

ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR

Rakotojaona Andrianoelisoa Nambinina

Monal Patel

Prof. Carlos Valderrama

ABSTRACT

Enhancement of night-time video using Dark Channel Prior IP accelerator is proposed in this project. Nighttime video processing is difficult due to low brightness, low contrast and high noise in the video. The above problems may affect the accuracy and may result in failures of object detection or object classification at night time. Dark Channel Prior (DCP) filter is used to improve the visibility, brightness, and contrast of the night-time input video. Processing speed is a challenging task on the real-time application of the DCP algorithm for night time video enhancement. Hence, the DCP algorithm is implemented on FPGA (ALVEO Board) to increase the speed of video processing. In order to demonstrate the quality of the proposed method, a practical environment with a Vitis software tool and Alveo board was also set up to evaluate the performance of hardware. Such extensive experimental results indicate that the proposed algorithm and hardware structure are effective, feasible and straightforward to apply to practice.

TABLE OF CONTENTS

Cover Page	i
Certificate	ii
Thesis Approval Certificate	iii
Paper publication Certificate	iv
Acknowledgment	v
Abstract	vi
Table of contents	vii
List of figures	viii
List of tables	ix
Chapter 1: INTRODUCTION	1
1.1 Overview of enhancement of night time video IP accelerator	1
1.2 Image processing	3
1.3 Reconfigure hardware Acceleration IP	4
Chapter 2: STATE OF ART	5
2.1 Literature survey of night video enhancement	5
2.2 Discussion of the method chosen for enhancement of night time	7
Chapter 3: THE DCP ALGORITHM	10
3.1 Mathematical model of DCP algorithm	10
3.2 The DCP video processing flow	13
Chapter 4: HARDWARE, TOOLS AND LANGUAGES	14
4.1 Vitis unified platform	14
4.2 Vitis software installation	16
4.3 Vitis vision library	17
4.4 OpenCL and C++ language	18
4.5 Alveo Board	19
Chapter 5: THE DCP HARDWARE SOFTWARE IMPLEMENTATION	22
5.1 System architecture	22
5.2 Components used for the enhancement of night time on FPGA	22

5.3 Global architecture of the proposed method	24
5.4 Design flow of the proposed method	25
5.5 Performance parameters	27
Chapter 6: RESULTS EVALUATION	29
Chapter 7: CONCLUSION	35
Chapter 8: DESCRIPTION OF ANNEX	38
<i>Annex 1</i> Activity report	39
<i>Annex 2</i> Integration of RTL kernel in application program	40
<i>Annex 3</i> Pynq Board Descriptions	50
<i>Annex 4</i> Building the host and the kernel	53
<i>Annex 5</i> Publications	59
<i>Annex 6</i> Plagiarism report	60

LIST OF FIGURES

Figure No	Figure Description	Page No
1.2.1	Enhancement of night time	4
1.3.1	Architecture of FPGA	5
3.1.1	Atmospheric scattering	10
3.1.2	Inversion of input frame 3 x 3	11
3.2.1	Chart flow of proposed algorithm	13
4.1.1	Global architecture of host and kernel	15
4.1.2	Architecture of Vitis unified software	15
4.4.1	Structure of OpenCL	18
4.5.1	Result of lspci- command	20
4.5.2	Card found after validate command	21
5.1.1	<i>Host and Kernel</i>	22
5.2.1	Alveo board	23
5.2.2	Host	23
5.2.3	PCIe	23
5.2.4	Power supply	23
5.2.5	FPGA interconnected with Host	24
5.3.1	Global architecture of proposed method	25
5.4.1	Design flow of host /Kernel	26
5.4.2	Host-Kernel Dataflow	27
6.1	Input Frame	23
6.2	Dark input frame	23
6.3	Medium transmission	24
6.4	Enhanced Frame	24

LIST OF TABLES

Table No	Table Description	Page No
1.1.1	Comparison of CPUs/GPUs/FPGAs/ASICs	2
2.1.1	Literature survey	6
6.1	Processing time comparison of DCP implemented on CPU/FPGA	30
6.2	Summary of resources used on FPGA	31
6.3	Timing summary of apc_clk	31
6.4	Latency	31
6.5	Details of the resources used on FPGA	32

CHAPTER 1

INTRODUCTION

1.1 Overview of enhancement of night time video IP accelerator

Object detection in real-time video is one of the prevalent applications in the field of video processing. The video captured at nighttime has a low illumination and usually suffers from poor visibility. The variety of algorithms for object detection is available and provides good results on still images. The algorithms are required to be accelerated for object detection in real-time video. Several methods can be used for the enhancement of nighttime such as infrared and gamma correction, histogram equalization, DFT and many more. Dark Channel Prior is used in this thesis due to its simplicity, in addition is one of the fastest algorithms for hardware implementation. Different types of hardware can be used for implementation such as CPU, GPU, FPGA and ASIC (Table 1.1.1 describes the differences between CPU, GPU, FPGA, and ASIC) [1]. ASIC is the best hardware for the implementation in terms of speed and size, ASICs has the highest speed and the lowest power consumption, while in term of flexibility ASIC can not be reconfigurable, it is dedicated only for a specific application and the cost of it is very high which makes the FPGA a good candidate for the hardware implementation. FPGA offers many advantages over traditional CPU/GPU acceleration, FPGA allows the user to create a custom architecture capable of implementing any function, and it also consumes less power compared to GPU. The main objective of this thesis is to accelerate the video processing speed of input video using the Dark Channel Prior Algorithm. Xilinx unified development environment is used to target the Xilinx Alveo board for the hardware implementation. The Vitis unified software tool is a new tool of Xilinx, that combines all the aspects of Xilinx software and hardware into one environment known as Vitis.

Table 1.1.1 Comparison of CPUs /GPUs /FPGAs/ ASICs [3]

Criteria	CPUs	GPUs	FPGAs	ASICs
Processing Peak Power	Moderate	Very High	High	Highest
Power consumption	High	Very High	Very Low	Low
Flexibility	Highest	Medium	Very High	Lowest

Vitis development environment is used in this thesis. Vitis allows the development of a hardware accelerator to be used by the embedded software flow, the Xilinx Software Development Kit (SDK). If users are looking to move into the next generation technology then application acceleration development flow is the latest in Xilinx FPGA-based software acceleration. Here Vitis development kit is used to develop an application program, the software and the hardware are interconnected through a driver known as Xilinx runtime. The new tool provides a framework as a Caffe tensorflow for developing an application on FPGA. A framework is a software that provides a large sample of code which makes the development of an application program easier using existing codes. In Vitis software, the application program is split into host and kernel. Both software and hardware can be used as high-level programming using C++ for hardware acceleration. High-level programming uses libraries that make the development easier. In this project, C++ language is used to develop an application program for both host and kernel.

This thesis is structured in 8 Chapters. In chapter 2, discusses the state of art done to find the gap of recent approaches. In chapter 3, discusses the algorithm used for the enhancement of night time. We will cover the theoretical background of the DCP algorithm and the data flow of the algorithm. Chapter 4, presents a short description of the tool and the hardware used for the implementation as well as the languages used for the development of the application program. Chapter 5, presents the implementation of the DCP algorithm into hardware and the data flow between the host and the kernel. Chapter 6, presents the experiment result and the

discussion of the result during hardware implementation. Chapter 7, concludes the thesis and future work. Chapter 8, presents the annex

1.2 Image processing

Image processing is used to enhance the received images from cameras or sensors. Different techniques have been developed in image processing during the last decade. Most of the techniques are developed for enhancing images to object recognition on flights and spaces. While in this project we focused only on the enhancement of night views in video images. Image processing is becoming popular due to the advanced technology of computer vision. Image processing is used in various application such as :

- Remote sensing
- medical imaging
- nighttime enhancement
- non-destructive evaluation
- graphic arts
- printing in the industry (and many more)

The schematic diagram of enhancement of night time is shown in Fig.1.2.1, The input frame is enhanced by the DCP algorithm implemented on the FPGA platform placed between the camera and the monitor.

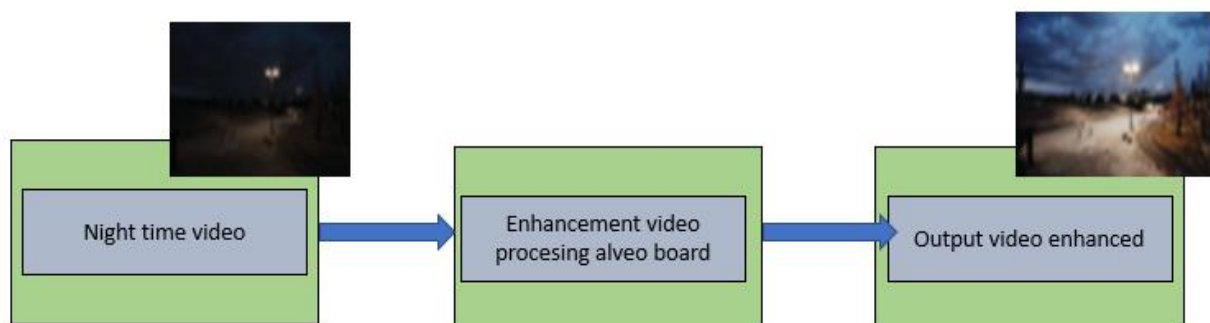


Fig 1.2.1 Enhancement of night time

1.3 Reconfigure hardware Acceleration IP

The hardware implementation of IP can be done by using reconfigurable digital circuits such as FPGAs. The FPGA is a reconfigurable integrated circuit that can be used to create a component to accelerate the execution of an application program. Basically, an FPGA has three main parts (Strikethrough Fig. 1.3.1) :

- Configurable Logic Block CLB: CLB provides physical support for the program downloaded on FPGA, this block can be reconfigurable as the user wish
- Input-Output BlockIOB: This block is responsible for the communication between the external and the FPGA which provides input and output for FPGA
- Programmable Interconnect and switches matrix SM:This block is used to interconnect every logic block inside of the FPGA, and this block also can be reconfigured as the user wishes.

The choice to build an enhancement of nighttime video in digital hardware comes from several advantages that are typical for digital systems. Digital designs have the advantages of low noise sensitivity, low latency, and fast video processing speed. With the advance in programmable logic device technologies, FPGAs have gained much interest in digital system design [14]. FPGA-based reconfigurable computing architectures are suitable for hardware implementation of enhancement of nighttime video with a real-time output video still a challenging task. FPGA is an excellent technology for the implementation of image processing.

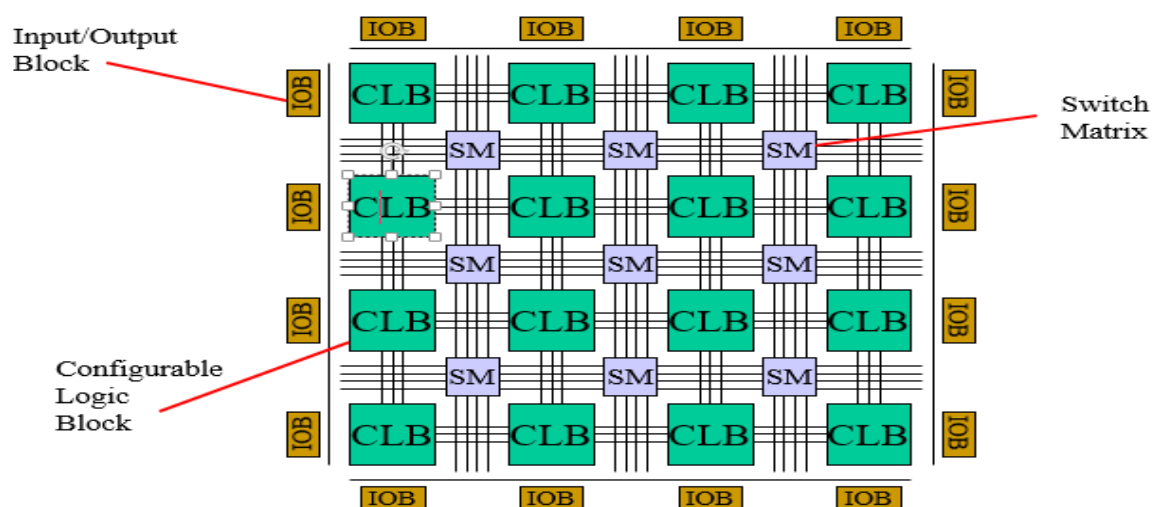


Fig 1.3.1 Architecture of FPGA [15]

1.4 Motivation

The improvement in technology of image processing at night makes the hardware and software engineers focus on the processing speed of an application program that can be used to enhance night-time vision in real-time. A digital video is a sequence of images with a constant time interval that can become critical depending on the size of the frames. After studying the literature, it is seen that enhancement of the night input frame with real-time output video is really a challenging task for a computer vision application. Video processing consumes more time on a software implementation (CPU), due to the limited processing power of non dedicated processors such as CPUs. Using an FPGA for the hardware implementation is one of the alternatives not just to accelerate the speed of video processing, but also to rapidly prototype a feasible solution.

CHAPTER 2

STATE OF ART

2.1 Literature survey of night video enhancement

I did a literature survey of different methods which can be used for the enhancement of nighttime for choosing the best algorithm which can be used for hardware implementation. The literature shows multiple algorithmic proposals seen as potential candidates for hardware implementation in terms of complexity, quality, and processing speed.

The table 2.1.1 below describes some of the algorithms found as well as the pros & cons when considered for hardware implementation.

Table 2.1.1 Literature survey

Reference	Author	Year	Method	Cons	Pros
[17]	Tan	2008	Contrast enhancement	1.Oversaturation 2. Halos effects due to the patch-based operation	Good contrast for the foggy image
[18]	Fattal	2008	Independent component analysis	1. Can not recover the gray image 2.Low brightness resulting 3. Can not enhance the image with dense foggy and insufficient signal to ratio 4. Not suitable for real-time	The visibility of the output frame is sufficient
[19]	Tarel	2009	Contrast enhancement	1. Fail for enhancement of images with discontinuous depth 2. Over enhancement 3. Halos effects	Fast and good visibility
[5]	Jiang	2009	Dark Channel Prior	1. Low brightness of the output 2. Dedicated only for the enhancement of foggy image	Fast and simple algorithm

[20]	Fattal	2014	Color lines	1. Low brightness 2. Not suitable for the gray input image	High image visibility
[21]	Tang	2014	Learning-based	Not suitable when there is an important thicker haze appears in the input image	1. Good contrast for hazy frame
[22]	Zhu	2015	Color attenuation Priors	Not suitable when there are an important thicker haze appears in the input image	1.The depth information can be well recovered 2.Recover the estimation transmission and the scene radiance easily
[23]	Cai	2016	Dehaze Net	1. A small error in air-light will drop the performance 2. Enhanced single image dark colors	Better restore the sky and white area
[24]	Berman	2016	Non local dehazing	1. May fall in scenes where the air-light is significantly brighter than the scene	Work well at a certain haze level

2.2 Discussion of the method chosen for enhancement of night time

Enhancement of dark input video improves the quality of the night time video and, a few years ago, a variety of different algorithms were used to enhance a nighttime video [3], [4], [5], [6], [7], [8], [9] [10], [11], [12], [13], [14]. They are here grouped according to the following criteria such as complexity, quality, processing speed and cost

Infrared is one of the techniques used to improve the quality of dark input images or videos. Infrared is an electronics dispositif used to detect any images with a higher temperature than its surrounding, it is also used to enhance the quality of the input frame by converting the infrared energy to an electronic signal. [3]. While infrared cameras are expensive compared to the normal cameras. This disadvantage limits the scope of their applications and future development.

We can also use traditional image processing techniques to improve the quality of night input images or videos. Histogram equalization and gamma correction are mostly used for the traditional image processing, Ko et. al. proposed methods for removal of noise motion adaptive temporal filtering based on the Kalman structured updating. By adaptive adjustment of RGB histograms causes the increment in the Dynamic range of denoised video. ASIC is the best hardware for the implementation in terms of speed and size, ASICs has the highest speed and the lowest power consumption, while in term of flexibility ASIC can not be reconfigurable, it is dedicated only for a specific application and the cost of it is very high which makes the FPGA a good candidate for the hardware implementation. mately, the remaining unwanted factor which is noise can be removed using Non-local means (NLM) denoising. This technique exploits color filter arrays (CFA) to minimize the resources used such as (memory consumption). The final experimental results indicate that this method is highly promising for various real-time applications to consumer digital cameras, especially CCTV and the surveillance video system [4]. Although the rumored pleasing results for low lighting pictures and videos, they still inevitably introduce undesirable halo effects and excessive improvement development [5]. Bhagya et al. proposed video enhancement using histogram equalization with JND model and this technique, besides being used to achieve visually pleasant enhancement effects, the over-enhancement and saturation artifacts are avoided in this method [6]. However, this method is complex, and it requires a large size of hardware. R. Peng et al. proposed a contrast stretching method which improves the quality of the image by stretching the intensity range, while it stretches the lower intensity pixels to lower and higher intensity pixels to higher. As each value in the input image can have several values in the output image, so objects in the original image may misplace their relative brightness values [7]. Choi et al. proposed a single scale retinex which is one of the newly emerging techniques in the enhancement field. The advantage of this method is the speed of execution. But there is also a limitation with Single retinex, as it deals with dynamic range enhancement and color interpretation, but fails in achieving both together [8].

Rahman et al. proposed multiple-scale retinex to solve this problem of Single scale retinex but this method fails to produce good quality of enhancement of input videos or images, it finds input videos or images having a high spectral characteristic in the single band [9].

Another alternative is to use the DCP algorithm which is based on the DCT. The DCP is an algorithm based on the observation of an outdoor haze-free image, in which at least one color channel of the frame has some pixels very low and close to zero. Jiang et al. proposed the enhancement of dark input video based on the DCP algorithm. This method is simple and improves the quality of nighttime video or images, but it's quite slower for real-time video processing for ultra-high-definition video [10]. Due to the low speed of video processing on CPU, our proposed method is to develop an accelerator for the DCP algorithm, to accelerate the video processing of night time, to get a real-time output video

Conclusion: After a review of the literature we can say that DCP is one of the best candidates for hardware implementation, due to its simplicity, and it's also fast compared to another algorithm available. While in terms of quality of output frame, DCP is not one of the best algorithms for the enhancement of nighttime, this algorithm can give a good enough quality of output frame which we can use in further processing such as video tracking, object detection, and many more.

CHAPTER 3

THE DCP ALGORITHM

3.1 Mathematical model of DCP algorithm

In this project, DCP algorithm to be used for enhancement of night input frames, the selection of this algorithm is due to the reasons depicted in the state of the art section (Chapter 2). This algorithm enhances a night input frame by considering the minimum pixel value of RGB. As Xu et al. inversion of the night input frame is quite similar to a foggy frame. Here we use the equation of McCartney to recover the foggy input frame.

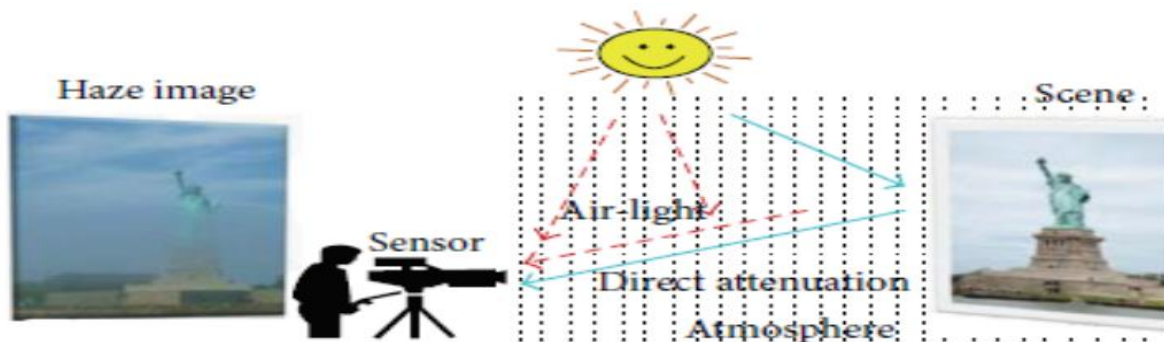


Fig .3.1.1 Atmospheric scattering [16]

Fig 3.1.1 shows that the input image captured by the sensor is not clear due to the existence of particles in the aerosol, the fine particles in the air making the quality of the image not well visible. In this case, once the image is taken, the camera absorbs the sunshine shut and scattered by these region particles, we call this technique as airlight.

In this thesis , $I(x)$ represents the night input frame (the frame is represented by the basic color components R, G and B per pixel), and $I_{inv}(x)$ is the inversion of the night input frame

(RGB). In this project we consider foggy frame as an input frame and used it in the implementation hardware and software.

We used the equation of McCartney to recover the enhanced output frame:

$$I_{inv}(x) = J_{inv}(x) + A(1 - t(x)) \quad (1)$$

Where,

$J_{inv}(x)$: Scene radiance (Output frame after enhancement) at position x (i,j),

A : Global atmospheric light (highest pixel value of the input frame),

$t(x)$: medium transmission at x (function used to enhanced the noise),

$I_{inv}(x)$: foggy input frame.

The aim of this algorithm is to recover J from the invert input image. The expression of the medium transmission is defined in equation 3 which depends on the dark channel of Invert input frame:

Using equation (1), we can express recover image ($J_{inv}(x)$):

$$J_{inv}(x) = \frac{I_{inv}(x) - A(1-t(x))}{t(x)} \quad (2)$$

With:

$$t(x) = 1 - \omega I_{dark}^c(x) \quad (3)$$

$$I_{dark}^c(x) = \min_{c \in \{r,g,b\}} \left(\frac{I_{inv}^c(x)}{A^c} \right) \quad (4)$$

Where:

ω : control parameter

I_{dark}^c : indicates a dark channel prior to the inverted frames.

Using the equations (3), (4), we can get the result of $J_{inv}(x)$. $J_{inv}(x)$ is the enhanced output frame. [10][11]

3.2 The DCP video processing flow

In this section, we are going to discuss the design flow of the DCP algorithm. Here we consider the input image as a foggy image. We are going to describes each function used in this project for the implementation of DCP on hardware

- **Dark channel prior :** This function is used to determine the minimum pixel value of foggy image (this frame has 3 channels R,G,B). In our implementation we execute the dark channel prior on hardware. The figure below describes the function used to determine the dark channel prior using the vitis vision library on FPGA.

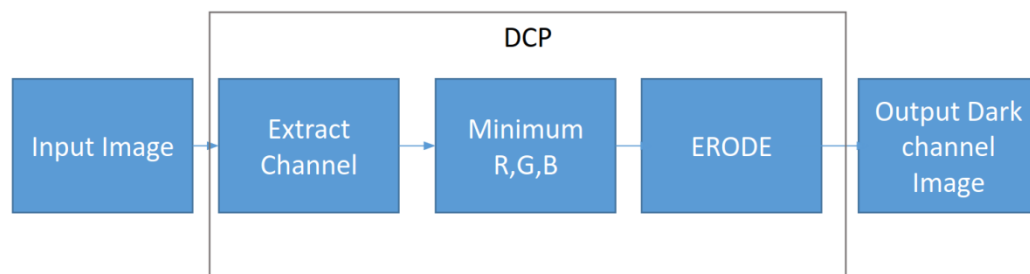


Fig .3.2.1 Dark channel prior

- **Medium transmission:** The output of DCP function is used in this function as an input to recover the medium transmission, this function is used to reduce the noise from any input images. Vitis vision library (xf::cv:: xf::cv::medianBlur) is used to determine the medium transmission on hardware.



Fig .3.2.2 Medium transmission

- **Dehaze function:** This function is used to recover the foggy input frame using the equation of McCartney(Described in section 3.1)

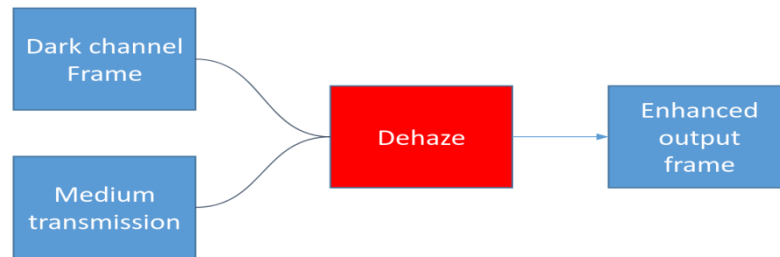


Fig .3.2.3 Dehaze function

The figure 3.2.3 below show the flow of DCP algorithm



Fig 3.2.4 chart flow of proposed algorithm

CHAPTER 4

HARDWARE, TOOLS AND LANGUAGES

4.1 Vitis unified platform

Vitis software is a development kit tool used to target hardware (FPGA ultrascale, Alveo board). This tool can be used to build an application program and run on a hardware accelerator. Vitis accelerated application consists of two components: the host which contains the software application program and the kernel which contains the compute unit. The application program in Vitis can be developed using :

- Pure software: In pure software the host code can be developed using high level languages using C/C++, OpenCL and Python for AI application, and the hardware can be developed using C++ language.
- Mixture of software and hardware: For the mixture of software/hardware, the host code is developed using OpenCL API and the kernel function is developed using RTL kernel

In this project, the host code and the kernel function are developed using C++ and OpenCL due to its simplicity. The Xilinx runtime or XRT (shown in Fig 4.1.2) is a driver used to manage the communication between the software system and the hardware. In Vitis software, the application program is split into host and kernel (as shown in Fig. 4.1.1), the host and the kernel are interconnected via Peripheral Component Interconnect Express, known as (PCIe). The host code is running on CPU, while the kernel function is running on FPGA. The following Fig.4.1.1 shows the global architecture of host and kernel, in this figure, we can see that the host has its own memory, this memory is accessible only from the host, while in kernel block, the kernel can have multiple kernels and each kernel can have multiple compute units, this compute unit is responsible to execute the beat stream on FPGA. The kernel also

has a local memory that is accessible only from the kernel. The constant global memory is known as share memory, this memory is accessible by the host and the kernel.

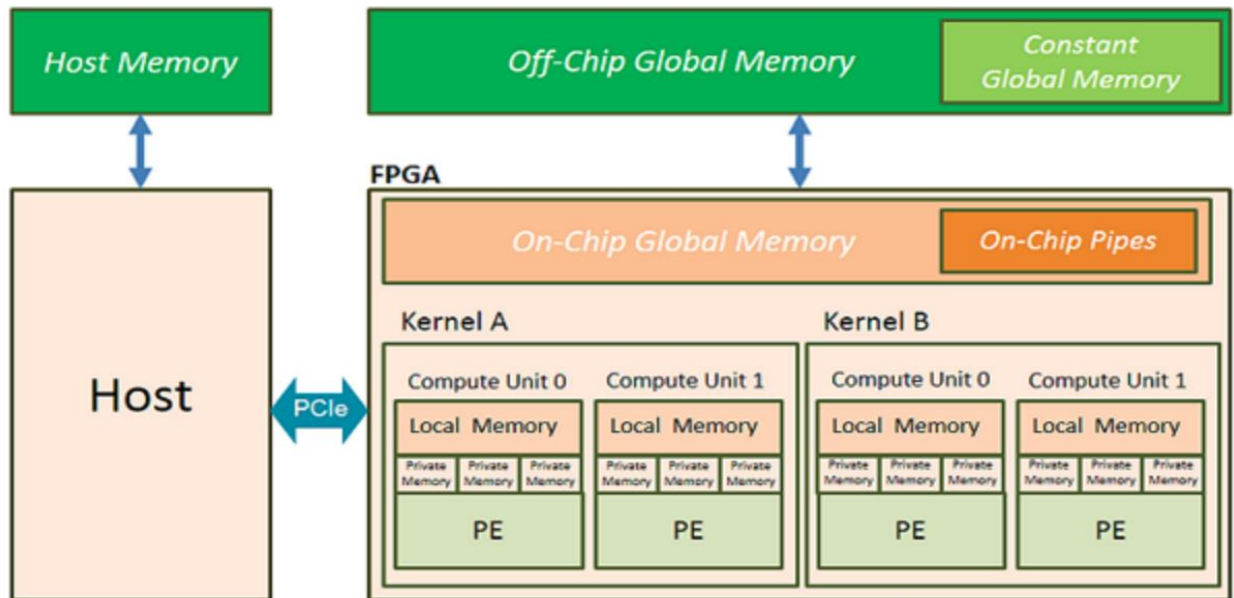


Fig 4.1.1 Global architecture of host and kernel

The Vitis software composed 3 layers :

- Vitis library: The Vitis library is used to develop an application program running on hardware, Vitis vision library is used in this project for the development of kernel function, Vitis software can be used in different platform on FPGA such as Zynq, ,Ultrascale+, and Alveo FPGAs. In this project, we focus only on the Alveo board acceleration.
- Vitis development kit: There are three blocks in Vitis development kit, the first block is the compiler which converts the high-level code into a Xilinx object file, and this Xilinx object file is linked with the platform (Alveo u-200-xdma) to generate a binary file (.xclbin). The second block is the analyzer, the analyzer report all of the time used during execution of the application program and the debuggers are used to locate any error during compiling the host and kernel function.
- Xilinx runtime: is a driver between software part and hardware which manage the communication between host and kernel (Alveo board)

The following Fig.4.1.2 shows the architecture of Vitis unified software

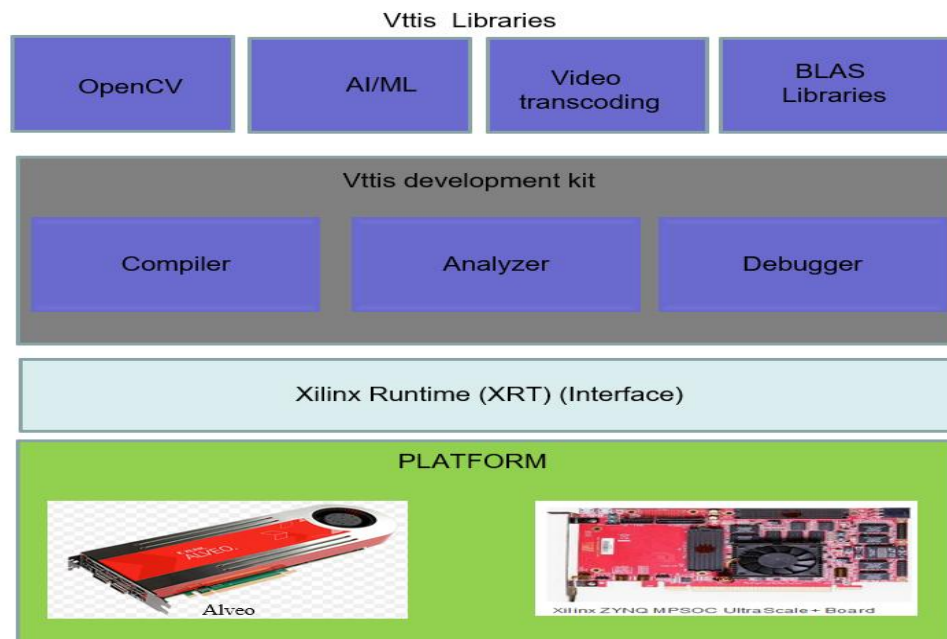


Fig.4.1.2 Architecture of Vitis unified software

4.2 Vitis software installation

Vitis unified software is used in this project to develop an application program (OpenCL API for the host code and C++ for the kernel function). To implement the application program on hardware we need to install a software tool, which allows us to target the hardware board. In this project, Vitis unified 2019.02 is used for the installation with Ubuntu 18.04 OS.

Following are the steps used for the installation of Vitis:

- Go to the website of Xilinx and download the Vitis unified using the following link (<https://www.xilinx.com/products/design-tools/vitis.html>)
- When we open this link many versions of Vitis software are available in the web site, while we download the latest version of vitis (2019.02)
- After that, we can start running the installer

- Accept the terms and the conditions by clicking each I agree on the checkbox and then click next
- Select the Vitis icon and then click next
- Customize your installation by selecting design tools and devices, here we selected Vitis accelerator , and then click next Select the installation directory, by using the optional shortcut and file association option, and then click next
- Review the installation summary, which shows the options and location you have selected
- Top proceeds with the installation of the vitis software platform, click install.
- When the installation has done, we can verify if the Vitis software installation has no problems during the installation by running the command below (this command is used to set up the Viits environment)

>\$ source *vitis*/path/installation

4.3 Vitis vision library

Vitis vision library is used in this project to develop the kernel function. Vitis vision is a pre-existing open source library developed by Xilinx developers. This library helps the user to build an application program by optimizing the source code available from Xilinx. This library is quite similar to a Computer Vision known as OpenCV. While in terms of speed, Vitis library is faster compared to OpenCV.

To write a vitis vision library in kernel function , we should follow the syntax described below:

- All arguments have to provide `xf::cv::Mat` in vitis vision (example: if we use an input image as an argument in kernel function: we have to convert the image array into Mat type:: `xf::cv::Mat`).
- The functions in vitis vision are following the format `xf::cv::` name of the function (example: `xf::cv::erode`, `xf::cv::max`, etc...)

the majority of the template are :

- The maximum size of image
- Data type of each pixel
- number of pixels going to process per clock cycle

- and other compile-time relevant to the functionality.

4.4 OpenCL and C++ languages

OpenCL or known as an Open computing Language is a framework used to develop an application program running on different platforms such as CPU, GPU, FPGA (Field Programmable Gate Arrays) and Digital signal processor (DCP) and other hardware accelerators or other processors.

In this project, the host code is developed using OpenCL, this language is used to target the Alveo board (FPGA platform), using following command such as : creation of the context, setting up of the kernel, writing the data into shared memory, executing the binary file on FPGA and many more. The figure below represents the structure of the OpenCL language.

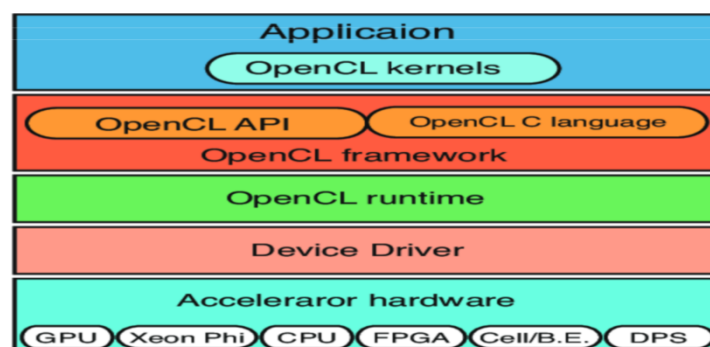


Fig4.4.1 Structure of OpenCL

Kernel functions can be developed using high-level programming (C++/C) or RTL kernel functions. C++ is selected here due to its simplicity, so it's easier to develop. In this project, C++ is used to build a kernel function. This language is fast compared to other high-level languages and it also provides several libraries such as OpenCV, which is used in this project. This language was developed by Bjarne Stroustrup in 1979 at the laboratory of Labs in Hill, New Jersey, it was an improvement of the C language and originally named C with Classes but later it was changed into C++ languages in 1983. C++ language is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language, generic

language, and object-oriented programming. It is also regarded as a middle-level language, as an amalgamation of high-level and low-level language features.

4.5 Alveo Board

Alveo board is an open-source FPGA introduced by the Xilinx in 2019, This board has low-level block resources as gates, flip-flops, and SRAM. The Alveo card is interconnected with the host (Desktop) through a PCIe. This board is used to accelerate an application program. In this project, enhancement of the night input frame to be accelerated on the Alveo board (FPGA). This board is quite easy to use because it supports Vitis software tool which allows us to build an application program using high-level languages such as C/C++ and OpenCL. With the help of Vitis vision, we do not need to start our code from scratch. We can optimize the source code provided by the Xilinx developer to develop our own program.

There are several types of Alveo cards available in the market such as Alveo u-50 , Alveo u-200, Alveo u-250, Alveo u-280.

Those Alveo boards are different by their performances and their power consumption. Here we used the Alveo board u 200 for the implementation. Installation of Alveo board on desktop and installation of packages (driver, deployment file) are required here :

The installation of the Alveo card on motherboard and installation of required packages are shown in the following steps:

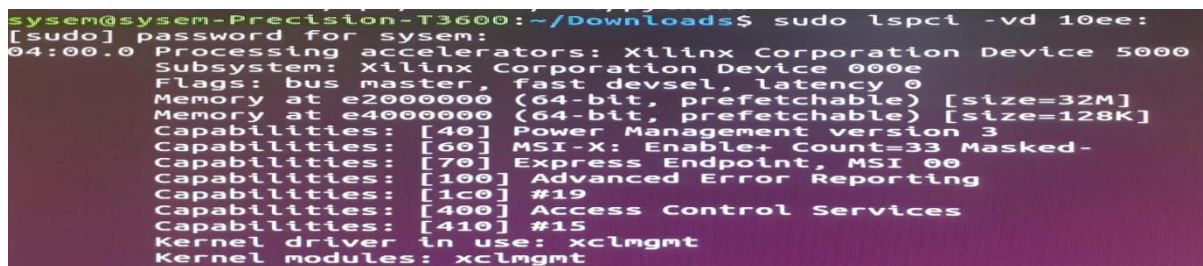
- Before installation of the Alveo board u200 on the motherboard we must verify the characteristic of our laptop if it meets the characteristic required by the Alveo board u200, the requirement is described below:
- Our desktop should have: PCIe 3.0 one dual-width x16 slots, 255W power supply, and minimum RAM of 64 GB, Hard disk should be minimum of 100 GB and internet connection
- Now we can start the procedure of installation by Turning off the power supply of the host (Desktop)
- Then open the central unit by removing the casing.
- Interconnect the U200 card on the motherboard through PCIe 16 slot and plug the power supply on the Alveo
- verify if all of the connection is plugged well
- And then turn on the desktop

- After installing the card on the motherboard, the card should be seen by the Operating system
- The command below is used to verify if the card is interconnected well with the host.

To do that we need to run the command below on terminal :

```
>$ lspci -vd 10ee
```

If the installation of the card is correct well, we should have the same result like this (Fig 4.5.1).



```
syssem@syssem-Precision-T3600:~/Downloads$ sudo lspci -vd 10ee:
[sudo] password for syssem:
04:00.0 Processing accelerators: Xilinx Corporation Device 5000
Subsystem: Xilinx Corporation Device 000e
Flags: bus master, fast devsel, latency 0
Memory at e2000000 (64-bit, prefetchable) [size=32M]
Memory at e4000000 (64-bit, prefetchable) [size=128K]
Capabilities: [40] Power Management version 3
Capabilities: [60] MSI-X: Enable+ Count=33 Masked-
Capabilities: [70] Express Endpoint, MSI 00
Capabilities: [100] Advanced Error Reporting
Capabilities: [1c0] #19
Capabilities: [400] Access Control Services
Capabilities: [410] #15
Kernel driver in use: xclmgmt
Kernel modules: xclmgmt
```

Fig 4.5.1 Result of lspci command

- Now we have confirmed that the card is correctly installed, we need to install all of the required drivers and deployment files. To do that we need to download the Xilinx run time packages and the deployment package from Vitis official web page. Then we can start the installation using the command below:

```
>$ sudo apt install ./path/of/xrt.deb
```

```
>$ sudo apt install ./path/of/deployment/u200.deb
```

- After installation of the driver and the deployment, we have to flash manually the kernel, to test the board, now we are ready to test the board by configuring a test kernel, to do that we used the flash procedure (write the first kernel into FPGA memory) :

```
>$ sudo /opt/xilinx/xrt/xbutil flash -a xilinx_u200_xdma -t 1535712995
```

- After flashing the card, we need to reboot our desktop.
- Now we can verify if the card function correctly, to do that we use the command below :

```
> $ validate -d card_id
```

- If the card functions correctly, we should have the same figure (Fig. 4.5.2) on screen which shows that the verification of the kernel is passed.

```
sysen@sysen-Precision-T3600:~/Downloads$ sudo /opt/xilinx/xrt/bin/xbutl validate -d 0000:04:00.0
INFO: Found 1 cards

INFO: Validating card[0]: xilinx_u200_xdma_201830_2
INFO: == Starting AUX power connector check:
INFO: == AUX power connector check PASSED
INFO: == Starting PCIE link check:
INFO: == PCIE link check PASSED
INFO: == Starting SC firmware version check:
INFO: == SC firmware version check PASSED
INFO: == Starting verify kernel test:
INFO: == verify kernel test PASSED
INFO: == Starting DMA test:
Host -> PCIe -> FPGA write bandwidth = 8089.01 MB/s
Host <- PCIe <- FPGA read bandwidth = 12161.7 MB/s
INFO: == DMA test PASSED
INFO: == Starting device memory bandwidth test:
```

Fig 4.5.2 Card found after validate command

CHAPTER 5

HARDWARE SOFTWARE IMPLEMENTATION OF DCP ALGORITHM

5.1 System architecture

Enhancement of night time video to be accelerated on Alveo board (FPGA), our proposed system has three main parts: the host part, the kernel part and the global memory (known as shared memory), this shared memory is on FPGA. The host and the kernel are interconnected through PCIe. The figure below depict the system architecture of the Host/Kernel/Global memory.

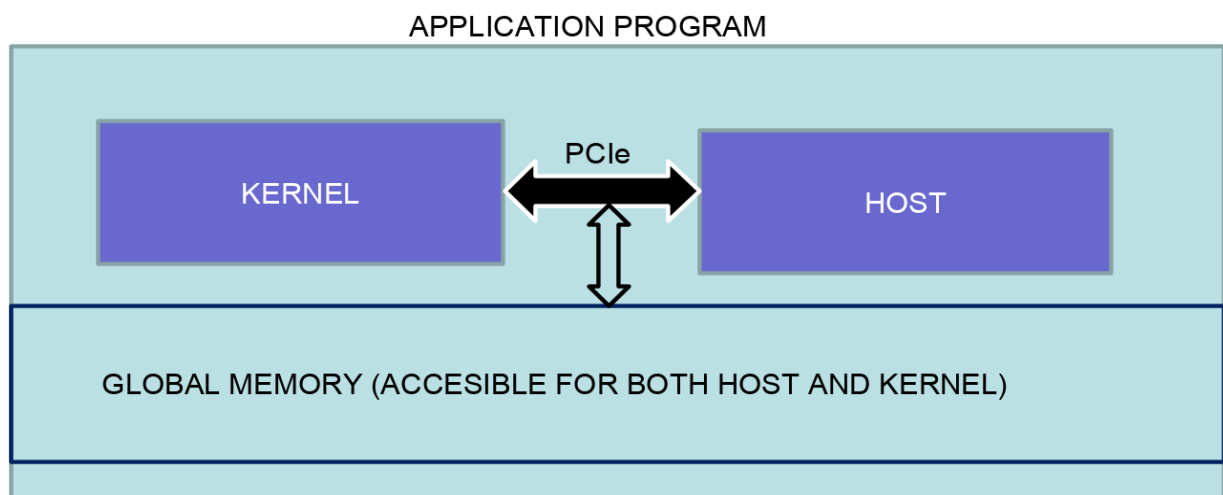


Fig 5.1.1 Host and Kernel

5.2 Component used for the enhancement of night time on FPGA

Some required component is used in this project to build the application program on hardware, the following are the required component with their characteristics :

- FPGA: Alveo u200 card is the platform used in this project with 182240 LUT , 2364480 FF , BRAM 6840 (18K)
- Host: Dell Core i7 -2.7GHZ with Ubuntu OS 18.04 ,16 Go RAM, PCIe connector

- Power supply cables: The Alveo card u200 required 225W from the host (Desktop)
- PCIe (Interconnect the host and the FPGA): Peripheral Component which interconnects the host (Desktop) with Alveo card (FPGA)
- Vitis software: Vitis software tool is used in this project to target the Alveo board (FPGA).

Fig. 5.2.1-5.2.4 shows all of the components required for the hardware implementation.



Fig 5.2.1 Alveo board



Fig 5.2.2 Host



Fig 5.2.3 PCIe



Fig 5.2.4 Power supply

Fig. 5.2.5 show the picture of the FPGA interconnected with the Host through PCIe

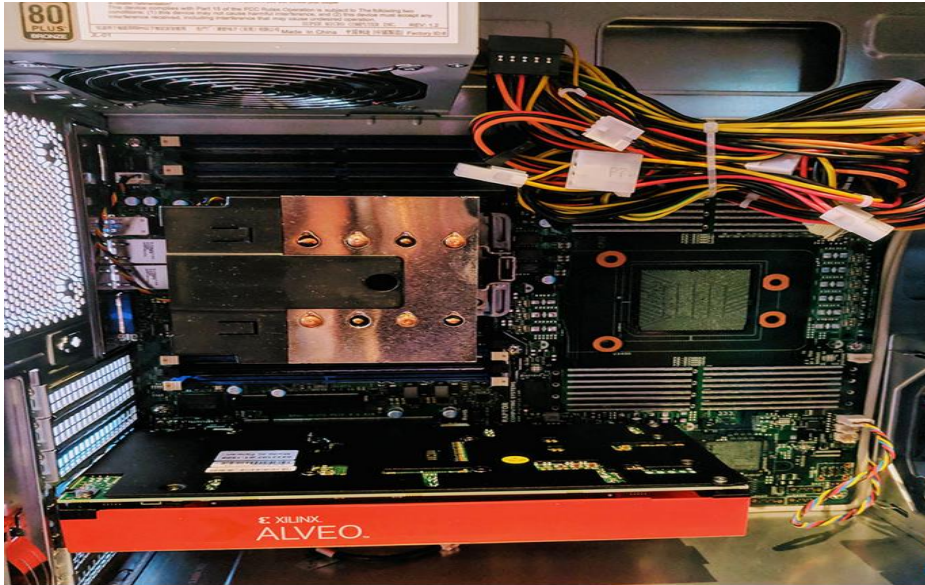


Fig 5.2.5 FPGA interconnected with Host

5.3 Global architecture of proposed method

In our global architecture, the host and the kernel are responsible for the execution of an application program, the host code is running on CPU, while the kernel function is running on FPGA, the steps below describe the data flow in our global architecture method.

- The host application writes the input data (input frame) into a global memory of the attached device through the PCIe interface
- After that the host set up a command to the kernel to execute the DCP and medium transmission of the input image
- The kernel executes the data using vitis vision library and writes the output data to the shared memory .
- The host application reads the data from global memory and then execute the Dehaze function in host code and then write the output enhancement into host memory

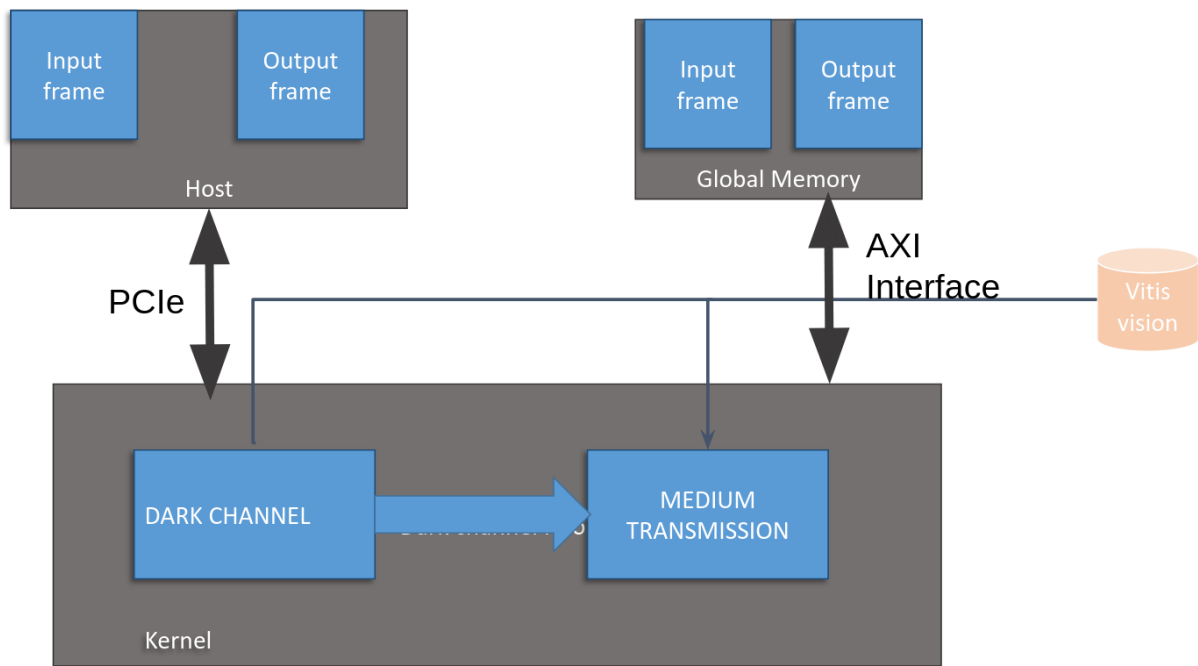


Fig 5.3.1 Global architecture of proposed method

5.4 Design flow of proposed method

As shown in the figure below (Fig. 5.4.1). There are 3 different layers in our proposed design flow :

- User applications: Application program and the library used for the development of application (Vitis vision)
- Tools : Vitis software unified tool
- Platform : Alveo board u200 (FPGA) and Dell 7000 core i7 with 2.7 GHz (CPU)

In this project the blue color box describes the flow of the data in the host and the gray color describes the flow of the data on hardware, as seen in Fig. 5.4.1, the host code and the kernel function are the top-level layers, which can be accessible by the developer, the kernel function used the Vitis library to execute the application program. The second layer is the tool, the tool is used to compile and link the application program, in our case the host code and the kernel function are compiled separately, the host used the GCC compiler to generate an executable file (.exe) from a high level (.cpp) to an executable file known as (.exe), while the kernel used the Vitis compiler to compile the high-level function, the Vitis compiler call the Vivado High-level synthesis (HLS) to convert the high-level code (.cpp) into Xilinx object file (.xo), the Xilinx object file is linked with the platform u200 using v++ linker, to

generate beat stream which can be run on FPGA. The third layer is a platform, the platform is hardware that used to execute the executable file, here the executable file of the host is running on CPU (core i7) and the beatstream is running on FPGA (Alveo card)

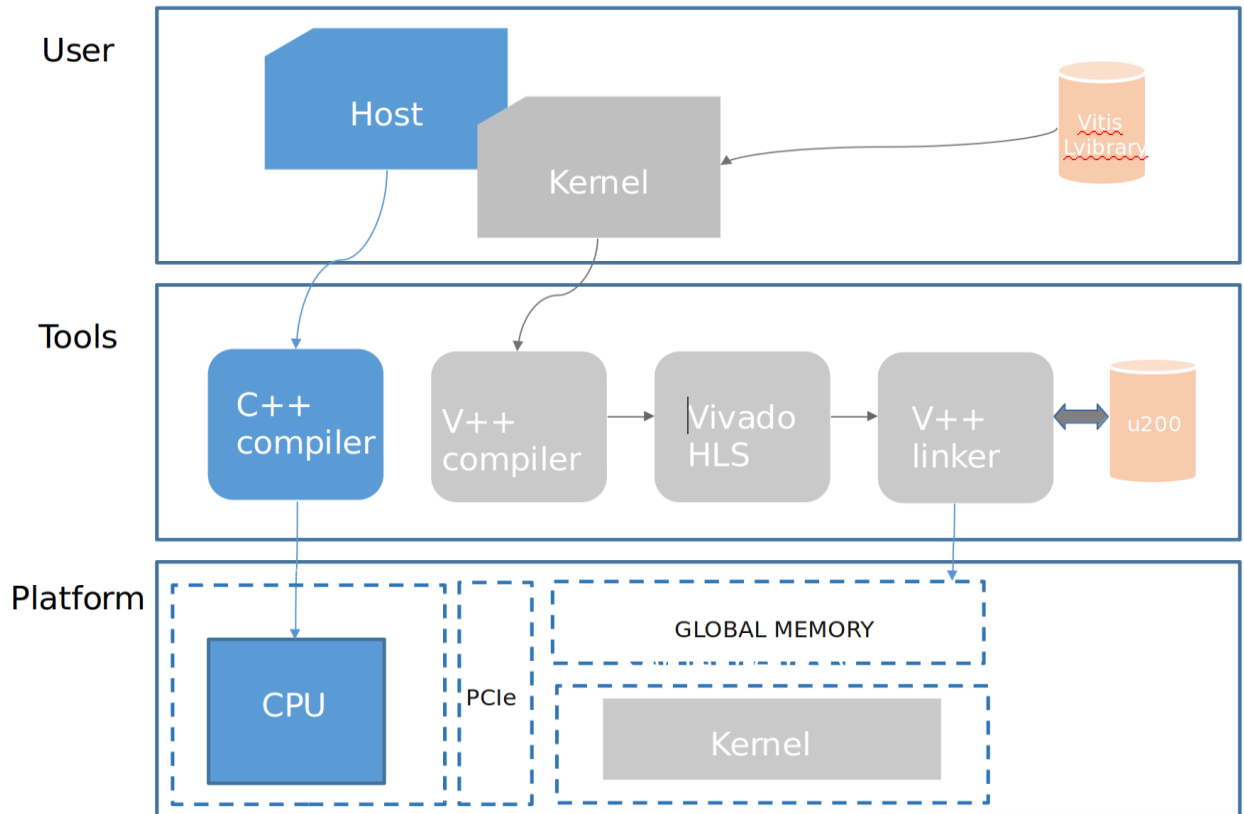


Fig 5.4.1 Design flow of host /Kernel

The figure below (Fig. 5.4.2) represents the flow of the data between the host and the kernel. The processing time of the DCP and Medium transmission function required a lot of time on software using CPU. In our proposed method we accelerate the DCP and Medium transmission function on hardware using FPGA (Alveo board) and accelerate the dehaze function or recover function on software using CPU. The figure below described the host-kernel data flow on software and hardware.

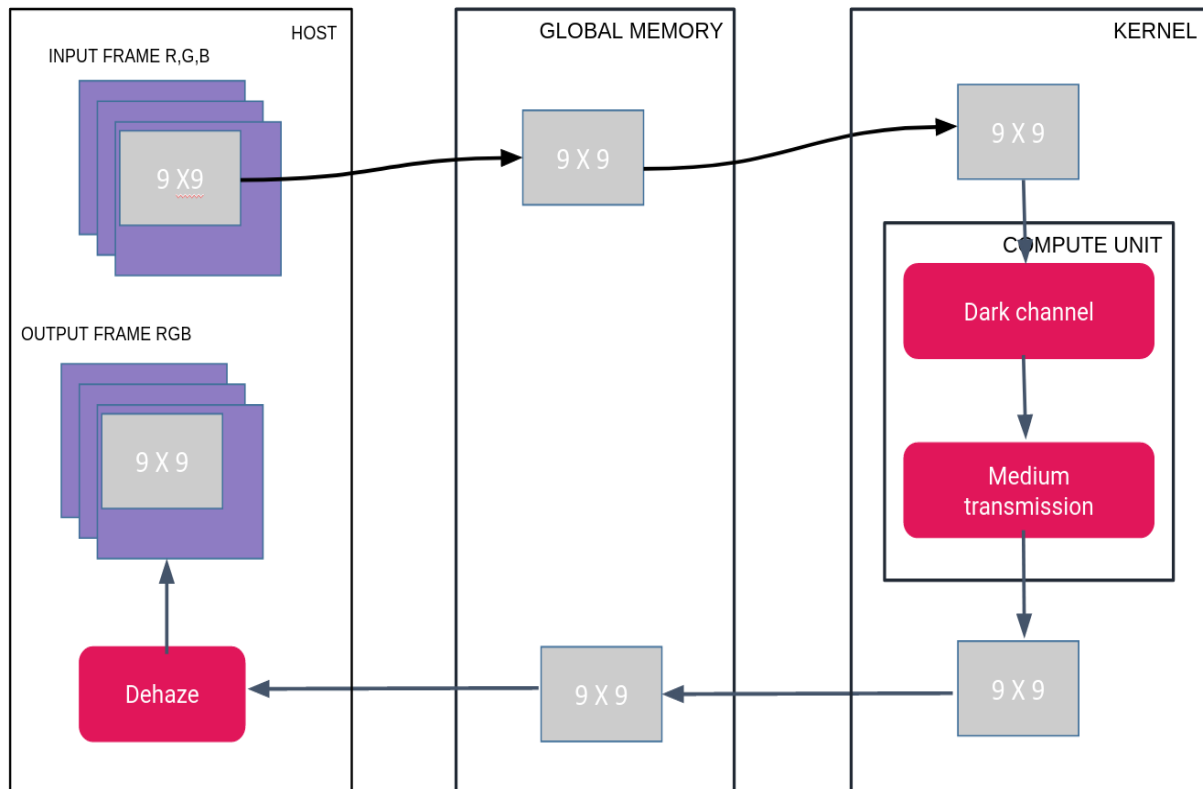


Fig 5.4.2 Host-Kernel Dataflow

Fig 5.4.2 Host-Kernel Dataflow. The image frames are collected on the Host, segmented and sent in small sets (Windows) of 9x9 size to the shared memory location. The internal components of the kernel are fetching the windows from the shared memory and execute the data following the DCP /Medium transmission on kernel (FPGA) and send back the result to the Host through the shared memory location, the host read the the frame from the shared memory and execute the dehaze function to recover the frame

5.5 Performance parameters

In order to evaluate the quality of the implemented DCP, some performance parameters will be used. In addition, we also have some video processing requirements such as FPS (frames per second), frame size, latency, memory size, that will impose restrictions on the kernel implementation.

The qualities of the images or frames from video are depending on the parameter below:

Mean square error (MSE):

MSE measures the average of the square errors, if $x(i,j)$ is the source images with M and N are the number of rows and columns of the source images,

And Y, the reconstructed by decoding the encoded version of $x(i,j)$

The mean square error is defined by the equation (6) bellow:

$$MSE = \frac{\sum_{i=1}^M \sum_{j=1}^N [x(i,j) - y(i,j)]^2}{M \times N} \quad (6)$$

The RMSE (Root Mean Square Error): is defined as the square root of MSE , the equation (7) below show the RMSE formula.

$$RMSE = \sqrt{\frac{1}{NM} \sum_{i=1}^M \sum_{j=1}^N [Y_{ij} - \tilde{y}_{ij}]^2} \quad (7)$$

Peak Signal to Noise Ratio (PSNR)

PSNR represents the measure of peak error, when we have a higher value of PSNR the error will be reduced. The equation (8) shows the PSNR expression.

$$PSNR = 10 \log_{10} \left[\frac{255}{MSE} \right]^2 \quad (8)$$

Latency

Latency measures the time taken by the FPGA to compute some data or images, normally FPGA should have a low latency compared to CPU.

Memory size

Evaluating the memory size used by the FPGA during the implementation helps us to provide information about different categories of FPGA which we can use for the deployment of our application program. The size of the memory used during the implementation describes how big our circuit is.

CHAPTER 6

RESULTS EVALUATION

We have introduced in this work an enhancement of night input frames accelerated on FPGA. The Dark Channel prior algorithm used permits to enhance the night input frame. The main advantage of this approach is the speed of video processing with very efficient FPGA resource utilization. Results of the enhancement of the night input frame using DCP filter are shown in the Figure below (Fig 6.1 -Fig 6.4). It shows that the contrast and the visibility of the output frame are improved. Our proposed method enhanced the quality of the night input frame and increased the video processing speed of night input frame using Alveo board U200 Xilinx as a platform for the implementation. Our proposed methods are very fast compared to the DCP algorithm implemented on CPU (Intel Core i7 Dell precision T1700, 3.0GHZ with 16Go RAM). The average running speed of our methods is shown in Tab.6.5 which is far away from the DCP algorithm implemented on CPU.

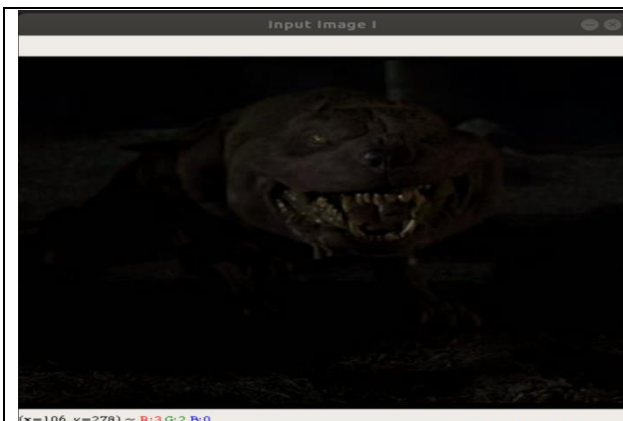


Fig 6.1 Input Frame

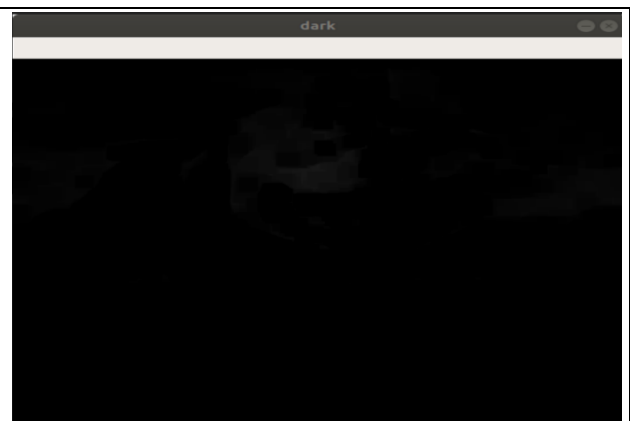


Fig 6.2 Dark input frame

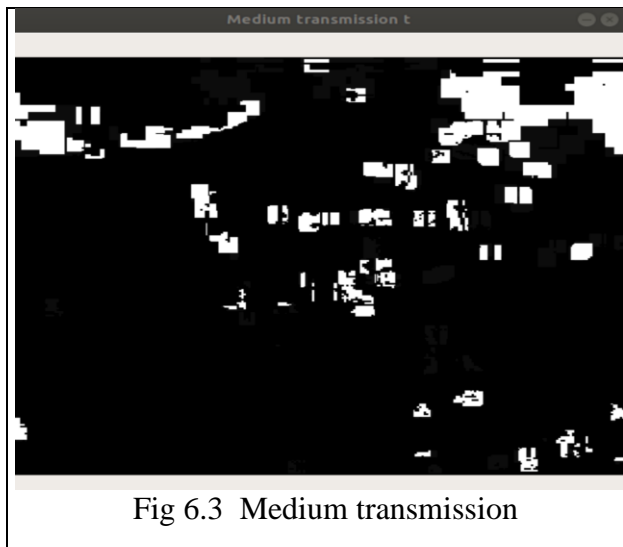


Fig 6.3 Medium transmission

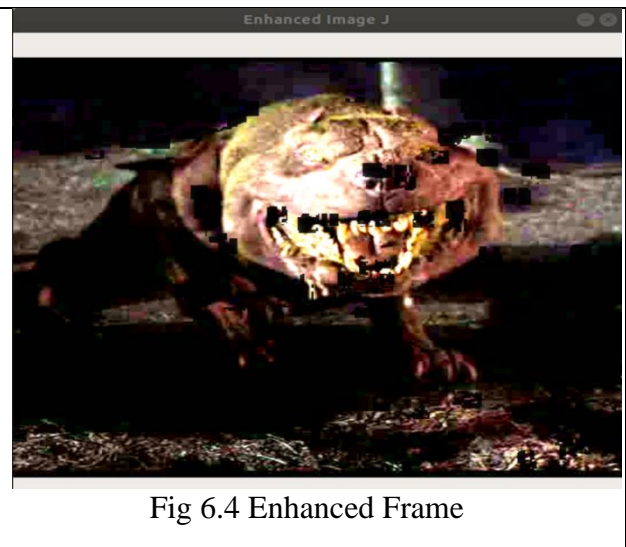


Fig 6.4 Enhanced Frame

Table 6.1 below represents the processing time of DCP implemented on software using core i7 -2.7GHZ and 16Go RAM with different frame resolutions, in our proposed method, we used 3 functions (Dark channel, Medium transmission, Dehaze function) to execute the data on CPU . Table 6.1 depicts that the processing time of dark channel prior and medium transmission function consume much time on software.

Table 6.1 Processing time of proposed algorithm on software

Frame size	Dark channel function (second)	Medium function transmission (second)	Dehaze function (second)
1600 x 900	15.5872 seconds	15.4683 seconds	0.144223 seconds
1600 x 900	5.22823 seconds	5.49135 seconds	0.085522 seconds.
1200 x 675	0.426047 seconds	0.439198 seconds	0.021492 seconds
320 x 180	0.044025 seconds	0.045346 seconds.	0.006188 seconds.

Table 6.2 below represents the summary of processing time of the DCP algorithm implemented on hardware with frame size 1600 x 900 using Alveo board u-200.

Table 6.5 Processing time of proposed algorithm on hardware

Frame size	Start time	End time	Duration
1600 x 900	18950.87817ms	45300.21113 ms	26349.33296 ms

Figure 6.5 -6.6 below represents the platform diagram of our proposed architecture and the application timeline of hardware emulation of proposed algorithm DCP

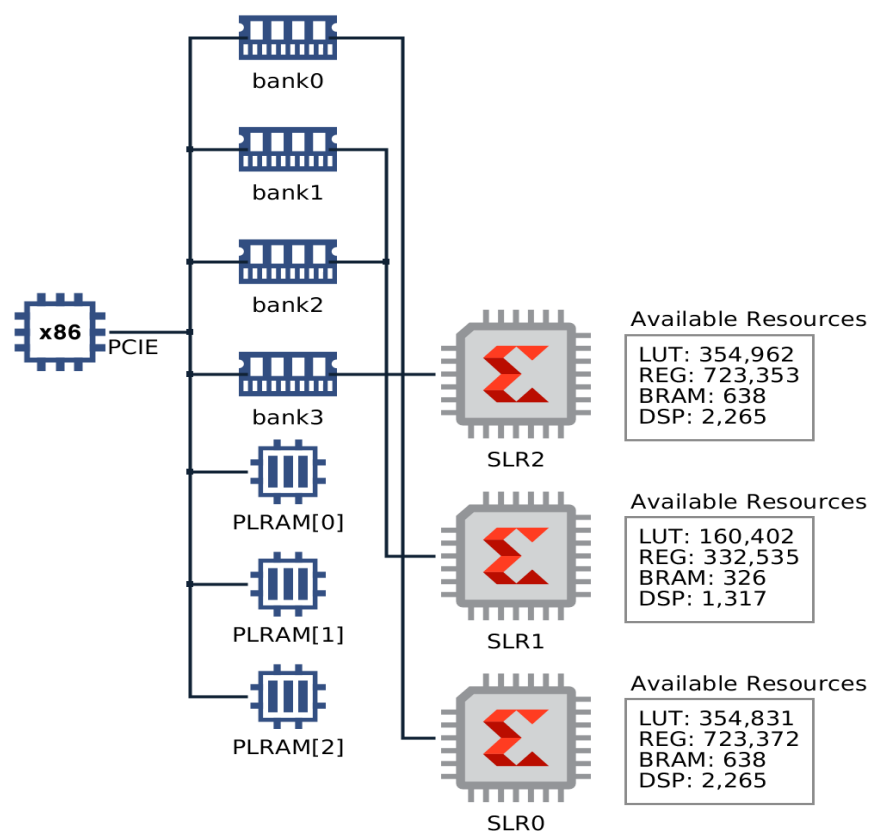


Fig 6.5 Platform diagram of proposed architecture of Host/Kernel

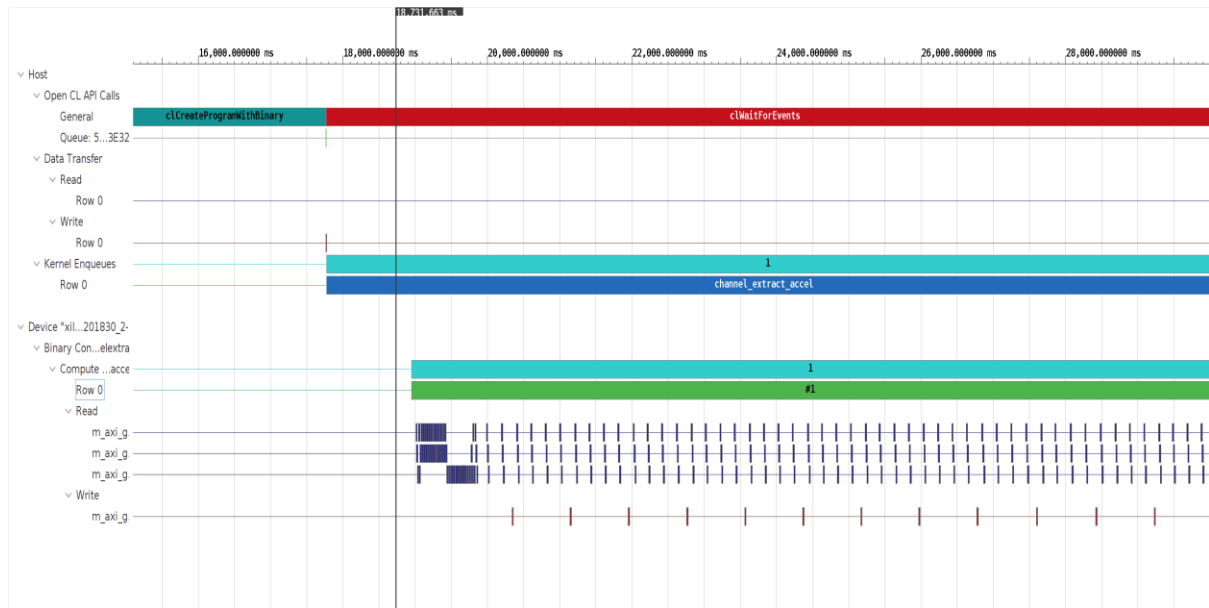


Fig 6.6 Application timeline hardware emulation (1600 x 900 frame size)

Furthermore, we made a comparison of processing time among software implementation and hardware implementation, as seen in Table 6.2 . This comparison demonstrates that hardware implementation using FPGA methods is the most time-efficient one. Besides, without complex functions, this method is more suitable to apply in limited hardware resource platforms.

Table 6.2 Processing time comparison of DCP /Medium transmission on CPU/FPGA

Frame size	CPU core i7	Alveo Board u200
1600 x 900	31.048 s	26.349 s

The result on the table (Table 6.4–6.6) below belongs to the Vivado HLS, which represents:

- The target clock frequency, clock uncertainty, and the estimate of the fastest achievable clock frequency (Table 6.4) .
- The latency of the DCP implemented on hardware (Table. 6.5)
- The resources (LUTs, FlipFlop, DSP48s) used to implement the design (Tab 6.6).The resources used on FPGA implementation is small,shown in Tab 6.2

Table 6.3 Details of the resources used on FPGA.

* Summary:						
Name	BRAM_18K	DSP48E	FF	LUT	URAM	
DSP	-	-	-	-	-	-
Expression	-	-	0	46	-	-
FIFO	0	-	183	1525	-	-
Instance	70	32	15770	47107	0	0
Memory	-	-	-	-	-	-
Multiplexer	-	-	-	72	-	-
Register	-	-	18	-	-	-
Total	70	32	15971	48750	0	0
Available SLR	1440	2280	788160	394080	320	320
Utilization SLR (%)	4	1	2	12	0	0
Available	4320	6840	2364480	1182240	960	960
Utilization (%)	1	0.4	0.6	4	0	0

Table 6.4 Timing summary of apc_cl

Clock	Target	Estimated	Uncertainty
apc_clk	3.33 ns	2.433ns	0.90ns

Table .6.5 Latency of DCP

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
8303056	8321548	27.674 ms	27.736 ms	8303044	8321533	dataflow

Table 6.6 FPGA Resource Requirement for our Method

Resource type	Resource used	Total available	Ratio
LUT	48750	1182240	4%
DSP	32	6840	0.4%

FF	15971	2364480	0.6%
URAM	0	960	0%
BRAM_18K	70	4320	1%

CHAPTER 7

CONCLUSION

In this project, I conclude about the progress and the states that in this project, I have studied and practiced the enhancement of night time using Dark Channel Prior algorithm on software part using core i7 (RAM 16 Go) and evaluated the speed of video processing of varieties quality of input video, the video processing speed of the nighttime video with 240,360,480p can be continuous using PC. Increasing the size and quality of the input video affects the speed of video processing in CPU core i7 (RAM 16 Go). Using hardware (Alveo board - FPGA) for the implementation of DCP algorithm is one of the best solutions to improve the video processing speed of the night input frame, the enhancement of night input frame can be continuous on hardware with small resources used in hardware, low latency, and fast processing. Here the atmospheric light of the input frame is executed on CPU using the python language and assigned on hardware as input argument. The experiment result showed that our proposed method can achieve real-time video processing with fast video processing compared to the DCP algorithm implemented on software.

Future work: Increasing the compute unit of the kernel on hardware to improve the parallelism design with minimization of power consumption used by the FPGA is still a challenging task.

6.1 BIBLIOGRAPHY

1. https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/cuu1526001449959.html
2. https://www.xilinx.com/html_docs/xilinx2019_2/vitis_doc/Chunk1353458556.html#ctb1559866511109
3. <https://www.flir.com/discover/what-is-infrared>
4. Kim, M., Park, D., Han, D. K., & Ko, H. (2014). "A novel framework for extremely low-light video enhancement". In Digest of Technical Papers-IEEE International Conference on Consumer Electronics. (pp.91-92).
5. Xuesong Jiang, Hongxun Yao, Shengping Zhang¹, Xiusheng Lu¹ and Wei Zeng, "Night video enhancement using improved Dark Channel Prior", IEEE, 2013, p553-557
6. Bhagya H.K Keshaveni N, "Video Enhancement using Histogram Equalization with JND Model", International Journal of Recent Technology and Engineering (IJRTE), 2 July 2019, p2506-2511
7. R. Peng, P. K. Varshney, H. Chen, and J. H. Michels, "Adaptive Algorithms for Digital Mammogram Enhancement", SPIE Medical Imaging Conference San Diego, CA, 2008, pp. 16-21
8. D. H. Choi, I. H. Jang, M. H. Kim, and N. C. Kim, "Color Image Enhancement using Single-Scale Retinex Based on an Improved Image Formation Model", EUSIPCO, 2008
9. Z. Rahman, D. J. Jobson, and G. A. Woodell, MultiScale Retinex for Color Image Enhancement, NASA Langley Research Center, IEEE, 1996
10. Shwartz, Namer, Schechner, "Blind haze separation," IEEE, 17-22 June 2006, pp. 1984-1991
11. Xuesong Jian, Shengping Zhang, Hong Yao, Xiusheng Lu, "Night video enhancement using improved dark channel prior," IEEE, 2013
12. X. Dong, G. Wang, Y. Pang, W. Li, J. Wen, W. Meng, and Y. Lu, "Fast efficient algorithm for enhancement of low lighting video," in Proceedings of International Conference on Multimedia and Expo. IEEE, 2011, pp. 1-6
13. Gengfei Li, Guiju Li and Guangliang Han, "Enhancement of Low Contrast Images Based on Effective Space Combined with Pixel Learning", MDP
14. K. He, J. Sun, and X. Tang, "Single image haze removal using dark channel prior," in Proceedings of Conference on Computer Vision and Pattern Recognition. IEEE, 2009, pp. 1956-1963.
15. M. Pedone and J. Heikkila, "Robust airlight estimation for haze removal from a single image," Proceedings Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2011, pp. 90-96.

16. Hussein,Mahdi, Nidhal El Abbadi, Hind, “Single Image Dehazing Through Improved Dark Channel Prior And Atmospheric Light Estimation”, Journal of Theoretical and Applied Information Technology,20017.
17. Tan RT, “Visibility in bad weather from a single image” ,IEEE; 2008. p. 1–8.
18. Fattal R, “Single image dehazing”,ACM Trans Graph TOG. 2008;27(3): 72.
19. Tarel J-P, Hautiere N. “Fast visibility restoration from a single color or gray level image”, IEEE; 2009. p. 2201–8. 34. Fattal R. Dehazing using color-lines. ACM Trans Graph TOG. 2014;34(1):13.
20. Fattal R,“ Dehazing using color-lines ”, ACM Trans Graph TOG. 2014;34(1):13.
21. Tang K, Yang J, Wang J, “ Investigating haze-relevant features in a learning framework for image dehazing”, In 2014. p. 2995–3000.
22. Zhu Q, Mai J, Shao L A, “ fast single image haze removal algorithm using color attenuation prior ”, IEEE Trans Image Process. 2015;24 (11) :3522–33.
23. Cai B, Xu X, Jia K, Qing C, Tao D, “Dehazenet: An end-to-end system for single image haze removal”, IEEE Trans Image Process. 2016;25(11):5187–98.
24. Berman D, Avidan S, “Non-local image dehazing”,In 2016. p. 1674–82.
25. www.pynq.io
26. <https://github.com/Xilinx/Vitis-Tutorials/tree/master/docs/vitis-getting-started>
27. www.xilinx.com/alveo
28. https://github.com/nambhine1/DCP_

Urkund Analysis Result

Analysed Document: Rakotojaona Andrianoelisoa Nambinina.pdf (D68354375)
Submitted: 4/16/2020 6:24:00 PM
Submitted By: prutha.pandya77614@paruluniversity.ac.in
Significance: 5 %

Sources included in the report:

<https://github.com/Xilinx/Vitis-Tutorials/tree/master/docs/getting-started-rtl-kernels>
https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/uqp1519743299633.html
<https://github.com/Xilinx/Vitis-Tutorials/tree/master/docs/vitis-getting-started>
[71c9f290-279d-47d6-b9e4-7c7dfc734d99](https://github.com/Xilinx/Vitis-Tutorials/blob/master/docs/Pathway3/BuildingAnApplication.md)
[a642f5ec-b613-4532-b122-e3c58e48cca3](https://github.com/Xilinx/Vitis-Tutorials/blob/master/docs/Pathway3/BuildingAnApplication.md)
[https://vtechworks.lib.vt.edu/bitstream/handle/10919/71389/Shi_Z_T_2016.pdf?](https://vtechworks.lib.vt.edu/bitstream/handle/10919/71389/Shi_Z_T_2016.pdf?sequence=1&isAllowed=y)
[sequence=1&isAllowed=y](https://www.mdpi.com/2227-7080/7/1/4/pdf)
<https://www.mdpi.com/2227-7080/7/1/4/pdf>