, **Label:** rootfs. Don't change other settings and click *Add* to finish.
- Apply all changes to create the partitions." [28]

■ Copying contents from host PC to SD card.

- Copy BOOT.bin and image.ub from *$TRD_HOME/sdcard* into BOOT partition on SD Card.
- Unmount second partition of SD card (do not eject card from reader yet).
- Copy contents from file $TRD_HOME/apu/resnet50_zedboard_bsp/$PET_PROJ/images/linux/rootfs.ext4 into second partition on SD card. This can be done with command:

```
sudo dd if=$TRD_HOME/apu/
    ↪ resnet50_zedboard_bsp/$PET_PROJ/images
    ↪ /linux/rootfs.ext4 of=/dev/sdc2
```

  NOTE! Make sure the selected destination partition corresponds to SD card's partition. This command could overwrite entire system partitions if used wrong. To make sure proper partition is selected, insert:

```
lsblk
```

  to list all partitions available on host PC.
- Open *GParted*, select proped device from top right. When listing all partitions on SD Card a single partition might note an error message. Select this partition, right click on it and select *check*.
- Apply changes to check for filesystem errors. This will extend SD Cards second partition to full extent making possible to use all this space on Zedboard.
- SD Card is now ready to unmount and eject from system.

To make sure the tutorial in this subsection has been completed successfully, Petalinux installation should be checked on Avnet ZedBoard.

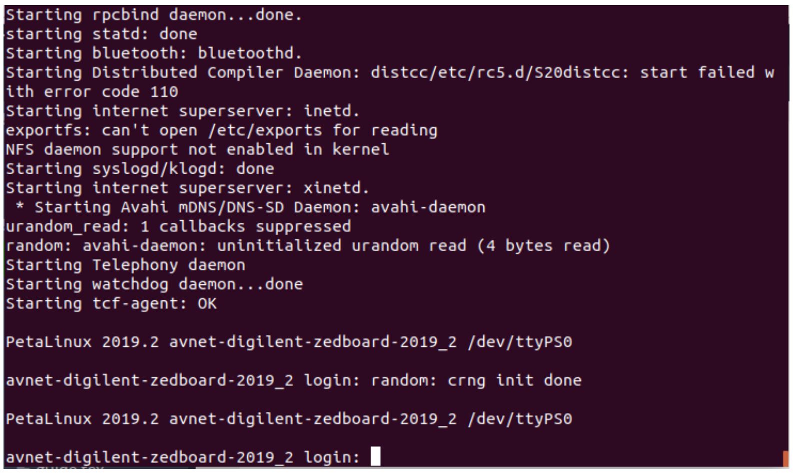
To check successful booting capability of Petalinux:

■ Make sure Zedboard is configured as stated in Zedboard getting started guide: [27, p. 11]

- If Zedboard is turned on, connect to Zedboard via USB_UART. Step by step guide can be found at [27, p. 11] for host machine running Microsoft Windows 7 or newer or [27, p. 36] for host machine running Linux. In this experiment a Linux host was used and after Zedboard is configured, a single command was used:

  ```
  sudo picocom -b 115200 /dev/ttyUSB0
  ```

  Note that device name varies by host and devices used. If Petalinux installation is successful a login screen appears in terminal, as shown in figure 10.



```
Starting rpcbind daemon...done.
starting statd: done
Starting bluetooth: bluetoothd.
Starting Distributed Compiler Daemon: distcc/etc/rc5.d/S20distcc: start failed w
ith error code 110
Starting internet superserver: inetd.
exportfs: can't open /etc/exports for reading
NFS daemon support not enabled in kernel
Starting syslogd/klogd: done
Starting internet superserver: xinetd.
 * Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
urandom_read: 1 callbacks suppressed
random: avahi-daemon: uninitialized urandom read (4 bytes read)
Starting Telephony daemon
Starting watchdog daemon...done
Starting tcf-agent: OK

PetaLinux 2019.2 avnet-digilent-zedboard-2019_2 /dev/ttyPS0

avnet-digilent-zedboard-2019_2 login: random: crng init done

PetaLinux 2019.2 avnet-digilent-zedboard-2019_2 /dev/ttyPS0

avnet-digilent-zedboard-2019_2 login: 
```

Figure 10. *Screenshot of terminal shortly after Petalinux has successfully booted greeting with login screen.*

  For logging in, the default username is "root" and password is "root".

- It is recommended to use sftp for more intuitive file transfer between host and device. In Linux this can be done by using file manager, like Nautilus, and inserting:

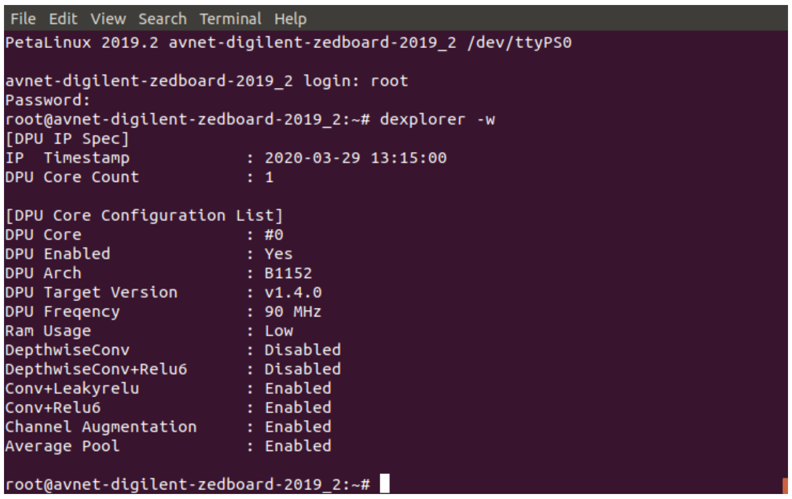
  ```
  sftp://root@IP_ADDRESS
  ```

  Note that IP_ADDRESS shall correspond to IP address set to Zedboard and network connection between host and device has been established.

When connection to the Zedboard is successful it is also important to check if DPU has been detected by the Petalinux kernel. For detection insert:

```
dexplorer -w
```

To view DPU signature information for selected board [17, p. 25]. If DPU is detected correctly then terminal output should look something like in figure 11:



```
File  Edit  View  Search  Terminal  Help
PetaLinux 2019.2 avnet-digilent-zedboard-2019_2 /dev/ttyPS0

avnet-digilent-zedboard-2019_2 login: root
Password:
root@avnet-digilent-zedboard-2019_2:~# dexplorer -w
[DPU IP Spec]
IP  Timestamp             : 2020-03-29 13:15:00
DPU Core Count            : 1

[DPU Core Configuration List]
DPU Core                  : #0
DPU Enabled               : Yes
DPU Arch                  : B1152
DPU Target Version        : v1.4.0
DPU Freqency              : 90 MHz
Ram Usage                 : Low
DepthwiseConv             : Disabled
DepthwiseConv+Relu6       : Disabled
Conv+Leakyrelu            : Enabled
Conv+Relu6                : Enabled
Channel Augmentation      : Enabled
Average Pool              : Enabled

root@avnet-digilent-zedboard-2019_2:~#
```

Figure 11. *DPU Signature Viewed on Zedboard with DExplorer*

It is necessary to include additional DNNDK library in order to compile YOLOv3 applications on Zedboard after the Zedboard has started. Here are steps to install additional dnndk.h libraries into Petalinux on Zedboard:

- Copy *xilinx_dnndk_v3.1/ZedBoard* folder and its contents into root filesystem on Zedboard.
- Copy contents of */path_to_folder/ZedBoard/pkgs/lib* into */usr/lib* folder. Some files already exist at the destination folder, feel free to skip duplicate files.
- Copy contents of */path_to_folder/ZedBoard/pkgs/include* into *usr/include/dnndk* folder. Folder */usr/include/dnndk* must be created in this path

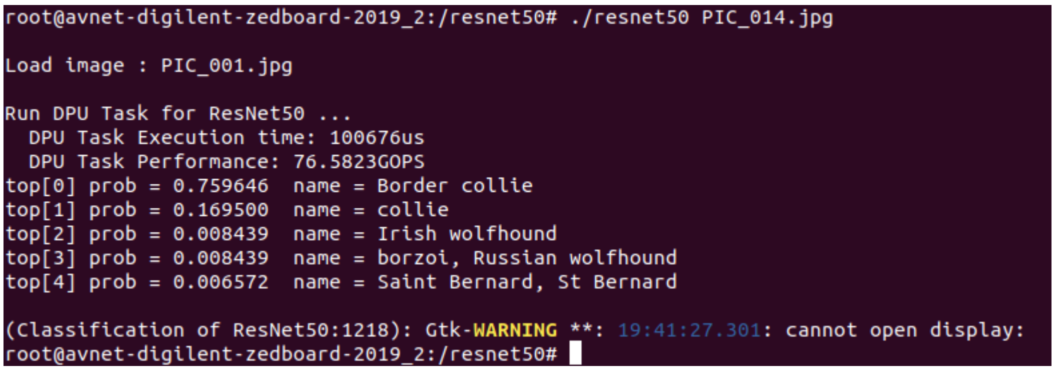
Resnet-50 is a suitable sample to check if application compiling can be done on Avnet ZedBoard. Navigate into */path_to_folder/ZedBoard/samples/resnet/* folder and insert:

```
make
```

If *make* is successful without any errors then Zedboard is configured and ready to compile and run YOLOv3. Resnet-50 can be run by inserting:

```
./resnet50 path\_to\_image/image.jpg
```

It should try to detect and classify objects as seen in figure 12.



```
root@avnet-digilent-zedboard-2019_2:/resnet50# ./resnet50 PIC_014.jpg

Load image : PIC_001.jpg

Run DPU Task for ResNet50 ...
  DPU Task Execution time: 100676us
  DPU Task Performance: 76.5823GOPS
top[0] prob = 0.759646  name = Border collie
top[1] prob = 0.169500  name = collie
top[2] prob = 0.008439  name = Irish wolfhound
top[3] prob = 0.008439  name = borzoi, Russian wolfhound
top[4] prob = 0.006572  name = Saint Bernard, St Bernard

(Classification of ResNet50:1218): Gtk-WARNING **: 19:41:27.301: cannot open display:
root@avnet-digilent-zedboard-2019_2:/resnet50#
```

Figure 12. *Resnet-50 output after running object detection on provided sample image.*

6.2.3 Application

This subsection covers converting YOLOv3 from Darknet to Caffe framework and application compilation on Zedboard and DPU configuration file compilation on host PC. By the end of this subsection it is possible to run YOLOv3 application on Zedboard, accelerated with Xilinx DPU. This subsection is mostly work of author with some sections from other guides from Xilinx.

Perquisites for this subsection:

- Previous subsections are completed (hardware and software) and Petalinux is running able to run on Zedboard.
- Xilinx DNNDK is installed on host PC and Zedboard
- Prequisites for Edge AI tutorial "YOLOv3 Tutorial: Darknet to Caffe to Xilinx DNNDK" are completed[18].

This guide consists of sections divided into subsections. The guide can be completed by following steps below:

- Creating hardware description file for DNNQ.
  This can be done by using *dlet* utility.
  - Locate *top.hwh* file. In this project *top.hwh* can be found at *path_to/Resnet50_ZedBoard_2019_2/pl/prj/zedboard/ zedboard.srcs/sources_1/bd/top/hw_handoff/* .
  - Copy the *top.hwh* to new folder made to desirable location, for example *home-/username/zedboard_hw* .
  - Navigate to this folder and open command prompt, insert:

```
dlet -f top.hwh
```

This creates new file ending with .dcf extension necessary for deep neural compiler. If *dlet* fails and reports stack smashing, relocate file to another location and try again. If message Generate DPU DCF File filename-dcf successfully is seen on terminal, then .hwh to .dcf conversion was successful. For more information, refer to Xilinx DNNDK user guide [17].
NOTE! During each use *dlet* might crash with "Aborted (core dumped)" printed on command line. This issue does not provide problems for file conversion and can be ignored.

- Converting YOLOv3 from Darknet to Caffe.
  This part is heavily inspired by Edge AI tutorial "YOLOv3 Tutorial: Darknet to Caffe to Xilinx DNNDK" Refer to this guide before continuing [18]. If all prequisites are met and all of the necessary software is installed, a single script will convert YOLOv3 from Darknet to Caffe.

  ```
  0_convert.sh
  ```

  This script will convert darknet to caffeemodel. Prior executing the script make sure *yolov3.cfg* and *yolov3.weights* files are in
  path_to/Edge-AI-Platform-Tutorials-3.1/docs/
  /Darknet-Caffe-Conversion/example_yolov3/0_model_darknet folder.
  If needed to check whether converter YOLOv3 works as intended
  1_test_caffee.sh can be used, for more information, refer to guide provided by Xilinx [18].
- Quantizing YOLOv3 in caffe network using DECENT.
  This step is similar with previous step. By running:

  ```
  2_quantize.sh
  ```

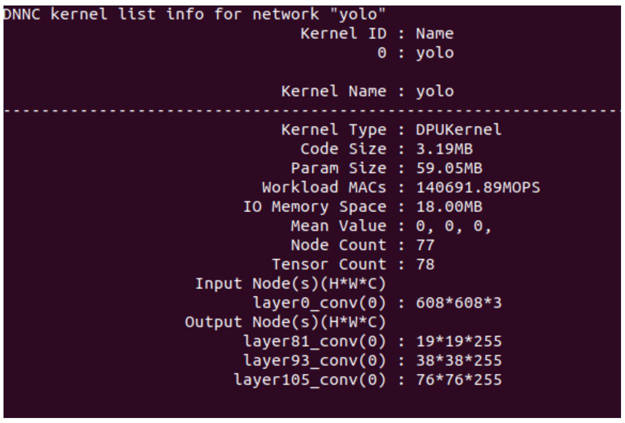
  Xilinx's DECENT tool will quantize YOLOv3 Caffeemodel and apply pruning, increasing its performance on DPU with small accurary degradation.
  For more information on DECENT, refer to DNNDK guide [17, p. 40].
  For more information on the script, refer to tutorial provided by Xilinx [18].
- Compiling YOLOv3 info .elf using DNNC.
  Edge AI tutorial [18] provides script to use Deep neural network compiler. For more rapid development it is better to use shell script provided by Xilinx DNNDK package.
  This script is located at *xilinx_dnndk_v3.1/host_x86/models*
  /caffe/resnet50/dnnc_Zedboard.sh.
  There are many scripts in *xilinx_dnndk_v3.1/host_x86/models/caffe/resnet50/dnnc* folder corresponding to different board models DNNDK officially supports. In this project the target board is Avnet ZedBoard so *dnnc_Zedboard.sh* is used. Before

launching:

- Copy contents (deploy.caffemodel and deploy.protxt) from prior working directory in Edge AI tutorial folder to *xilinx_dnndk_v3.1/host_x86/models/caffe/resnet50/decent_output* folder. If last folder does not exist, create one named *decent_output*.
- Modify *dnnc_zedboard.sh* as follows.
  Replace line 3 from *Resnet50* to *yolo*.
  Replace line 7 *dnndk_dcf="../../../dcf/ZedBoard.dcf"* with path to previously created .dcf file on host PC.
- Execute script:

  ```
  dnnc_ZedBoard.sh
  ```

  This should generate .elf file into *dnnc_output* folder. If successful the terminal output will look similar as in figure 13:



```
DNNC kernel list info for network "yolo"
                               Kernel ID : Name
                                       0 : yolo

                             Kernel Name : yolo
---------------------------------------------------------------
                             Kernel Type : DPUKernel
                               Code Size : 3.19MB
                              Param Size : 59.05MB
                           Workload MACs : 140691.89MOPS
                         IO Memory Space : 18.00MB
                              Mean Value : 0, 0, 0,
                              Node Count : 77
                            Tensor Count : 78
                    Input Node(s)(H*W*C)
                          layer0_conv(0) : 608*608*3
                   Output Node(s)(H*W*C)
                         layer81_conv(0) : 19*19*255
                         layer93_conv(0) : 38*38*255
                        layer105_conv(0) : 76*76*255
```

Figure 13. *Screenshot of terminal shortly after DNNC has completed successfully*

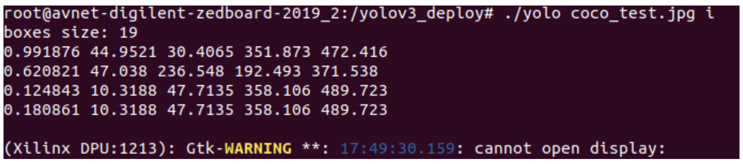
- Copying source files from host PC to target device. Extract *yolov3_deploy.tar.gz* into desired folder on Zedboard. Recommended to use sftp and graphical file browser for easy file transfer. Navigate into */path_to/yolov3_deploy/model* folder and delete all content inside of it.
- Compiling and running YOLOv3 application on Zedboard.
  To compile application, use terminal to navigate into *yolov3_deploy* folder on Zedboard and insert:

  ```
  make
  ```

  If successful, no error or warning messages should appear. To test application, launch it by typing:

  ```
  ./yolo coco_test.jpg i
  ```

  If successful, the result should look something like in figure 14:



```
root@avnet-digilent-zedboard-2019_2:/yolov3_deploy# ./yolo coco_test.jpg i
boxes size: 19
0.991876 44.9521 30.4065 351.873 472.416
0.620821 47.038 236.548 192.493 371.538
0.124843 10.3188 47.7135 358.106 489.723
0.180861 10.3188 47.7135 358.106 489.723

(Xilinx DPU:1213): Gtk-WARNING **: 17:49:30.159: cannot open display:
```

Figure 14. *Screenshot of terminal shortly after YOLOv3 application had run successfully*

If there is need to see visual output of YOLOv3 from the application, make sure in main.cc line nr. 443, is written as *imwrite("result.jpg", img);* and is not commented out. A result image appears in the same folder as application is executed in.
Here is example image output of YOLOv3 running on ZedBoard with modified application, showing class names in output images. Detected classes were: person 99.1% confidence; dog 62% confidence; chair 12.4% confidence; sofa 18.1% confidence.

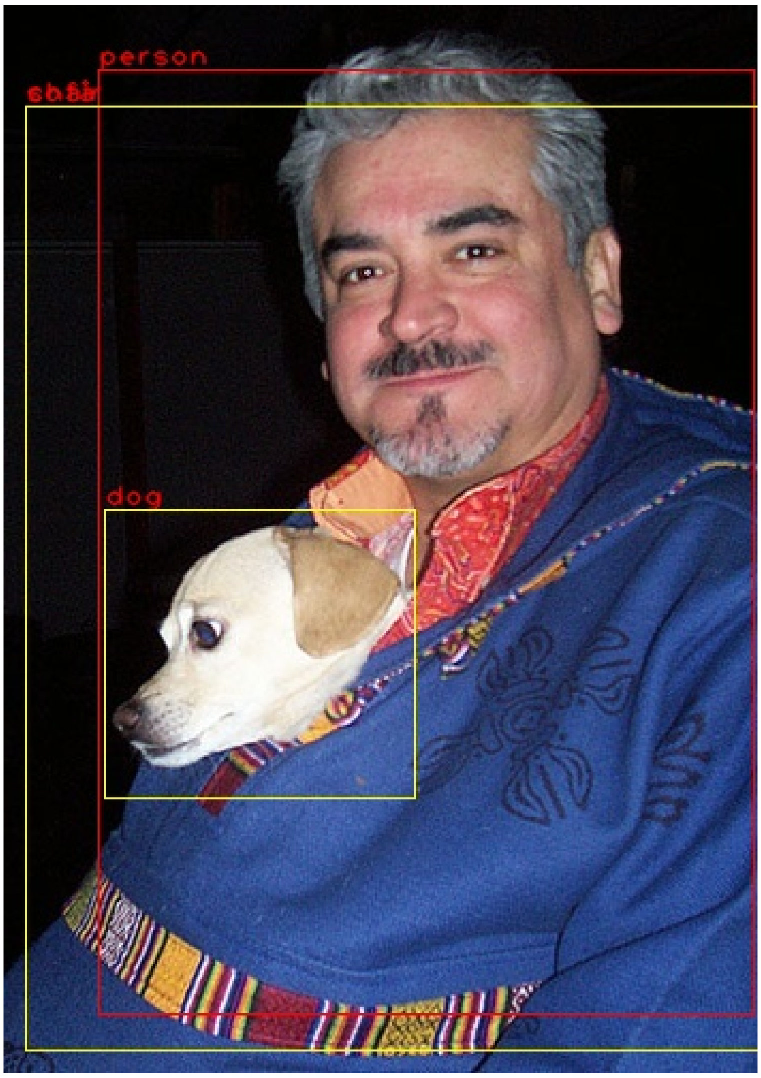


Figure 15. *Test image included in Edge ai toolset with detection boxes from YOLOv3 on ZedBoard*

7. Analysis

The analysis of the thesis consists of 3 main aspects that are analyzed in this solution: image measuring and performance and latency. Image measuring part determines if neural network on Zedboard is able to detect objects. The performance part analyzes how much application utilizes DPU and its efficiency. The third part displays what is average measured latency of YOLOv3 on Zedboard.

7.1 Proof of Concept and Accuracy

This section proves that the proposed solution performs as intended and the YOLOv3 is able to detect and classify objects on pictures. The calculations for mean average precision (mAp) were done by using ready made open source solution. This solution is made for evaluation of the object detection problem and is excellent for evaluating object detections in a simplified way [29]. For this evaluation a *sample2* practical example system was used as a basis.

In order to use this software for evaluation, it is important to set up an evaluation data. Truth information is located in separate files for each picture in folder */groundtruths*. "In these files each line should be in the format: <*class-name*> <*left*> <*top*> <*right*> <*bottom*>" [29]. The COCO 2017 validation dataset contains 5000 pictures, named "2017 Val images [5K/1GB]" [2]. Due to the time restrictions of the thesis and the size of the given dataset, a set of 20 pictures were selected randomly from COCO validation dataset. Those pictures were then captioned by hand and groundtruth files of these pictures were copied to */groundtruth* folders. Additional information of the selected files and their ground truth information can be found at GitHub page by author [30].

Detection data for evaluation was created accordingly: a shell script was made to run YOLOv3 application on ZedBoard that iterated all pictures in the specified folder and inserted detections one at the time into separate files of the */results* folder. The contents were then copied into */detections* folder for evaluation.

The sample provided by validation software is designed to calculate mean average precision for each object class detected in ground truth files. If object class is detected in detection files, but has not been mentioned in ground truth files, the mAp will be n/a meaning it could not calculated. If specified object class is present in ground truth files, but neural network is unable to detect it, the mAp will be 0. Based on 20 picture sample set the results were as seen in table 2.

Table 2. *Average accuracy values per class:*

Class name	**mAp**
apple:	0.666667
backpack:	1.000000
baseball bat:	1.000000
baseball glove:	1.000000
bench:	0.972222
bird:	1.000000
book:	1.000000
bottle:	1.000000
bowl:	1.000000
car:	0.750000
clock:	1.000000
horse:	0.000000
hot dog:	0.666667
person:	0.790208
skateboard:	1.000000
snowboard:	1.000000
sofa:	1.000000
surfboard:	1.000000
tie:	1.000000
train:	1.000000
umbrella:	1.000000
zebra:	1.000000

As seen most of classes are having 100% accuracy based on the results of these pictures. This leads to an assumption that YOLOv3 is perfectly capable of detecting mentioned objects from pictures. This is not true, because most of these classes had an occurrence of 1-3 in this custom made dataset, making it easy for the validation application to output such high accuracy results. This analysis still prove that YOLOv3 is capable to detect and classify objects on pictures while running on Avnet ZedBoard.

7.2 Performance and Latency

One of the tasks was to measure performance and latency of the given system and analyse its results. These results give overview on how efficiently YOLOv3 performs on Avnet ZedBoard with the given configuration and what is its performance.

7.2.1 Performance and efficiency

Performance and efficiency are important variables used for measuring artificial neural networks. These metrics show capabilities of object detection algorithms in action and can be used to decide use cases in the future.

In this context the performance is regarded as a computational capacity of a processing unit to perform calculations needed for a single forward pass of a neural network in a fixed set of time. This is measured in billions floating point operations per second, also named GFLOP/s or GFLOP/sec.

Workload of an artificial neural network is calculated by counting all floating point operations needed for a single pass. Single pass means a single image is passed through artificial neural network at the time. Workload is measured in billion of operations, also named GOP.

Efficiency in this analysis is regarded as measured computing performance in FLOPS per energy consumption (in Watts). This value, also called "Energy efficiency", is measured in “Operations/Second ”per Watt (OPS/W).

7.2.2 Workload calculation

There were difficulties in calculating the workload of YOLOv3 in this thesis, because it demands some expertise in this field. Workload depends on network model, framework and application it is running on. Official workload for YoloV3 provided by whitepaper is 18.7GOP [3]. This workload refers to YOLOv3 on a single forward pass in Darknet model. Looking at another source presented by Xilinx in 2018, claims YOLOv3 workload to estimate around 65.25GOP [31, p. 36] when running on DPU, specifications of its architecture were not noted. Another claim can be taken directly from DNNC which compiles neural network with its weights to specific architecture and system and estimates its total workload. From DNNC workload was claimed to be 140.691GOP as seen in figure 13. Due to large differences between each claims one was needed to be taken as a base to calculate upon.

In this analysis a DNNC version of workload was chosen because of the following reasons:

1. Workload can depend on platform on which an application is running on, especially when neural network has been quantized or modified from original. DNNC calculates workload for DPU with specific architectural parameters.
2. Xilinx provides sample networks for Avnet ZedBoard, notable Resnet-50 model. In this model a claim of 7.71GOP workload is given to Resnet-50 for a single forward pass. This claim can be viewed in line 74 of *main.cc* source file in Resnet-50 sample. This claim can be checked by running DNNC for same configuration and network and see if results deviate. As a result DNNC calculated the same workload as was written to source files of Resnet50 sample. Therefore it can be assumed this compiler can calculate workload of YOLOv3 as well.
3. Workload was manually calculated for checking purposes. A quantized Caffe model, which was directly converted from original YOLOv3 model and quantized using DNNQ. This provides most accurate model for calculating total workload of YOLOv3 as it will be compiled into DPU. More information on workload calculation is written below.

Here is an explanation how the workload was calculated to check the results from the DNNC. To understand convolutions, a well-explained web article by Sumit Saha explains basics on convolution calculations in an artificial neural networks. [32]. An approximate result on how to calculate workload can be found by following calculations done below. A convolution calculation of a single pixel can be simplified as a matrix multiplication of kernel by given size with corresponding values from weights and depth of input layer and result matrix summed up into a single value. This workload can be calculated by the given formula:

Single convolutional workload

$$n_{conv} = 2(k_x)^2 z_{input} \tag{7.1}$$

Where k_x is kernel size and z_{input} is depth of an input layer. This process is repeated for every pixel in a single filter and then in every filter. There is also a guide on convolution arithmetic that explains how output size can easily be calculated [33]. Formula for this is relationship 6 from given guide and it is described as follows:

Output size calculation

$$x_{out} = [\frac{i + 2p - k}{s}] + 1 \tag{7.2}$$

Where *i* is size of an input layer, *p* is convolution padding size, *k* is convolution kernel size and *s* is convolution stride size. As seen in YOLOv3 model provided by quantized model *v3.protxt* where width and height of layers are the same, making this architecture symmetric. Therefore both layer and width can be calculated at once. To calculate total convolution workload of a single layer this equation can be used:

Convolution calculation workload of single layer

$$a_{conv} = n_{filters} n_{conv} (x_{out})^2 \tag{7.3}$$

Where a_{conv} is total convolution workload of single layer and $n_{filters}$ are number of filters applied to given layer.

To calculate total workload of an layer, an activation layer also needs to be considered. In YOLOv3 model an activation layer is Relu, or in other case LeakyReLu, more information on ReLu's are chapter 4.2. To summarise a calculation workload for an each pixel is a single multiplication is done which uses floating point values. The calculation of a total activation function workload of given layer can be calculated by given formula:

Activation calculation workload

$$a_{act} = n_{filters} (x_{out})^2 \tag{7.4}$$

To sum up total calculation workload of a single convolutional layer in artificial neural network this formula can be used:

Total estimated workload of single convolutional layer

$$a_{total} = a_{conv} + a_{act} \tag{7.5}$$

For example, *layer1_conv* total calculation can be calculated when input size is 608 x 608 and depth 32. Using 3 x 3 kernel with 64 filters with padding size 1 and stride 2 yields to output of 304 x 304 layer with depth of 64. Single convolution workload was 576 operations (OP) and total workload of this layer was calculated 3,412,738,048 OP or 3.41 GOP.

By calculating total workload of all convolution layers we can get approximate workload of any convolutional neural network, because vast majority of computationally expensive workload is in convolutional and activation calculations. Total number of convolutional layers in YOLOv3 Caffe was counted 75.

There are also many instances in YOLOv3 architecture where 2 layers with the same size parameters are merged into 1. These layers were detected 27 in Caffe. The calculation workload for this kind of operation was multiplying dimensions of an input layer, counting for every addition needed for a single output layer.

When all convolutional layer parameters were added and calculated as described before, a total estimated workload of YOLOv3 Caffe is 140,056,261,363 GOP which is similar to result provided by DNNC (140.691GOP). Those differences might come from workload calculation of upsampling some layers which were not counted in current calculation, because of a low workload compared to the rest of the algorithm.

All of the calculations for workload can be seen in author's GitHub page [30].

7.2.3 Performance

Performance can be calculated by dividing workload with execution time. This can be done both manually (measuring execution time of forward pass and dividing workload with it) or by automatically using Xilinx's DPU Profiler feature, called Dsight. Dsight is a DNNDK performance profiling tool, mainly for neural network model profiling. To use Dsight a DPU profiling mode must be turned on before running task on DPU. This can be done by inserting *dpuEnableProfiling()* before dpuRunTask in YOLOv3 application. More information about this utility can be found at Vitis AI User guide [34, p. 84]. By running Dsight on YOLOv3 on Avnet Zedboard following results were acquired:

Table 3. *DPU Profiler results running YoloV3 at 90MHz*

Class name	**mAp**
Total nodes (workload) in MOP	140691.89
Memory (MB)	267.38
Execution time (ms)	1504.94
Efficency(%)	90.2
Perf(GOPS)	93.5

To validate results, a test was taken place on a same system, but with reduced clock speed: from 90MHz to 50MHz and results can be seen in table 4:

Table 4. *DPU Profiler results running YoloV3 at 50MHz*

Class name	**mAp**
Total nodes (workload) in MOP	140691.89
Memory (MB)	267.38
Execution time (ms)	2708.29
Efficency(%)	90.2
Perf(GOPS)	51.9

Measurements with different clock speed were required to determine whether Dsight is able to detect maximum theoretical performance of DPU with given hardware parameters or not. The tests above prove that these are correct. In fact it can be further displayed by table that visualizes maximum theoretical performance of DPU at different clock speeds. As a reference point a theoretical maximum of 230GOP/sec was used from DPU user guide [35, p.29] which states this value from Z7020 device (same as on Avnet ZedBoard) using same DPU configuration (B1152 x 1) at higher clock speed 200MHz.

Table 5. *A table showing measured performance and efficiency with different clock speeds*

Clock speed [MHz]	**Theoretical maximum performance [GOP/s]**	**Measured performance [GOP/s]**	**efficency**
200	230	cannot measure	cannot measure
90	103.5	93.5	90.3
50	57.5	51.9	90.26

Performance was measured and result is 93.5GOP/s, power consumption can be read from power delivery report from Vivado when generating bitstream from project. Another design was also generated which had only Processing system included. To calculate only DPU power consumption, it can be done by dividing total CPU power consumption from DPU. Total CPU power consumption was rated 1.669W. Total combined power consumption was rated 3.23W. Efficiency can be calculated using this formula:

$$\eta[GOP/S/W] = \frac{N}{P_{total} - P_{cpu}} \tag{7.6}$$

$$\eta[GOP/S/J] = \frac{N}{(P_{total} - P_{cpu}) * t} \tag{7.7}$$

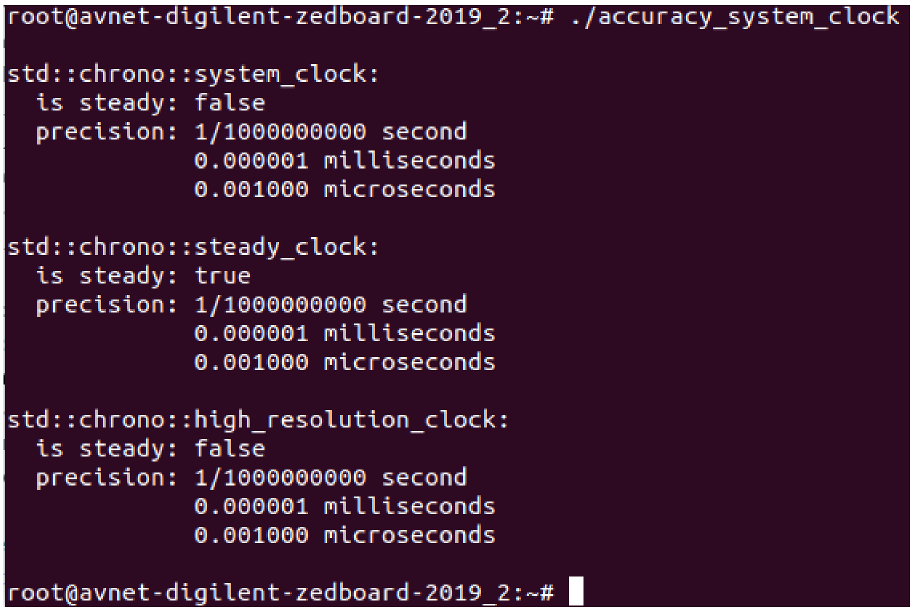
Where *N* is workload, *P* is measured power consumption and *t* is an elapsed time of inference during single pass. The results are 59.513GOP/s/J or 89.686 GOP/s/W.

At 200MHz performance can not be read on Zedboard, because ZedBoard does not provide enough electrical power for Z7020 to power DPU at 200MHz. This was noted in Xilinx guide 73058 [16]. Efficiency in table 5 are calculated from theoretical maximum performance and performance measured by Dsight. As seen, efficiency values in table 5 are nearly identical to ones Dsight provided. Provided by results it is safe to say that DNNDK tools have a knowledge of theoretical maximum performance of DPU in a given configuration

7.2.4 Latency

Here is a brief explanation of how latency was measured in the proposed solution and what were the results. Latency is regarded as a total latency of a single image pass on YOLOv3. The time measurement starts from beginning of the application to the end until results are displayed in terminal. For this measurement, a *std::chrono* library was used which is widely utilized in C++ to measure different time intervals at medium to high accuracy.

To measure accuracy of results, a console application was downloaded from a web article "The Three clocks" [36]. This article briefly explains how *std::chrono* measures time and how it is possible to measure accuracy of time measurement *std::chrono* provides. For more explanation, refer to the article [36]. A sample program was downloaded as a source file and copied to Zedboard. After building and running the application, the results showed accuracy values of the measuring time in the current system. The accuracy value was measured 0.001 μs (microseconds). Overall output of the given application is shown below in figure 16.



```
root@avnet-digilent-zedboard-2019_2:~# ./accuracy_system_clock

std::chrono::system_clock:
  is steady: false
  precision: 1/1000000000 second
             0.000001 milliseconds
             0.001000 microseconds

std::chrono::steady_clock:
  is steady: true
  precision: 1/1000000000 second
             0.000001 milliseconds
             0.001000 microseconds

std::chrono::high_resolution_clock:
  is steady: false
  precision: 1/1000000000 second
             0.000001 milliseconds
             0.001000 microseconds

root@avnet-digilent-zedboard-2019_2:~# 
```

Figure 16. *Accuracy of std::chrono time measurement functions on Avnet Zedboard running Petalinux*

A series of tests were made to measure a total latency and a latency of different components in the software. A single picture was used to measure the duration of each part in YOLOv3 application. Each section had 10 measurements for more consistent results. A median was taken because of a tendency of some rare deviations from other results. Total latency is calculated as a sum of all components' latency in YOLOv3 application.

Below is an explanation of each part and their median latency time:

1. Start to DPULoadKernel (302.725ms): program initialization, image acquisition from file system to application memory, DPU opening.
2. DPULoadKernel (2038.235ms): in this phase DPU is loaded with YOLOv3 network model and its weights.
3. RunYolo preprocessing (306.4475ms): setting tensor height and width for image and image transmission to DPU.
4. RunDPUTask (1507.305ms): Forward pass of image in YOLOv3 in DPU.
5. Postprocess (308.75): post processing of data from output layers of YOLOv3: calculating detection boxes and constructing result image.
6. program to end (62.901ms): printing results to terminal, closing DPU and freeing used memory.

Below is a figure 17 visualizing total latency of YOLOv3 single image inference.

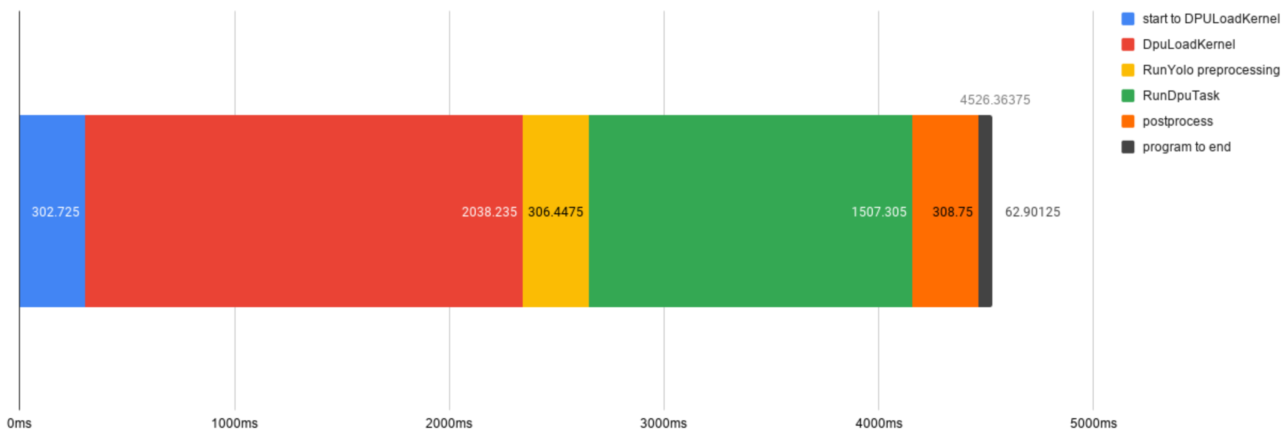


Figure 17. *Total Latency of single image YOLOv3 object detection on Avnet ZedBoard*

As seen in illustration, total average latency in this solution is 4526.36375 ms which is over 4.5 seconds. This is considerably long than noted in YOLOv3 documentation, which states 78 ips (images per second) while performing at 1457 GOP/sec [3]. This visualises that the original workload is 18.7GOP compared to 140GOP used in proposed solution. In conclusion, the proposed solution takes more time to calculate mainly because of the lower performance of DPU on ZedBoard and the high workload of YOLOv3 on ZedBoard.

8. Conclusions

As seen from analysis results, it can be claimed that YOLO object detection algorithm can be implemented on Avnet ZedBoard while its inference is being accelerated by DPU. This solution is still a proof of concept and there are many more parameters that can be analysed and improved over time.

The 1.507 second inference time and total of 4.526 seconds in latency of a forward pass in a single image indicates the proposed solution's inability to detect objects in real-time applications. Most real-time computer systems require input data from at least several pictures within a second to successfully make time-critical decisions. This long latency comes from a high workload of YOLOv3 running on DPU provided by Xilinx. This problem can be solved by running more lightweight applications than YOLOv3, for example TinyYOLOv3. Running YOLOv3 on a more computationally capable platform is also a solution.

The formulas in chapter 7.2.2 provide useful information on calculating workload of convolution neural networks and can be used in various situations. The calculation results also indicate DNNC ability to accurately measure workload of artificial neural networks, which results were then used to calculate performance and efficiency.

The utilization of DPU during inference (90.3%) stayed constant during testing of proposed system with different clock speeds. This indicates that until 90MHz the only constraints of inference's computational performance is limited by computational capability and the power delivery capacity of Avnet ZedBoard.

9. Future work

This thesis solution proves the feasibility to run YOLOv3 artificial neural network on Avnet ZedBoard without developing custom accelerator and using Xilinx's DPU for this application. However, the proposed solution is only proof of concept and requires more work and troubleshooting in order to get more practical results out of it.

Creating a custom dataset of 20 images proves that YOLOv3 on Avnet ZedBoard is able to detect objects. This does not provide sufficient measuring results of its accuracy. For this reason, one of the main challenges should be to evaluate the detection accuracy of YOLOv3 on Avnet ZedBoard. A benchmark with larger set of images must be executed for more accurate results.

As stated in behavioural notes in chapter 5.4, the particular design had some technical issues that posed notable usage difficulties. This system will become more stable if the aforementioned problems get solved. Afterwards, the precision calculations could also be improved.

Performance calculations were done using utilities provided by Xilinx. These results can be further evaluated by measuring YOLOv3 performance accelerated by DPU on other platforms and then comparing results.

To speed up the configuration time of a proposed solution, a sample project can be made which could contain hardware descriptions, configuration for Petalinux and modifications for user applications. This makes possible to create a single script which could build and compile the whole solution ready to upload to SD card and use on Avnet ZedBoard.

If above mentioned suggestions are completed, it will be simpler to implement a practical solution. Proposed system can be used to rapidly speed up inference in state of the art convolutional neural networks in various universities, speeding up research and development in various fields.

10. Summary

The initial task of the thesis was to briefly analyse Vitis Unified Software Plaftorm by Xilinx and use it to accelerate inference of an object detection algorithm on Avnet ZedBoard. An algorithm, named YOLO, was selected because it is well researched algorithm and is easy to use and modify.

After the initial experiments author found out that Vitis is not designed for artificial neural network inference. Vitis AI was experimented next. After some research, author also found that although Vitis AI is said to support YOLOv3, an actual support was not yet implemented in the official tools. An older version of Vitis AI software, called Deep Neural Network Development Kit (DNNDK), was used for inference acceleration on Avnet ZedBoard.

A solution was found after many attempts that proved satisfactory. The proposed solution uses Xilinx DNNDK with modified accelerator and Petalinux operating system to run YOLOv3 object detection algorithm on Avnet ZedBoard. Chapter 4 provides a brief explanation of all the tools used to implement proposed solution.

Chapter 6 contains is a guide on replicating a proposed solution with tools listed in chapter 4.

The analysis part divided into performance, efficiency and latency. Performance section describes the calculations that were made to determine workload of YOLOv3 on Avnet ZedBoard. This proved Xilinx's Deep Neural Network Compiler (DNNC) ability to measure workload of artificial neural networks. An utility, called *Dsight*, was used to benchmark performance of YOLOv3 on ZedBoard and as a result 93.5GOP/s at 90% utilization was measured. Efficiency was calculated 89.686 GOP/s/W respectively. Latency was measured by using *std::chrono* in user application. Total latency of a single image pass was measured 4.526 seconds, including DPU initialisation, inference, preprocessing and post-processing.

Author pointed out several tasks that could improve the proposed solution, noted in chapter 9. These include solving technical issues and accuracy benchmarking with large set of images.

The proposed solution can be useful for educational purposes. It can be used for inferencing different artificial neural networks without the need for a developer to have an expertise in hardware design. The proposed solution can speed up the development flow for prototyping new potential systems and solutions.

Bibliography

[1] Ahmed Ghazi Blaiech et al. “A Survey and Taxonomy of FPGA-based Deep Learning Accelerators”. In: *Journal of Systems Architecture* 98 (Sept. 2019), pp. 331–345. DOI: 10.1016/j.sysarc.2019.01.007. URL: https://doi.org/10.1016/j.sysarc.2019.01.007.

[2] COCO Consortium. *COCO - Common objects in Context*. [Online; accessed 12-April-2020]. URL: http://cocodataset.org/#home.

[3] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018). URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[4] Tallinn University of technology. *Course details IAY0550 SYSTEMS-ON-CHIP DESIGN*. [Online; accessed 27-04-2020]. URL: https://ati.ttu.ee/index.php?page=10193&aine=IAY0550#.

[5] Xia Jun Chen Chenchai Zhilei. *Design and Implementation of YOLOv2 Accelerator Based on Zynq7000 FPGA Heterogeneous Platform*. [Online; accessed 24-04-2020]. URL: https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDTEMP&filename=KXTS201910005&uid=WEEvREcwSlJHSldRa1FhdXNXaEhoOGhUTzA5T0tESzdFZ2pyR1NJR1ZBaz0=%5C$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4IQMovwHtwkF4VYPoHbKxJw!!&v=MjkwNzdXTTFGckNVUkxPZVp1ZHVGeXZnVzdyT0xqWGZmYkc0SDlqTnI0OUZZWVI

[6] Yali Nie et al. “Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks”. In: *2019 E-Health and Bioengineering Conference (EHB)*. IEEE, Nov. 2019. DOI: 10.1109/ehb47216.2019.8970033. URL: https://doi.org/10.1109/ehb47216.2019.8970033.

[7] Dewi Putrie Lestari et al. “Fire Hotspots Detection System on CCTV Videos Using You Only Look Once (YOLO) Method and Tiny YOLO Model for High Buildings Evacuation”. In: *2019 2nd International Conference of Computer and Informatics Engineering (IC2IE)*. IEEE, Sept. 2019. DOI: 10.1109/ic2ie47452.2019.8940842. URL: https://doi.org/10.1109/ic2ie47452.2019.8940842.

[8] Wenbo Lan et al. "Pedestrian Detection Based on YOLO Network Model". In: *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, Aug. 2018. DOI: 10.1109/icma.2018.8484698. URL: https://doi.org/10.1109/icma.2018.8484698.

[9] Jia-Ping Lin and Min-Te Sun. "A YOLO-Based Traffic Counting System". In: *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. IEEE, Nov. 2018. DOI: 10.1109/taai.2018.00027. URL: https://doi.org/10.1109/taai.2018.00027.

[10] Wang Yang and Zheng Jiachun. "Real-time face detection based on YOLO". In: *2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)*. IEEE, July 2018. DOI: 10.1109/ickii.2018.8569109. URL: https://doi.org/10.1109/ickii.2018.8569109.

[11] Hiroki Nakahara, Masayuki Shimoda, and Shimpei Sato. "A Demonstration of FPGA-Based You Only Look Once Version2 (YOLOv2)". In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Aug. 2018. DOI: 10.1109/fpl.2018.00088. URL: https://doi.org/10.1109/fpl.2018.00088.

[12] Xilinx. *Vitis Unified Software Platform*. [Online; accessed 24-04-2020]. URL: https://www.xilinx.com/products/design-tools/vitis.html.

[13] *Vitis - FPGA - Digilent Forum*. [Online; accessed 24-04-2020]. URL: https://forum.digilentinc.com/topic/19447-vitis/.

[14] Xilinx. *Frequently asked questions*. [Online; accessed 24-04-2020]. URL: https://github.com/Xilinx/Vitis-AI/blob/master/doc/faq.md.

[15] *I want to use ZC702(Zynq-7000) with Vitis AI - Community Forums*. [Online; accessed 24-04-2020]. URL: https://forums.xilinx.com/t5/AI-and-Vitis-AI/I-want-to-use-ZC702-Zynq-7000-with-Vitis-AI/m-p/1077172.

[16] Giovanni Guasti Antonello Di Fresco. *Long Form Answer Record73058: ResNet-50CNN applicationimplemented on a ZedBoard using Vivado and PetaLinux 2019.2*. [Online, accessed on 20-04-2020]. URL: https://www.xilinx.com/Attachment/73058_Porting_a_ResNet-50_CNN_application_to_a_ZedBoard_2019_2.pdf.

[17] Xilinx. *DNNDK User Guide*. [Online; accessed: 20-04-2020], UG1327 (v1.6) August 13, 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/ai_inference/v1_6/ug1327-dnndk-user-guide.pdf.

[18] Xilinx. *YOLOv3 Tutorial: Darknet to Caffe to Xilinx DNNDK*. [Online; accessed: 20-04-2020]. URL: https://github.com/Xilinx/Edge-AI-Platform-Tutorials/tree/3.1/docs/Darknet-Caffe-Conversion.

[19] Wikipedia contributors. *Xilinx — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-20-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Xilinx&oldid=950507093.

[20] Himanshu Sharma. *Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning*. [Online; accessed 24-04-2020]. 2019. URL: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

[21] Avnet. *ZedBoard*. [online; accessed 24-04-2020]. URL: http://zedboard.org/product/zedboard.

[22] Digilent. *Digilent academic verification | Login*. [Online; accessed 27-04-2020].

[23] Avnet. *ZedBoard(Zynq^{TM}Evaluation and Development)Hardware User's Guide*. [Online; accessed on: 20-04-2020]. URL: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[24] Xilinx. *Vivado Design Suite UserGuide Release Notes, Installation, and Licensing*. [Online; accessed: 20-04-2020], UG973 December 17, 2019. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug973-vivado-release-notes-install-license.pdf.

[25] Xilinx. *Petalinux User Guide*. [Online; accessed : 20-04-2020], UG1144, December 5, 2018. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1144-petalinux-tools-reference-guide.pdf.

[26] Joseph Redmon and Ali Farhadi. *YOLO: Real-Time Object Detection*. [Online; accessed 12-04-2020]. 2018. URL: https://pjreddie.com/darknet/yolo/.

[27] Avnet. *ZedBoardTM Getting Started Guide*. [Online; accessed on: 20-04-2020]. URL: http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf.

[28] Chathura Niroshan. [Online; accessed 12-20-2020]. URL: https://medium.com/developments-and-implementations-on-zynq-7000-ap/install-ubuntu-16-04-lts-on-zynq-zc702-using-petalinux-2016-4-e1da902eaff7.

[29] Sergio Lima Netto Rafael Padilla and Eduardo A. B. da Silva. “Survey on Performance Metrics for Object-Detection Algorithms”. In: (2020).

[30] Lembitu Valdmets. *YOLOv3_ZedBoard_DNNDK*. [Online; accessed 18-05-2020]. URL: `https://github.com/LembituValdmets/YOLOv3_ZedBoard_DNNDK`.

[31] Xilinx. *Machine learning for embedded deep dive*. 2018. URL: `https://www.xilinx.com/publications/events/developer-forum/2018-frankfurt/machine-learning-for-embedded-deep-dive.pdf`.

[32] Sumit Saha. *A guide to convolution arithmetic for deep learning*. [Online; accessed 05-05-2020]. 2018. URL: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[33] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. eprint: `arXiv:1603.07285`.

[34] Xilinx. *itis AI Library User Guide*. [online; accesses 20-04-2020], UG1414 (v1.0) December 18, 2019]. URL: `https://www.xilinx.com/support/documentation/sw_manuals/vitis_ai/1_0/ug1414-vitis-ai.pdf`.

[35] Xilinx. *DPU for Convolutional Neural Network v3.0 DPU IP Product Guide*. [Online; accessed 20-04-2020], PG338 (v 3.0) August 13, 2019. URL: `https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf`.

[36] Rainer Grimm. *The Three Clocks*. [Online; accessed 24-20-2020]. 2016. URL: `https://www.modernescpp.com/index.php/the-three-clocks`.