ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR

M Tech Dissertation
Submitted in
partial fulfillment of the requirements
for the degree of

MASTERS OF TECHNOLOGY

in

VLSI Design And Embedded System

by

Rakotojaona Andrianoelisoa Nambinina

180305212004

Under the supervision of

Mr.Haresh Suthar
Mr. Yogesh D. Parmar
Prof. C.Valderrama

April 2020

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

PARUL INSTITUTE OF TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

PARUL UNIVERSITY

P.O. Limda – 391 760, GUJARAT, INDIA

CERTIFICATE

This is to certify that the work contained in this dissertation thesis entitled **ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR** submitted by **RAKOTOJAONA ANDRIANOELISOA NAMBININA Enrollment No** 180305212004 studying at **Parul Institute of Technology** for the Phase - II Dissertation (M. Tech- VLSI DESIGN AND EMBEDDED SYSTEM) is absolutely based on his own work carried out under my supervision and that this work/thesis has not been submitted elsewhere for any degree/diploma. To our satisfaction, this work is approved for the Phase - I (M. Tech- VLSI DESIGN AND EMBEDDED SYSTEM) Internal/External exam

The extent of plagiarism does not exceed the permissible limit laid down by the University

Mr. Yogesh Parmar	**Prof. C. Valderrama**
Faculty Supervisor	External Supervisor

Date: _________________

Mr. Haresh A. Suthar		**Dr. Bhavesh Mewada**
Faculty Supervisor &HOD	PG Co-ordinator	Principal, PIT

THESIS APPROVAL CERTIFICATE

This is to certify that research work embodied in this dissertation thesis entitled **ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR** carried out by Mr **Rakotojaona Andrianoelisoa Nambinina** (Enrollement No. 180305212004) at Parul Institute of Technology is approved for the degree of M.Tech with specialization of VLSI Design And Embedded System by Parul University.

Date:
Place:

External Examiners' Sign and Name:

1) ____

2) ____

PAPER PUBLICATION CERTIFICATE

This is to certify that research work embodied in this dissertation thesis entitled **ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR** carried out by Mr. **Rakotojaona Andrianoelisoa Nambinina** (Enrollement No. 180305212004) at Parul Institute of Technology for partial fulfillment of M.Tech degree to be awarded by Parul University, has published article entitled ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR for publication by the Blue Eyes Intelligence Engineering & Sciences Publication, International Journal of Engineering and Advanced Technology, submitted by the author: 4 th -march-2020

Date:
Place:

Rakotojaona Nambinina	Mr. Haresh Suthar	Mr. Yogesh Parmar	Prof.C .Valderrama
	Faculty Supervisor	Faculty Supervisor	External Supervisor

ACKNOWLEDGMENTS

This thesis is end of my long journey in achievement of degree of M.Tech (VLSI Design And Embedded System).There are some people who made this journey easier with words of encouragement and more intellectually satisfying by offering different place to look at and expand my theories and idea. First special thank goes to my helpful project guide Mr.Haresh Suthar ,HOD ,Electronics & Communication Department,PIT, Limda and Mr.Yogesh Parmar, Coordinator ,Electronics & Communication Department,PIT, Limda and Prof. Carlos Alberto VALDERRAMA SAKUYAMA Microelectronics & Electronics Unit Université de Mons. This thesis is end of my long journey in achievement of degree of M.Tech (VLSI Design And Embedded System).There are some people who made this journey easier with words of encouragement and more intellectually satisfying by offering different place to look at and expand my theories and idea. Not forget, great appreciation goes to Mr. Yogesh Parmar, Assistant Professor, Electronics & Communication Department and Mr. Haresh Suthar, Head of Department, Electronics & Communication for help me from time to time during my dissertation.

RAKOTOJAONA A NAMBININA
(180305212004)

PARUL UNIVERSITY, FACULTY OF ENGINEERING AND TECHNOLOGY
Electronics and Communication Department
M.Tech (Branch: VLSI Design and Embedded System)

ENHANCEMENT OF NIGHT TIME VIDEO USING DARK CHANNEL PRIOR IP ACCELERATOR

Submitted By

Rakotojaona Andrianoelisoa Nambinina

180305212004

Supervised By

Mr. Haresh Suthar

Mr. Yogesh D. Parmar

Prof. C.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 results 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 Background Overview of enhancement of night time video IP accelerator	1
1.2 Image processing	3
1.3 Field Programmable Gate Array	4
Chapter 2: STATE OF ART	5
2.1 Literature survey of the algorithm used	5
2.2 Discussion of the method used for enhancement of night time	7
Chapter 3: DCP ALGORITHM	10
3.1 Mathematical model of DCP algorithm	10
3.2 Chart flow of DCP Algorithm	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: HARDWARE SOFTWARE IMPLEMENTATION OF DCP	22
5.1 System architecture	22
5.2 Component used for the enhancement of night time on FPGA	22

5.3 Global architecture of proposed method 24

5.4 Design flow of proposed method 25

5.5 Performance parameters 27

Chapter 6: RESULT AND DISCUSSION 29

Chapter 7: CONCLUSION 35

Chapter 8: DESCRIPTION OF ANNEX 38

Annex 1 Work Plan 39

Annex 2 Integration of RTL kernel in application program 40

Annex 3 Pynq Board Descriptions 50

Annex 4 Building the host and the kernel 53

Annex 5 Publications 59

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
3.2.1	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	*Host and Kernel*	*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
5.1	Performance parameter	30
6.1	Summary of resources used by the FPGA	31
5.2	Timing summary of apc_clk	32
5.3	Latency of DCP	32
5.4	FIFO resources used during hardware emulation	33
5.5	Expression resources used during hardware emulation	34
5.6	Multiplexer resources used during hardware emulation	34
5.7	Register resources used during hardware emulation	35
5.8	Summary of utilization estimates	35

CHAPTER 1
INTRODUCTION

1.1 Background 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 project due to it is simplicity and it 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 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 project 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	High	Very High	Highest
Power consumption	High	Very High	Very Low	Low
Flexibility	Highest	Medium	Very High	Lowest

Fig.4.1.1 depicts the architecture of the Vitis development environment. 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 tensor flow 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. The 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, we will discuss the state of art done to find the gap of recent approaches. In chapter 3, we will discuss 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 will present 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 will present the implementation of the DCP algorithm into hardware and the data flow between the host and the kernel. Chapter 6 will present the experiment result and the discussion of the result. Chapter 7 will conclude the thesis and describe future work. Chapter 8 will describe 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 from reconnaissance flights and spaces. While in this project we focused only on the enhancement of night input frame from the input cameras. 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
- night time enhancement
- non destructive evaluation
- graphic arts
- printing in industry (and many more)

The schematic diagram of enhancement of night time is shown in Figure 1.2.1, The input frame is enhanced by the DCP algorithm (using FPGA as a platform) and produced an enhanced output frame.

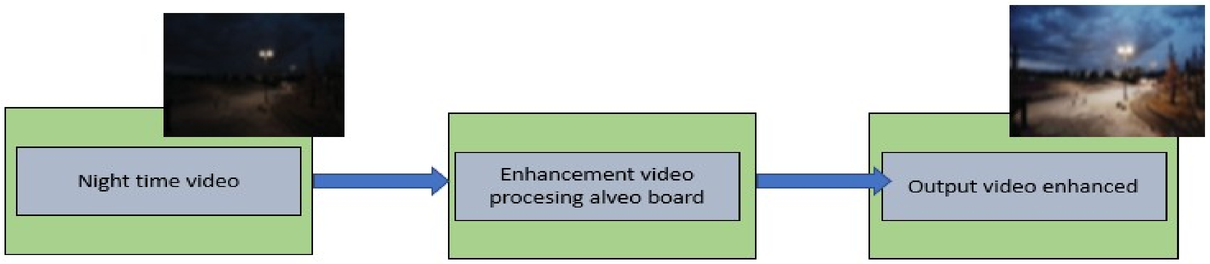


Fig 1.2.1 Enhancement of night time

1.3 Field Programmable Gate Array (FPGA)

FPGA is a reconfigurable integrated circuit mostly used for the acceleration of an application program. Each FPGA has three main parts (shown in figure 1.3.1):

- Configurable Logic Block
- Input-Output Block
- Programmable Interconnect

CLB provides physical support for the program downloaded on FPGA, this block can be reconfigurable as the user wish. Another part is the Input-Output Block (IOB), this block is responsible for the communication between the external and the FPGA which provides input and output for FPGA. The last part is the Programmable Interconnect (PI), as its name implies this block used to interconnect every logic block inside of the FPGA, and this block also can be reconfigured as the user wish. 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.

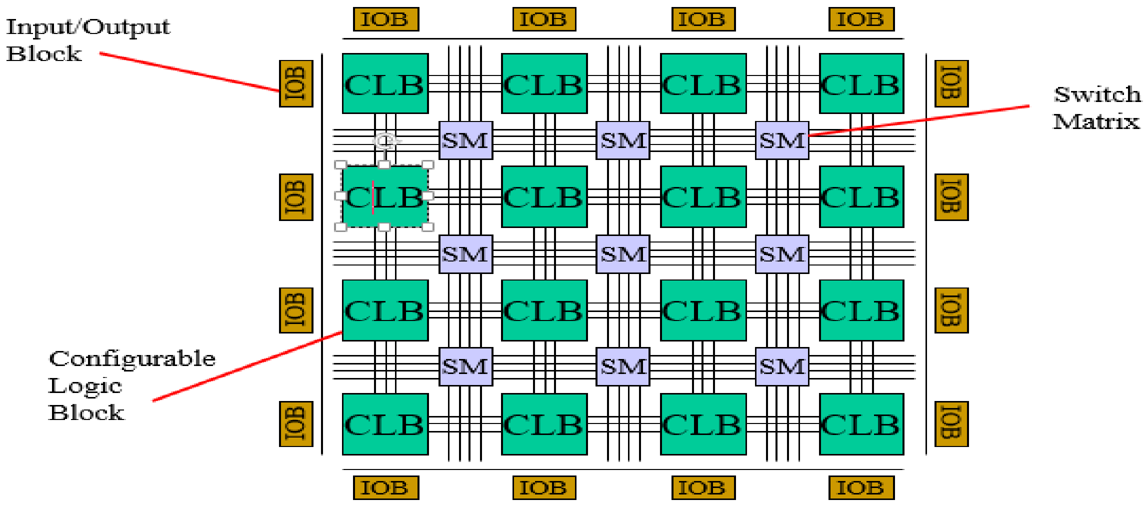


Fig 1.3.1 Architecture of FPGA [15]

1.4 Motivation

The improvement in technology of image processing at night makes the hardware and software engineer focus on the development of an application program that can be used to enhance a night-time input frame. This reason increases the number of applications that can be used at night. A digital video is a sequence of images with a constant time interval. So there is more information present in the video. After studying the literature, it is seen that enhancement of the night input frame with real-time output video is really challenging task for a computer vision application. Video processing consumes more time for software implementation such as CPU, due to the huge number of data present in the video. Using FPGA for implementation is one of the best solutions to accelerate the speed of video processing. Enhancement of nighttime video is currently of immense interest due to its implication in video surveillance, security, robotics system and many more.

CHAPTER 2

STATE OF ART

2.1 Literature survey of the algorithm used for the enhancement of night time

The table 2.1.1 below describes the pros & cons of the available methods for the enhancement of night time.

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	High image visibility

				input image	
[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 used 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].

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 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. all 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. Ultimately, the remaining unwanted factor which is noise can be removed using Non-local means (NLM) denoising. This technique exploits color filter array (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.

CHAPTER 3

DCP ALGORITHM

3.1 Mathematical model of DCP algorithm

In this project, we use the algorithm of DCP to enhance the night input frame. The main reason for using the DCP algorithm to enhance the night input frame is described in the state of art (Chapter 2). DCP algorithm is an algorithm used to enhance a night input frame based on the minimum pixel value of RGB. As Xuang et al. inversion of night input frames are quite similar to foggy days frames[10]. We use this hypothesis of Xuang et al. To solve our problem of the enhancement of night time by inverting the night input frame to get a foggy frame.

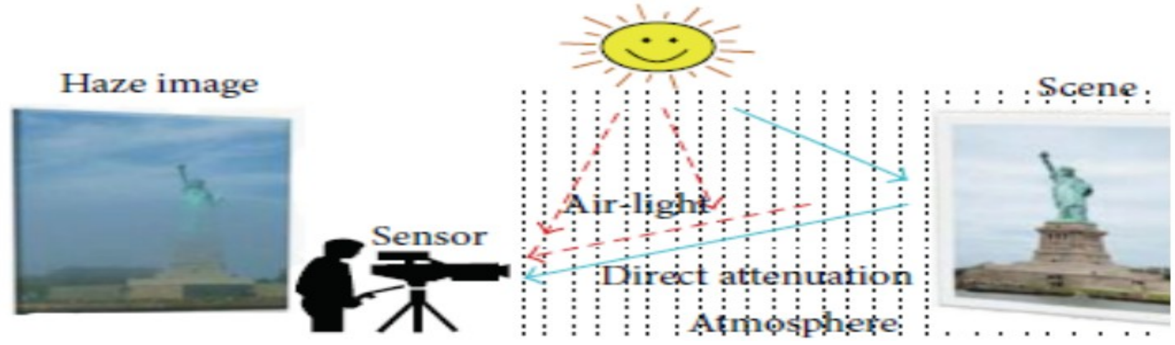


Fig .3.1.1 Atmospheric scattering [16]

Fig 3.1.1 saw that the input image captured by the sensor is not clear due to the existence of particles in the aerosol which make 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 called this technique as airlight.

In our case, the night input frames are inverted here to obtain a foggy input image so we can apply the DCP using the inversion of the night input frame.

In this project, we used I (x) as a night input frame (RGB), and Iinv(x) is the inversion of the night input frame(RGB). The data flow of the algorithm is described below:

- Inverse the input frame
- Determine the dark channel prior
- Estimating the atmospheric lightest
- Estimate the transmission value
- Recover the dehazed image
- Inverse the dehazed recover image.

To start with, we used the equation of McCartney to recover the enhanced output frame or original frame:

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

Where,

Jinv(x): Scene radiance (recover image) at x,

A:Global atmospheric light,

t(x): medium transmission at x,

x: coordinate of the image x = (i,j)

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

Using equation (1), we can express the Jinv(x):

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

With:

$$t(x) = 1 - \omega I^{c}{}_{dark}(x) \qquad (3)$$

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

Where:

ω: control parameter

$I^{c}{}_{dark}$: indicates a dark channel map (DCM) of inverted frames. DCM for each color channel is derived from the equation number (3), Ac for color, channel c is defined as the average of the lightest pixels per frame for the first 5 frames.

Generally, all of the pixels in an object should have the same depth. However, some pixels in an object may have different DCM values. If we use equation (3) and (4) directly, some detail can disappear. To solve this problem, we use equation (5) [10]

$I^{c}{}_{dark}(x) = \mu I^{c}{}_{dark}(x) \quad if\ \mu I^{c}{}_{dark}(x) < I^{c}{}_{dark}(x)$

$I^{c}{}_{dark}(x) = I_{smooth}(x)$ others (5)

Where

I smooth (x): smoothed result of *I dark (x)* by median filter of *J inv (x)*

μ: Constant parameter that controls the strength of constraints

Using the equations (3), (4), (5) in (2), we can get the result of *Jinv (x)*. Finally inversing $J_{inv}(x)$,to get the enhancement of dark input frames [10] [11].

3.2 Chart flow of DCP Algorithm

In our proposed method the input frames are inverted using bitwise not function (Vitis vision) to invert the night input frame to foggy frame. The inverted frame is used as input for the dark channel function, the output of the dark channel function is used as input for the medium transmission function and the output of the medium transmission is used as input of the recovery frame and the output of the recovery frame is inverted to obtain the enhanced night input frame. Figure 3.2.1 below shows the flow of our proposed algorithm.

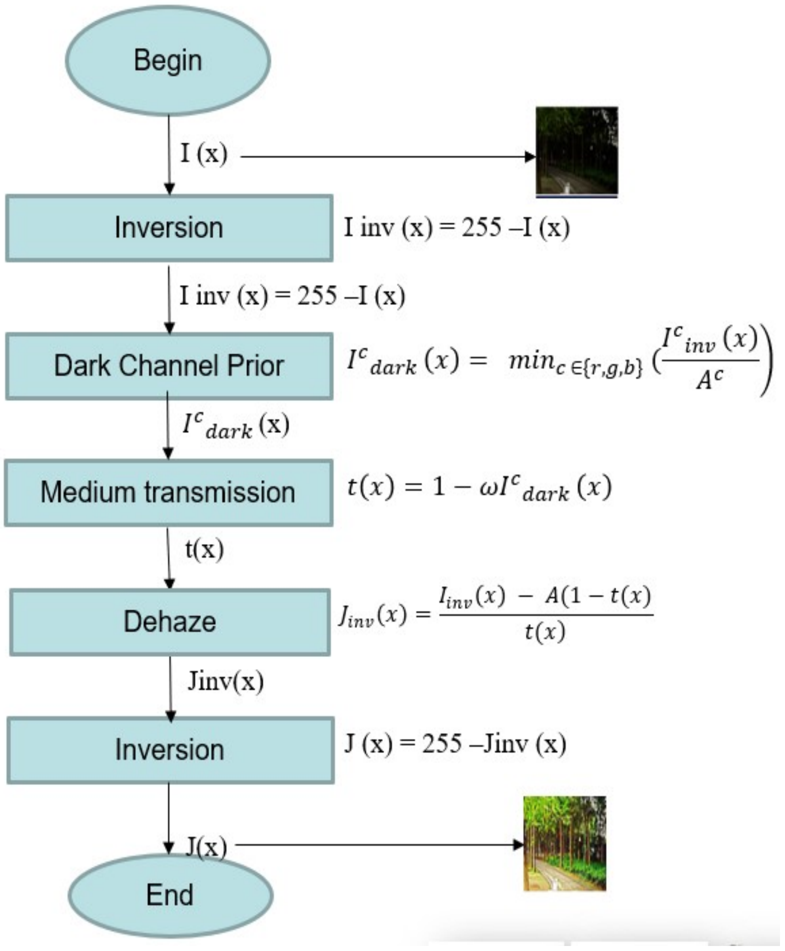


Fig 3.2.1 chart flow of proposed algorithm

CHAPTER 4

HARDWARE, TOOLS AND LANGUAGES

4.1 Vitis unified platform

Vitis software is a new tool of Xilinx that allows the user to build an application program on software running on hardware using Alveo board or ultra-scale FPGA, In this project, Alveo board is selected for the implementation.

Vitis accelerated application consists of two components: the host which contains the software application program and the FPGA which contains the accelerator kernel.The Xilinx runtime is known as XRT (shown in Fig 4.1.2) is a driver manage the communication between the software system and also the hardware. In Vitis software the application program is split into host and kernel (as shown in Figure 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, 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.

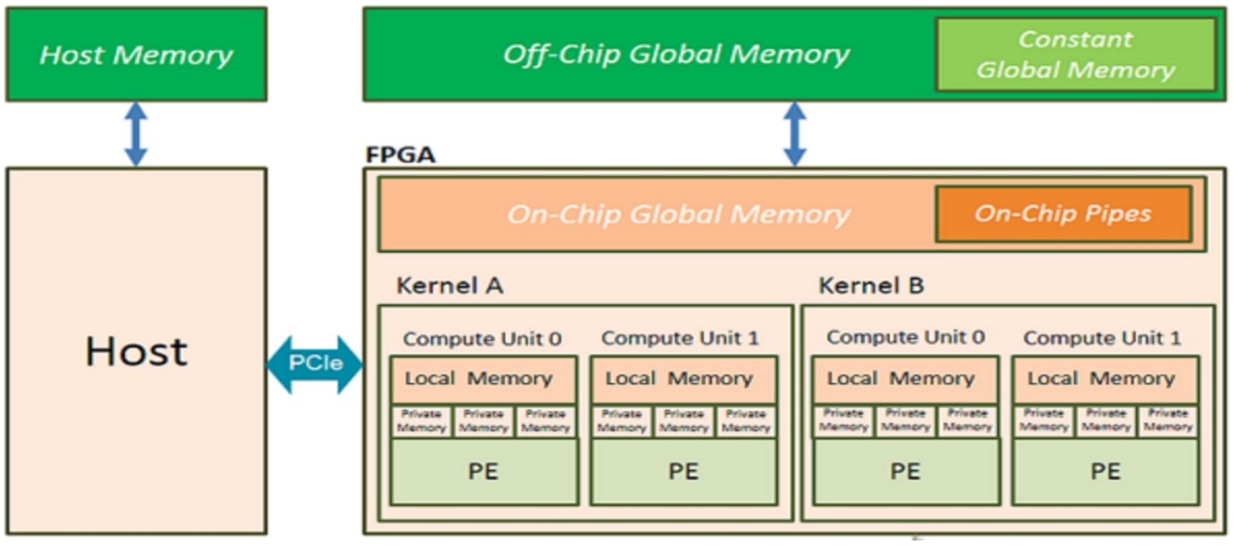


Fig 4.1.1 Global architecture of host and kernel

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 of FPGA such as Zynq, 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.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 driver, between the software part and the hardware which allow the communication between host and kernel (Alveo board)

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

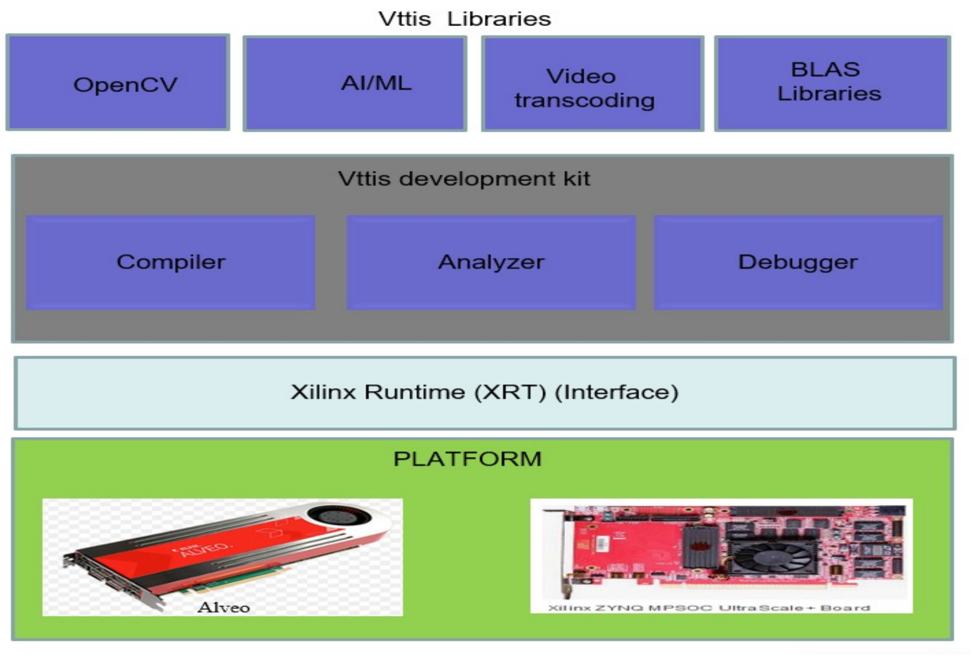


Fig.4.1.2 Architecture of Vitis unified software

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). We can not implement a high-level program on hardware, we need a software tool that can be compiled and linked the host code and kernel function into an executable file and bit-stream file. Alveo board support Vitis tool software which can be used in our development and deployment of our application program. So we have to install this tool on our laptop before we can start the development of an application program. Vitis unified 2019.2 is the version used here and Ubuntu OS is used as an 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 a library used in this project to develop the kernel function. Vitis vision is a pre-existing library developed by the Xilinx developer. This library helps the user to build an application program by optimizing the sources 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 compare to OpenCV.

The syntax used in Vitis vision function is described below:

- All argument 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 it into a xf::cv::Mat).
- The function 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 :
  - ◆ 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 know as an Open computing Language is a framework that's used to write an application program on different platforms such as CPU, GPU, FPGA (Field Programmable Gate Arrays) and Digital signal processor (DCP) and other hardware accelerators or other processors. OpenCL is probably easiest to learn and interesting programming language with widespread use.
- In this project, we use OpenCL language to develop the host code. OpenCL is a high-level language that can be used to target the FPGA platform by setting up commands such as the creation of the context, setting up of the kernel, write the data into share memory, execute the binary file on FPGA and many more. The figure below represents the structure of the OpenCL language.

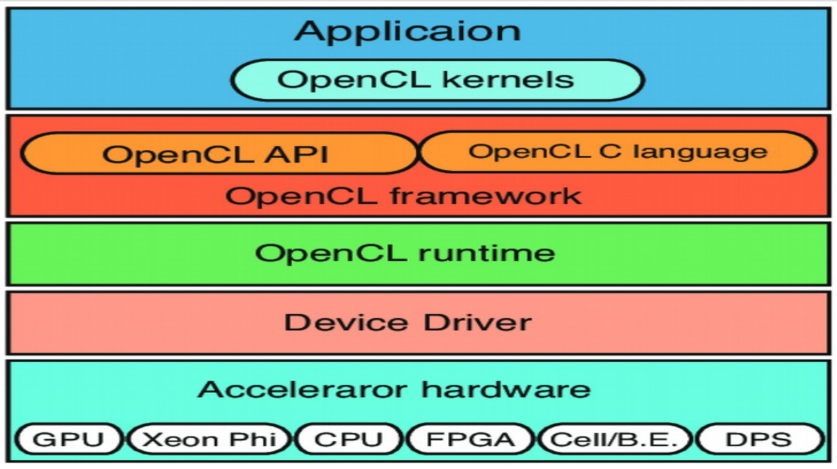


Fig4.4.1 Structure of OpenCL

- In this project, we use C++ in the host code as well in the kernel function to build an application program. This language is fast compare 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 Strostrup in 1979 at the laboratory of Labs in Hill, New Jersey , it was an improvement of 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 of 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 night input frame to be accelerated on 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 sources code provides 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 for the implementation. So we have interconnected the board with the PC and install some required components used by the FPGA (Driver, deployment package).

The installation of the Alveo card on motherboard is 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 x 16 slots, 255W power supply, and minimum RAM of 64 GB, Hard disk should be minimum of 100 GB and internet connection

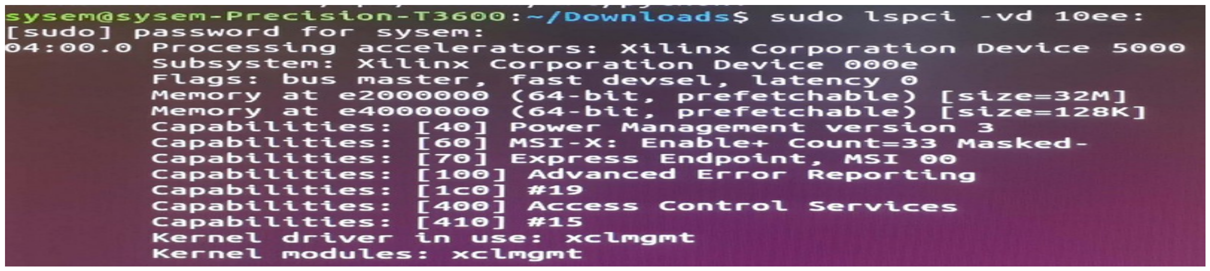
- Now we can start the procedure of installation by Turning off the power supply of the host (Desktop)
- Then open the central unit by removing the casing.
- Interconnect the U200 card on the motherboard through PCIe 16 slot and plug the power supply on the Alveo
- verify if all of the connection is plugged well
- And then turn on the desktop
- After installing the card on 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 correctly well, we should have the same result as this Fig 4.5.1.



```
sysem@sysem-Precision-T3600:~/Downloads$ sudo lspci -vd 10ee:
[sudo] password for sysem:
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 4.5.2 Result of lspci command

- Now we have confirmed that the card is correctly installed, So we can start installing some driver (XRT) which used to interconnect our host with the hardware
- To do that we need to download the Xilinx run time package 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/deployement/u200.deb

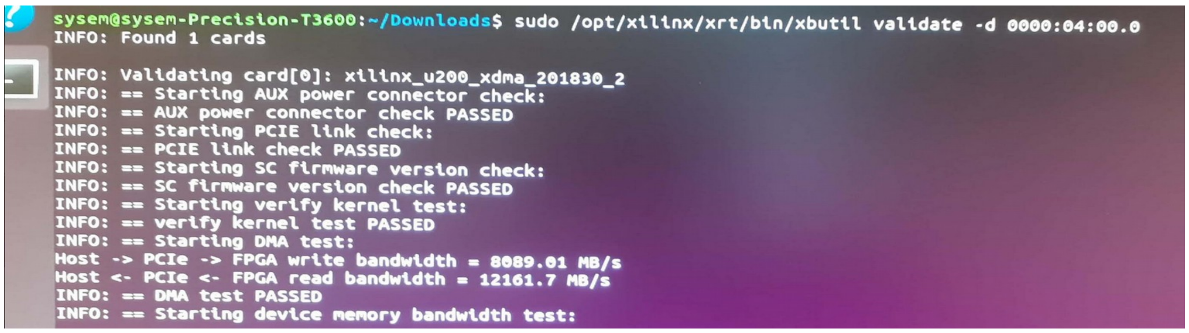
- After installation of the driver and the deployment, we have to flash manually the kernel, to test the board, now we are ready to test the board by configuring a test kernel, to do that we used the flash procedure (write the first kernel into FPGA memory) :

  >$ 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 function correctly, we should have the same figure (Fig 4.5.2) in our screen which shows that the verification of the kernel is passed.



```
sysem@sysem-Precision-T3600:~/Downloads$ sudo /opt/xilinx/xrt/bin/xbutil 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 4.5.3 Card found after validate command

CHAPTER 5

HARDWARE SOFTWARE IMPLEMENTATION OF DCP ALGORITHM

5.1 System architecture

In this project we are going to develop an application program of enhancement of nighttime video and accelerate it on Alveo card (FPGA), our proposed system has three main part: the host part, the kernel part and the global memory (known as shared memory) which is located on FPGA. The host and the kernel are interconnected through PCIe. The figure below shows the system architecture of the Host/Kernel/Global memory.

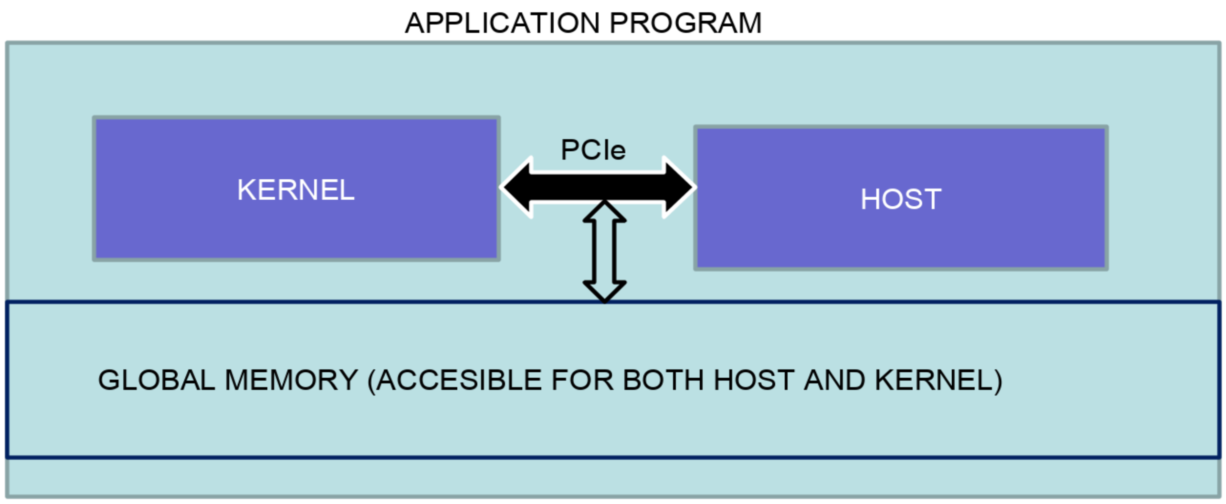


Fig 5.1.1 Host and Kernel

5.2 Component used for the enhancement of night time on FPGA

To implement the proposed algorithm (DCP algorithm) on the Alveo board (FPGA), we required some component as below :

- FPGA: Alveo u200 card is the platform used in this implementation with 182240 LUT , 2364480 FF , BRAM 6840 (18K)

- Host: Dell Core i7 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).

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

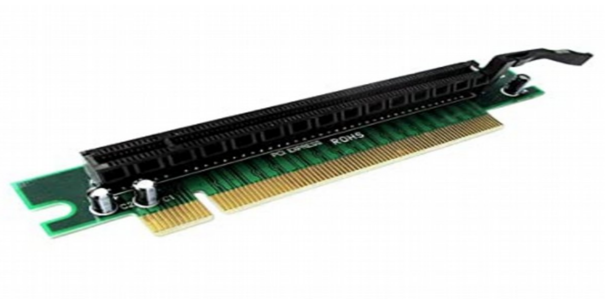
Fig 5.2.1 Alveo board	Fig 5.2.2 Host
 Fig 5.2.3 PCIe	Fig 5.2.4 Power supply

Figure 5.2.5 show the picture of the FPGA interconnected with the Host through PCIe

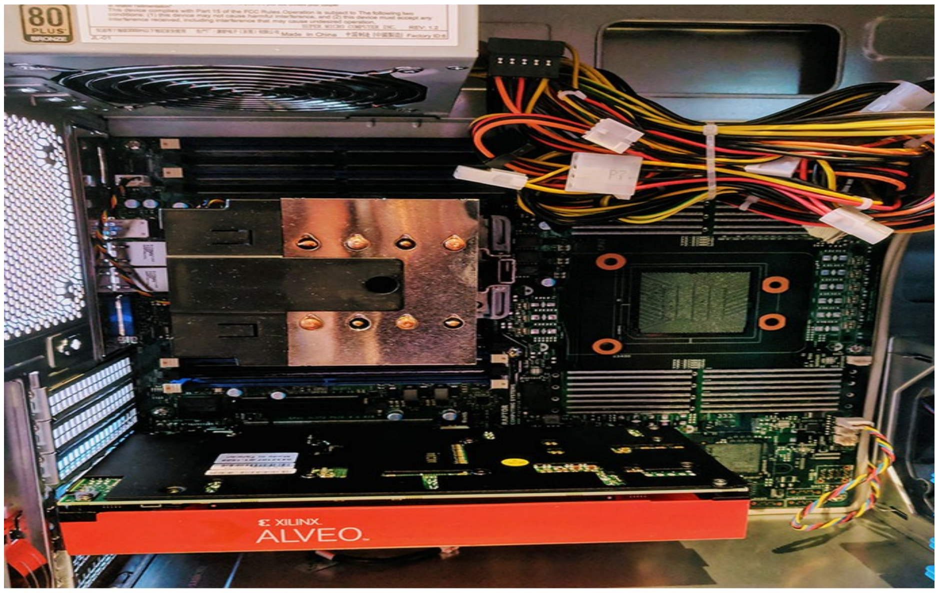


Fig 5.2.5 FPGA interconnected with Host

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 describes the data flow of the input frame in our proposed architecture.

- The host application writes the input data (input frame with atmospheric light) into a global memory of the attached device through the PCIe interface
- The host application setup the kernel
- The host application set up the kernel to execute the DCP algorithm following the 3 blocks in the pipeline (1. dark channel of the input frame, 2. medium transmission, 3. recover frame)
- The kernel writes the enhancement frame back after performing the DCP based algorithm into global memory and acknowledges the host that it has completed the execution
- The host application read the data from global memory and write it into his local memory.
- The host read the enhanced frame

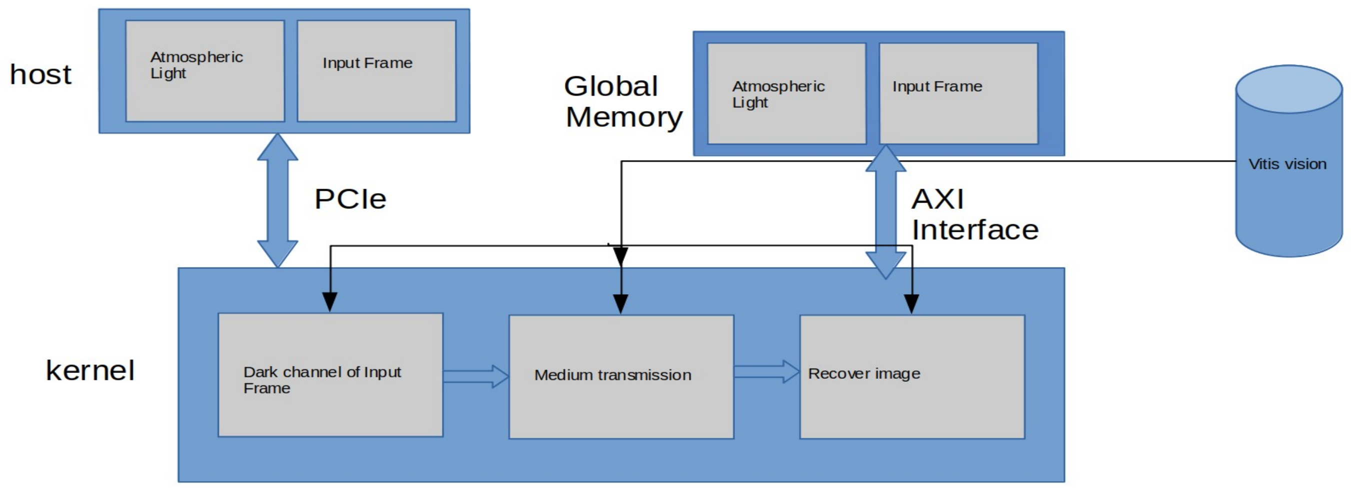


Fig 5.3.1Global architecture of proposed method

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 library)
- Tools (Vitis Xilinx)
- Platform (FPGA)

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 into hardware, as seen in figure 5.4.1, the host code and the kernel 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)

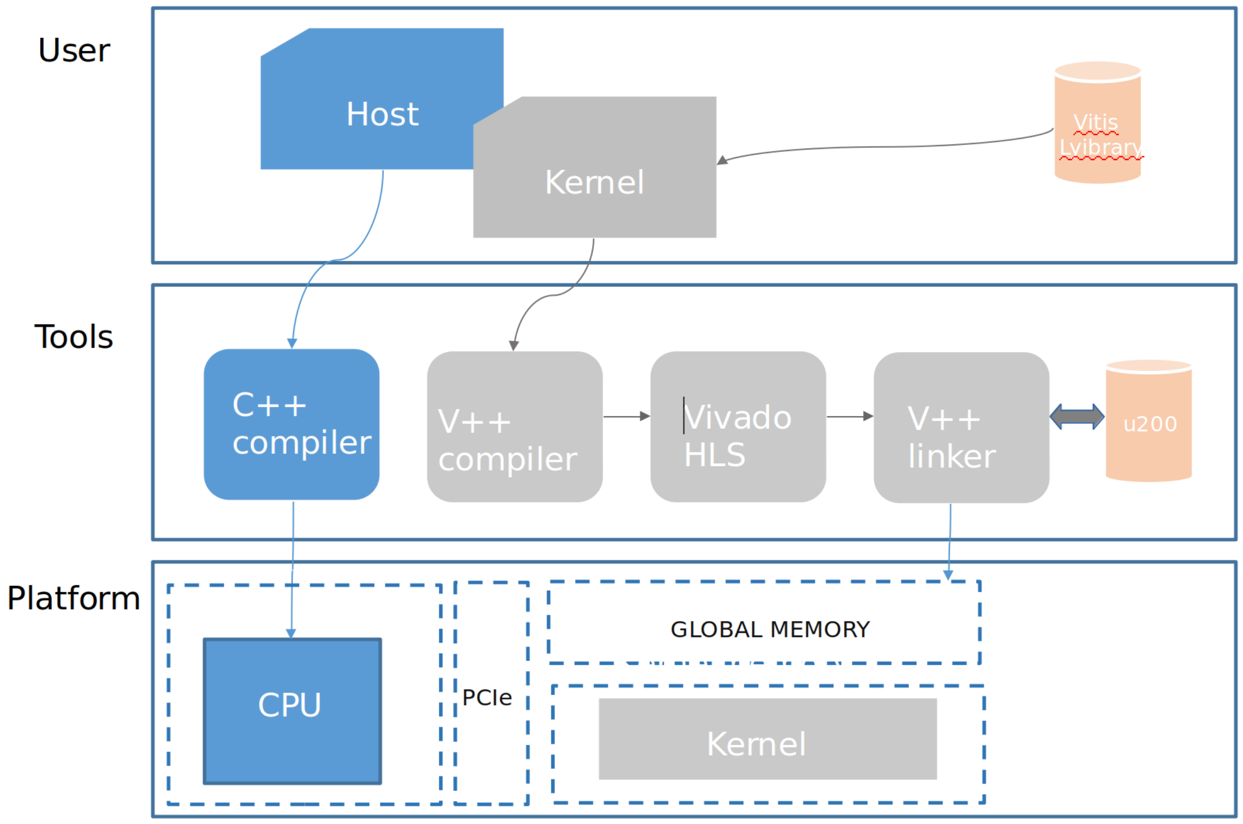


Fig 5.4.1 Design flow of host /Kernel

The figure below (Fig 5.4.2) represents the flow of the data between the host and kernel in our proposed method, we used pipelines flow in the kernel to execute the DCP algorithm on hardware, The host writes the input frame into share memory and the kernel read the data from share memory through AXI interface and then execute the data :

- Inversion of frame
- dark channel prior to the invert input frame
- medium transmission
- Dehaze or recover the invert input frame
- Inversion of the recovered frame

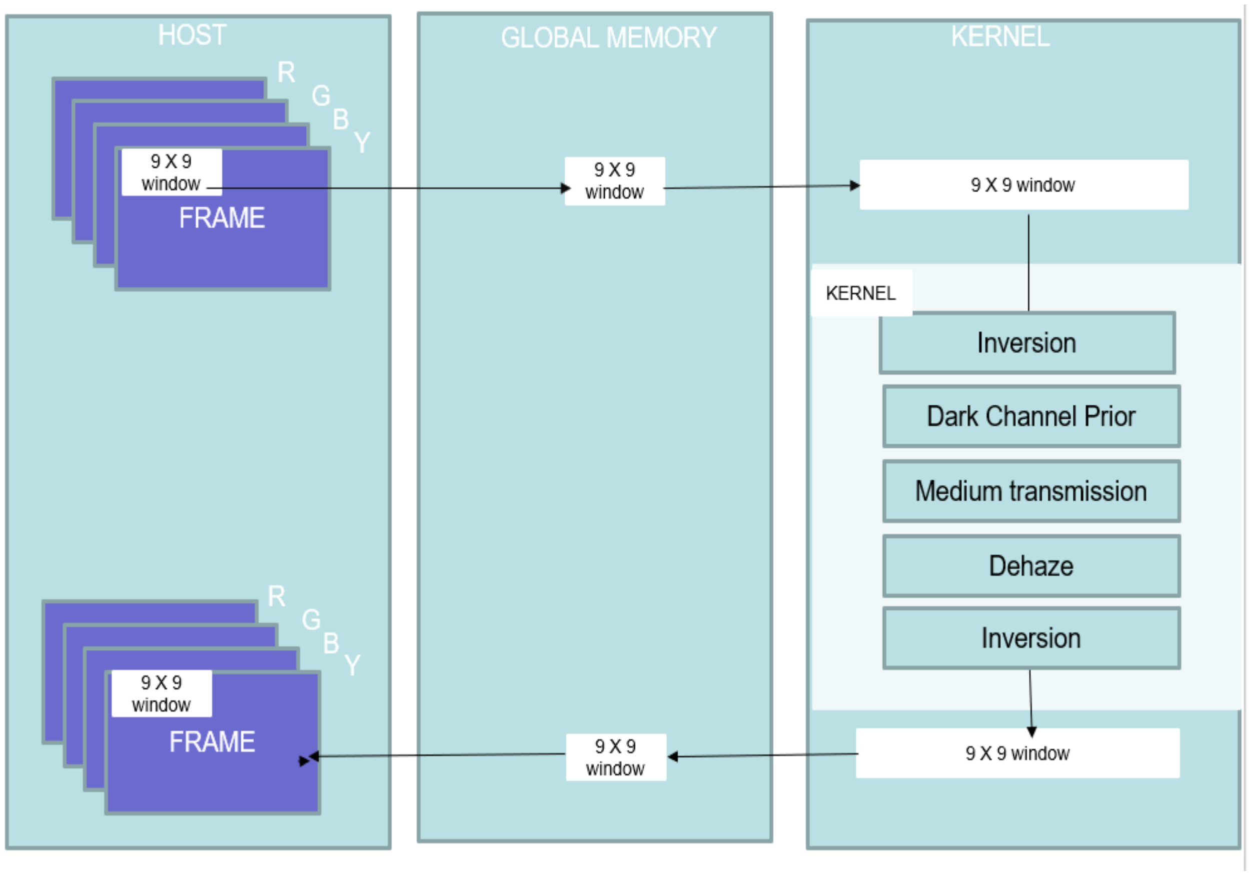


Fig 5.4.2 Host-Kernel Dataflow

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 kernel implements the DCP algorithm. The internal components of the kernel are fetching the windows from the shared memory and execute the data following the DCP algorithm and sent back the result to the Host through the shared memory location.

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 bellow

- **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:

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

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

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

- **Peak Signal to Noise Ratio (PSNR)**

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

$$PSNR = 10\ log_{10}[\frac{255}{MSE}]^2 \qquad \textbf{\textit{(8)}}$$

latency

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

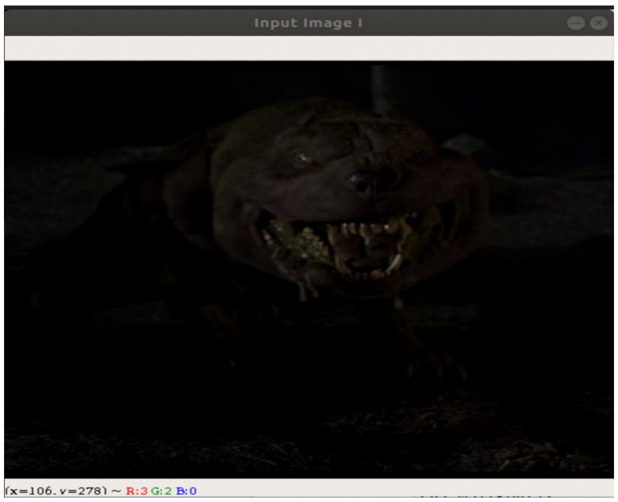
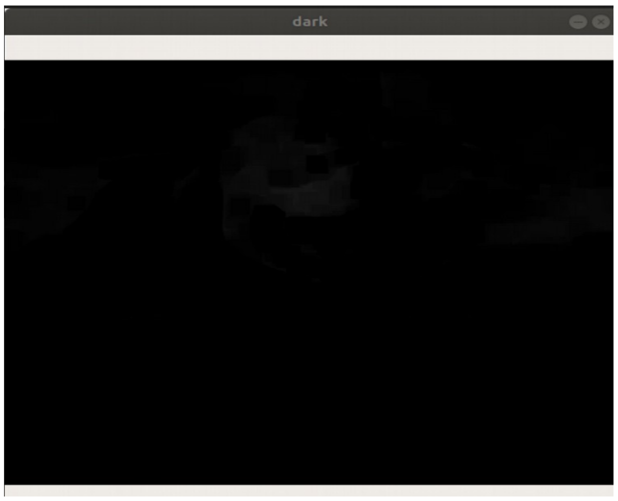
Memory size

Evaluating the memory size used by the FPGA during the implementation helps us to provides 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 is our circuit.

CHAPTER 6

RESULT AND DISCUSSION

We have introduced in this work an enhancement of night input frame accelerated on FPGA using the Alveo board. The Dark Channel prior algorithm used permits to enhanced 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 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 by using Alveo board U200 Xilinx as a platform for the implementation. Our proposed methods are faster than DCP implemented on CPU, the average running speed of our methods is 17 times faster than DCP implemented on CPU (Intel Core i7 Dell precision T1700, 3.0GHZ with 16Go RAM). The implementation of the DCP algorithm on software can achieve a real-time video speed with 37 frames per second (320p video) while increasing the quality of night input frames makes the output video slow. Our proposed method can solve this problem with the real-time output video.

 Fig 6.1 Input Frame	 Fig 6.2 Dark input frame

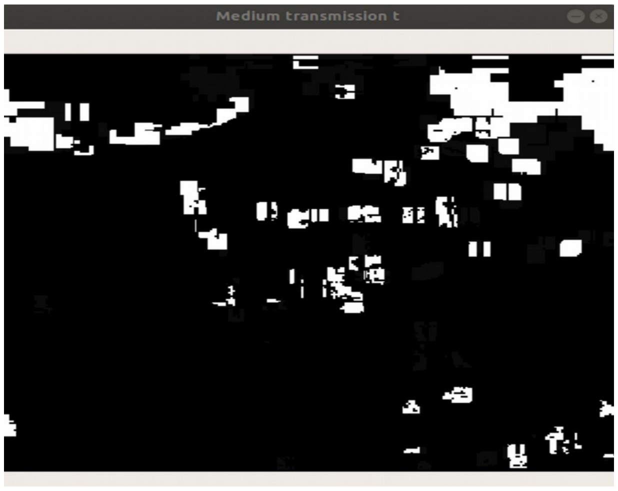


Fig 6.3 Medium transmission

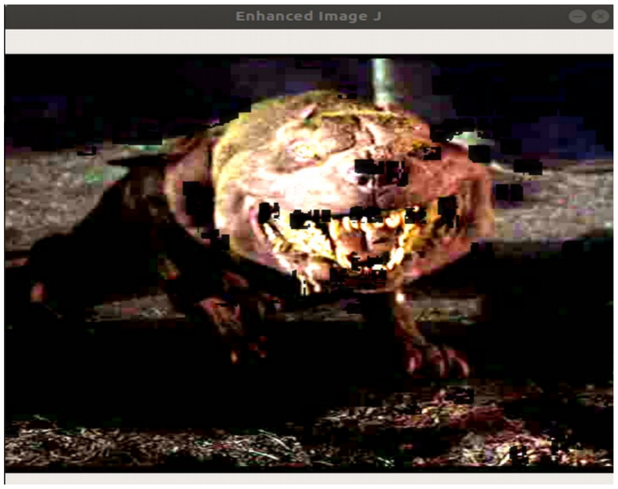


Fig 6.4 Enhanced Frame

The performance parameter of the output frame is shown in the following table (Table 6.1) which includes the PSNR, SNR and processing time. The value of the PSNR and SNR of the recover frames are improved after the enhancement of night time, the value of SNR and PSNR becomes high, which means that the noise is reduced.

Table 6.1 Performance parameter

Input frames [1242(W),375 (H)]	PSNR	SNR	Processing time Frames /s
Alveo	48.80	11.256	629

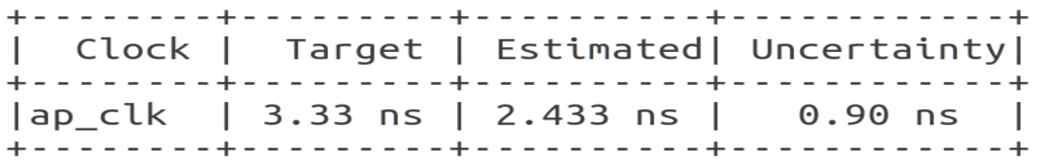
The result on the table (Table 6.2–6.9) below belongs to the Vivado HLS, which represents the resources used, during hardware implementation using Alveo u200 as a platform.

The result below shows that our proposed method is efficient, with low latency (maximum 27.736 ms), and target time (approximately equals to 3.3ns),

Table 6.2 Summary of resources used of FPGA

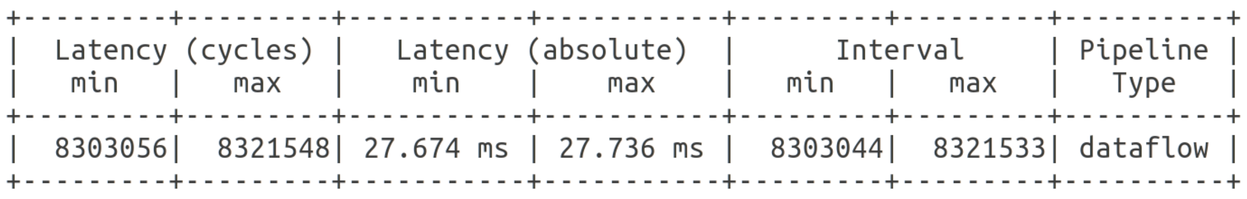
Clock timing	latency	BRAM	DSP	FF	LUT
2.44ns	27ms	1 %	0.4%	0.6%	4%

Table 6.3Timing summary of apc_cl



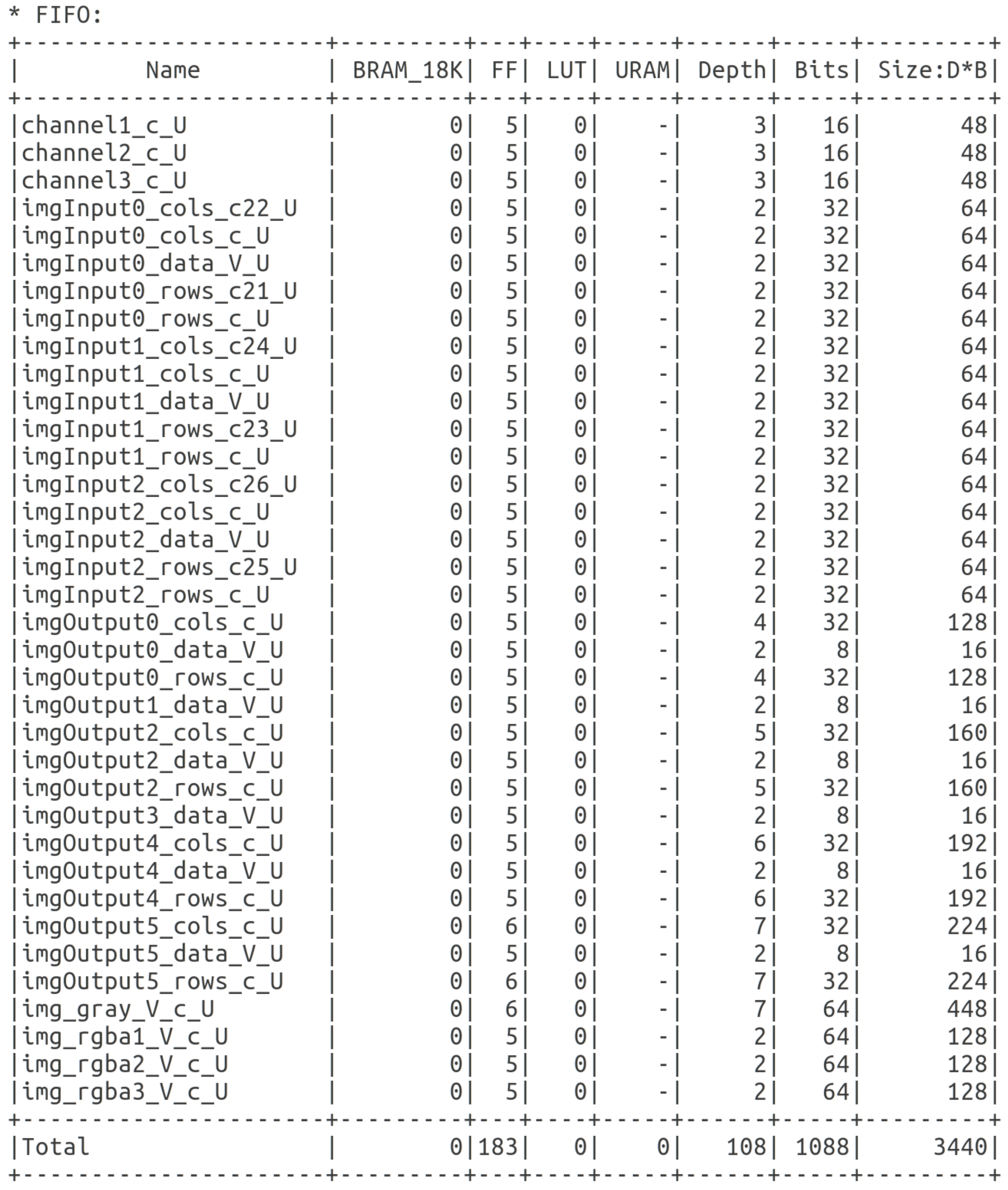
```
+--------+---------+----------+------------+
|  Clock |  Target | Estimated| Uncertainty|
+--------+---------+----------+------------+
|ap_clk  | 3.33 ns | 2.433 ns |   0.90 ns  |
+--------+---------+----------+------------+
```

Table 6.4 latency of DCP



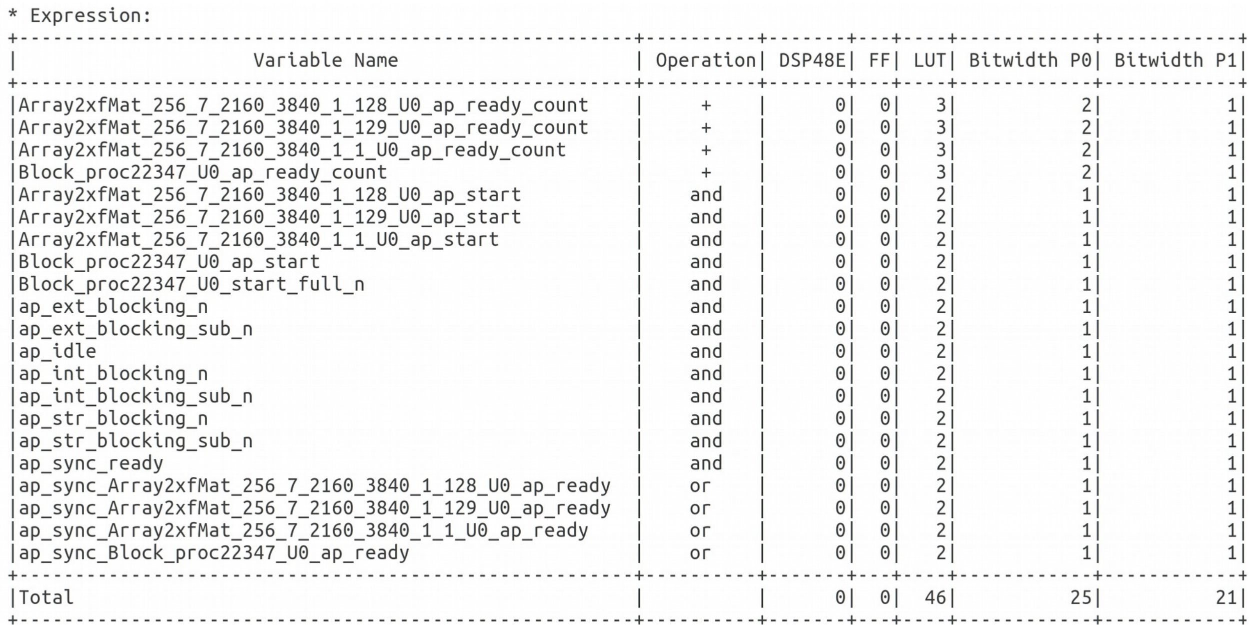
```
+---------+---------+-----------+-----------+---------+---------+----------+
|  Latency (cycles) |   Latency (absolute)  |      Interval     | Pipeline |
|   min   |   max   |    min    |    max    |   min   |   max   |   Type   |
+---------+---------+-----------+-----------+---------+---------+----------+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

Table 6.5 FIFO resources



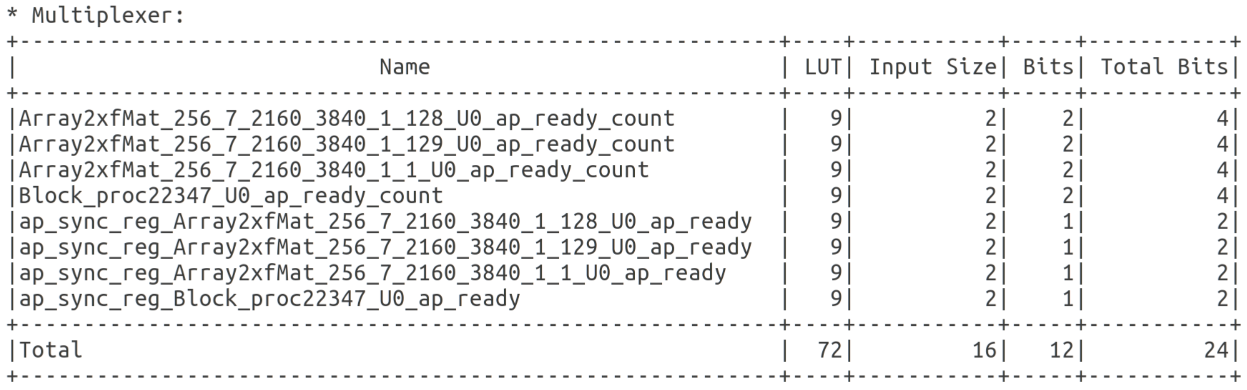
```
* FIFO:
+---------------------+---------+---+----+-----+------+-----+---------+
|         Name        | BRAM_18K| FF| LUT| URAM| Depth| Bits| Size:D*B|
+---------------------+---------+---+----+-----+------+-----+---------+
|channel1_c_U         |        0|  5|   0|    -|     3|   16|       48|
|channel2_c_U         |        0|  5|   0|    -|     3|   16|       48|
|channel3_c_U         |        0|  5|   0|    -|     3|   16|       48|
|imgInput0_cols_c22_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput0_cols_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput0_data_V_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput0_rows_c21_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput0_rows_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput1_cols_c24_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput1_cols_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput1_data_V_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput1_rows_c23_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput1_rows_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput2_cols_c26_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput2_cols_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput2_data_V_U   |        0|  5|   0|    -|     2|   32|       64|
|imgInput2_rows_c25_U |        0|  5|   0|    -|     2|   32|       64|
|imgInput2_rows_c_U   |        0|  5|   0|    -|     2|   32|       64|
|imgOutput0_cols_c_U  |        0|  5|   0|    -|     4|   32|      128|
|imgOutput0_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput0_rows_c_U  |        0|  5|   0|    -|     4|   32|      128|
|imgOutput1_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput2_cols_c_U  |        0|  5|   0|    -|     5|   32|      160|
|imgOutput2_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput2_rows_c_U  |        0|  5|   0|    -|     5|   32|      160|
|imgOutput3_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput4_cols_c_U  |        0|  5|   0|    -|     6|   32|      192|
|imgOutput4_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput4_rows_c_U  |        0|  5|   0|    -|     6|   32|      192|
|imgOutput5_cols_c_U  |        0|  6|   0|    -|     7|   32|      224|
|imgOutput5_data_V_U  |        0|  5|   0|    -|     2|    8|       16|
|imgOutput5_rows_c_U  |        0|  6|   0|    -|     7|   32|      224|
|img_gray_V_c_U       |        0|  6|   0|    -|     7|   64|      448|
|img_rgba1_V_c_U      |        0|  5|   0|    -|     2|   64|      128|
|img_rgba2_V_c_U      |        0|  5|   0|    -|     2|   64|      128|
|img_rgba3_V_c_U      |        0|  5|   0|    -|     2|   64|      128|
+---------------------+---------+---+----+-----+------+-----+---------+
|Total                |        0|183|   0|    0|   108| 1088|     3440|
+---------------------+---------+---+----+-----+------+-----+---------+
```

Table 6.6 Expression resources used



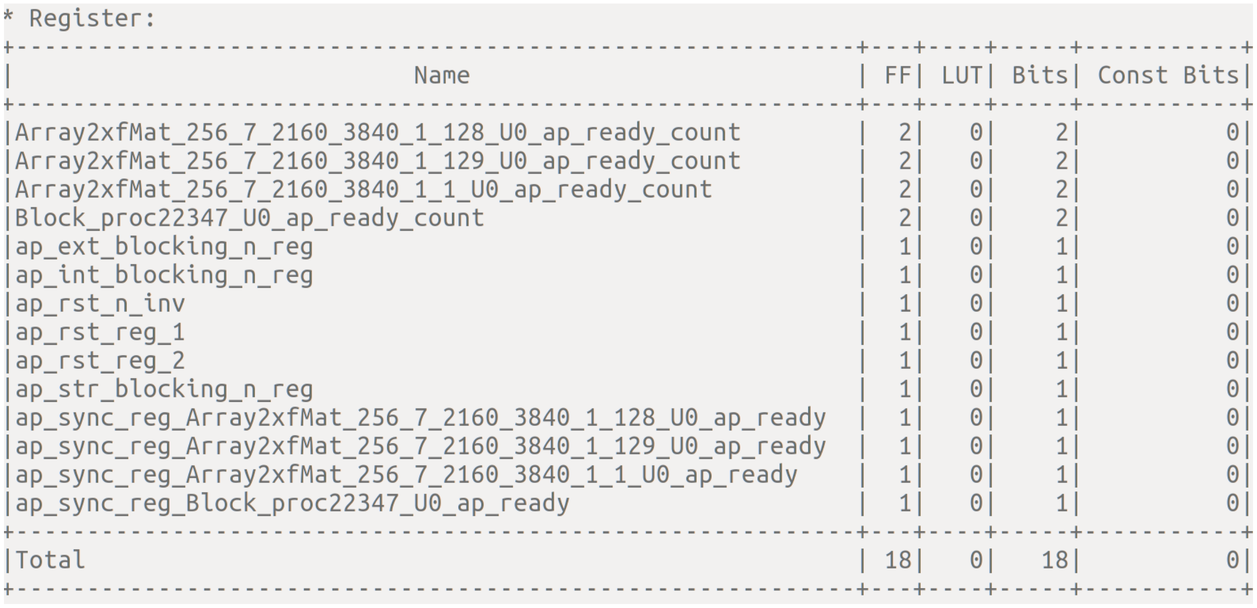
```
* Expression:
+--------------------------------------------------------+----------+------+---+----+------------+------------+
|                    Variable Name                       | Operation| DSP48E| FF| LUT| Bitwidth P0| Bitwidth P1|
+--------------------------------------------------------+----------+------+---+----+------------+------------+
|Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready_count    |    +     |     0|  0|   3|           2|           1|
|Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready_count    |    +     |     0|  0|   3|           2|           1|
|Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready_count      |    +     |     0|  0|   3|           2|           1|
|Block_proc22347_U0_ap_ready_count                      |    +     |     0|  0|   3|           2|           1|
|Array2xfMat_256_7_2160_3840_1_128_U0_ap_start          |   and    |     0|  0|   2|           1|           1|
|Array2xfMat_256_7_2160_3840_1_129_U0_ap_start          |   and    |     0|  0|   2|           1|           1|
|Array2xfMat_256_7_2160_3840_1_1_U0_ap_start            |   and    |     0|  0|   2|           1|           1|
|Block_proc22347_U0_ap_start                            |   and    |     0|  0|   2|           1|           1|
|Block_proc22347_U0_start_full_n                        |   and    |     0|  0|   2|           1|           1|
|ap_ext_blocking_n                                      |   and    |     0|  0|   2|           1|           1|
|ap_ext_blocking_sub_n                                  |   and    |     0|  0|   2|           1|           1|
|ap_idle                                                |   and    |     0|  0|   2|           1|           1|
|ap_int_blocking_n                                      |   and    |     0|  0|   2|           1|           1|
|ap_int_blocking_sub_n                                  |   and    |     0|  0|   2|           1|           1|
|ap_str_blocking_n                                      |   and    |     0|  0|   2|           1|           1|
|ap_str_blocking_sub_n                                  |   and    |     0|  0|   2|           1|           1|
|ap_sync_ready                                          |   and    |     0|  0|   2|           1|           1|
|ap_sync_Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready  |   or     |     0|  0|   2|           1|           1|
|ap_sync_Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready  |   or     |     0|  0|   2|           1|           1|
|ap_sync_Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready    |   or     |     0|  0|   2|           1|           1|
|ap_sync_Block_proc22347_U0_ap_ready                    |   or     |     0|  0|   2|           1|           1|
+--------------------------------------------------------+----------+------+---+----+------------+------------+
|Total                                                  |          |     0|  0|  46|          25|          21|
+--------------------------------------------------------+----------+------+---+----+------------+------------+
```

Table 6.7 Multiplexer resources used



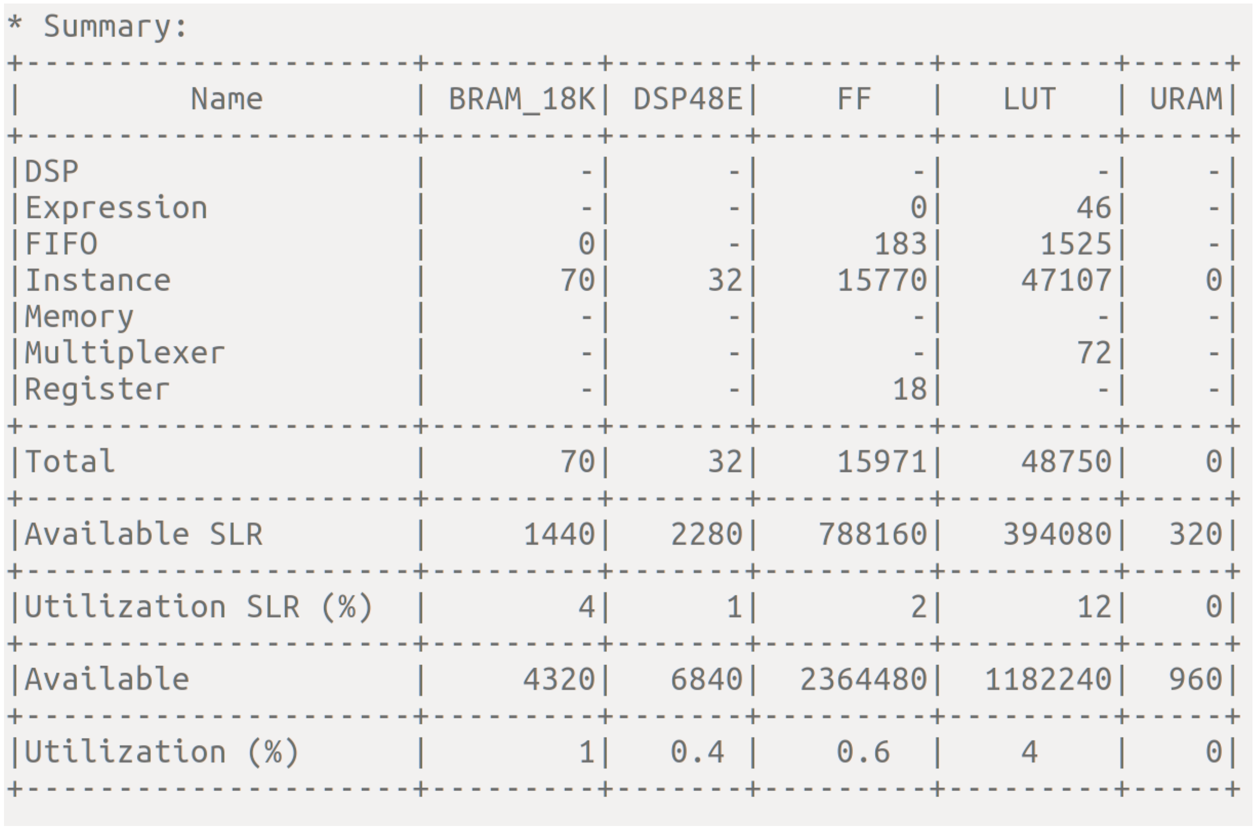
```
* Multiplexer:
+--------------------------------------------------------+----+-----------+-----+-----------+
|                          Name                          | LUT| Input Size| Bits| Total Bits|
+--------------------------------------------------------+----+-----------+-----+-----------+
|Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready_count     |   9|          2|    2|          4|
|Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready_count     |   9|          2|    2|          4|
|Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready_count       |   9|          2|    2|          4|
|Block_proc22347_U0_ap_ready_count                       |   9|          2|    2|          4|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready |   9|          2|    1|          2|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready |   9|          2|    1|          2|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready |   9|          2|    1|          2|
|ap_sync_reg_Block_proc22347_U0_ap_ready                 |   9|          2|    1|          2|
+--------------------------------------------------------+----+-----------+-----+-----------+
|Total                                                   |  72|         16|   12|         24|
+--------------------------------------------------------+----+-----------+-----+-----------+
```

Table 6.8 Register used



```
* Register:
+-----------------------------------------------------------+---+----+-----+-----------+
|                           Name                            | FF| LUT| Bits| Const Bits|
+-----------------------------------------------------------+---+----+-----+-----------+
|Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready_count        |  2|   0|    2|          0|
|Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready_count        |  2|   0|    2|          0|
|Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready_count          |  2|   0|    2|          0|
|Block_proc22347_U0_ap_ready_count                          |  2|   0|    2|          0|
|ap_ext_blocking_n_reg                                      |  1|   0|    1|          0|
|ap_int_blocking_n_reg                                      |  1|   0|    1|          0|
|ap_rst_n_inv                                               |  1|   0|    1|          0|
|ap_rst_reg_1                                               |  1|   0|    1|          0|
|ap_rst_reg_2                                               |  1|   0|    1|          0|
|ap_str_blocking_n_reg                                      |  1|   0|    1|          0|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_128_U0_ap_ready  |  1|   0|    1|          0|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_129_U0_ap_ready  |  1|   0|    1|          0|
|ap_sync_reg_Array2xfMat_256_7_2160_3840_1_1_U0_ap_ready    |  1|   0|    1|          0|
|ap_sync_reg_Block_proc22347_U0_ap_ready                    |  1|   0|    1|          0|
+-----------------------------------------------------------+---+----+-----+-----------+
|Total                                                      | 18|   0|   18|          0|
+-----------------------------------------------------------+---+----+-----+-----------+
```

Table 6.9 Summary of resources used



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

CHAPTER 7

CONCLUSION

In this project, I conclude about the progress of the project and states that in this project, I have studied and practiced the enhancement of night time using DCP based algorithm on software part using core i-7 (RAM 16 Go) and evaluated the speed of video processing with 240p,360,480p 720p 1080p, and 4K nighttime video, the video processing speed of the nighttime video with 240,360,480p can be continuous using software, while for the ultra-high definition video (720p,1080p and 4k), the video processing speed becomes slow. Using hardware implementation on the FPGA Alveo board was solving this problem, the enhancement of night input video can be continuous with low latency and fast processing on FPGA. Here the atmospheric light of the input frame is calculated on software using python language. The experiment result showed that our proposed method can achieve real-time video processing with fast video processing compare to the DCP algorithm implemented on software. The future work is to increase the Compute Unit on kernel (FPGA) to enhance the speed of video processing.

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 novelframework for extremely low-light video enhancement. InDigest of Technical Papers-IEEE InternationalConferenceon Consumer Electronics. (pp.91-92).

5. Xuesong Jiang, Hongxun Yao, Shengping Zhang1, Xiusheng Lu1and 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, AdaptiveAlgorihms for Digital Mammogram Enhancement, SPIE MedicalImaging Conference San Diego, CA, 2008, pp. 16-21

8. D. H. Choi, I. H. Jang, M. H. Kim, and N. C. Kim, Color ImageEnhancement using Single-Scale Retinex Based on an Improved ImageFormation Model, EUSIPCO, 2008

9. Z. Rahman, D. J. Jobson, and G. A. Woodell, Multi Scale Retinex forColor Image Enhancement, NASA Langley Research Center, IEEE,1996

10. Shwartz, Namer, Schenchner, "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 oflow 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 onEffective Space Combined with Pixel Learning", MDP

14. K. He, J. Sun, and X. Tang, "Single image haze re-moval using dark channel prior," inProceedings of Con-ference 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,” proceeding Conference on Computer Vision and Pattern Re cog-nition 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

CHAPTER 8

DESCRIPTION OF ANNEX

This chapter describes the report which I have done during my internship at SEMI Lab of UMONS, and the description of the work plan and the publication, Vitis software tool is used in this project for the development of an application program and Alveo board is the platform used for the deployment of the executable file. Learning of this tool helped me to implement the enhancement of night time into the FPGA, Some difficulties were faced during the implementation as a selection of the interface used in High-Level Synthesis in hardware, and many more. This chapter described also all of the reports which I have done during my internship. The description of the annex is organized as below the first annex describe the work plan and the second annex is a tutorial of executing the application program using RTL Kernel and the third annex describe the features of the PYNQ board, the forth annex describes the data flow of compiling and linking an application program in the host and the kernel and the next annex the number of publication which I have done during the second year of M.Tech

ANNEX 1

WORK PLAN

Table 7.1 work plan

Monthly	July	Augst	Sept	Oct	Nov	Dec	Jan	Feb	March	April
Counseling	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Literature review	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Problem identification and study of software	✓	✓	✓	✓	✓	✓	✓			
Writing report			✓	✓	✓	✓	✓	✓	✓	✓

ANNEX 2

INTEGRATION OF RTL KERNEL IN APPLICATION PROGRAM

➔ *Introduction*

In Vitis software, an application is split into a host program and hardware-accelerated kernels, the host and kernel are interconnected by the PCIe. The host application is written in C/C++ using OpenCL API, and run on the processor, while the kernel program is written using C/C++ or RTL or OpenCL C and run in the Xilinx FPGA. The host program sets a command to set the kernel, the host writes the buffer to the kernel, command the kernel to execute the program and read the buffer and release the kernel. The figure below show the software/hardware Build process.

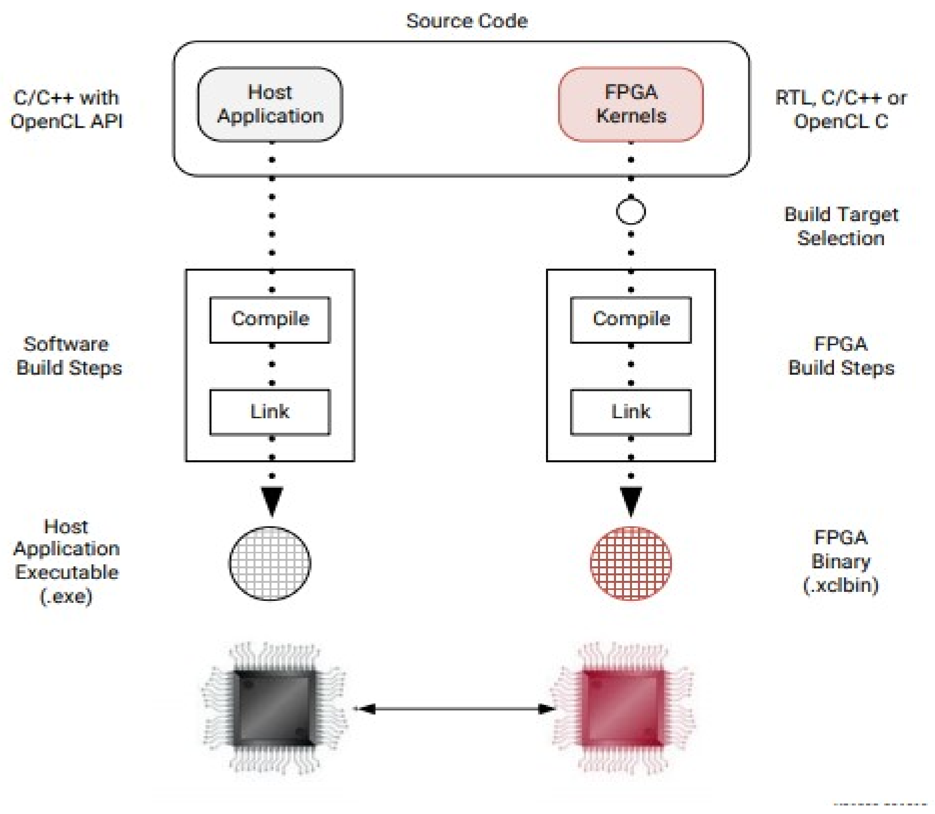


Figure 1.1 software/hardware Build Process **[ref 2]**

From this figure above we can see that the host application code and the FPGA Kernel code are compiled and linked separately the host application generate an executable host application (.exe) and the FPGA kernel generate an FPGA binary after compiled and linked.

- **Objective**

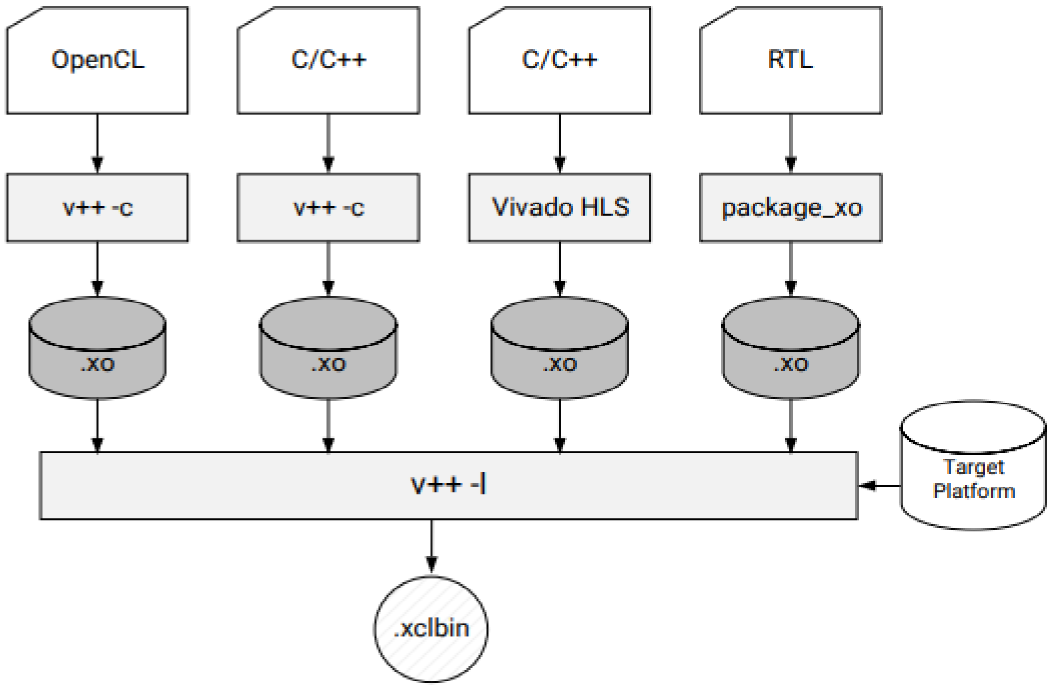
The main objective of this tutorial is to show all of the different steps to build an application program using RTL kernel as the kernel function

- **Motivation**

This tutorial leads us to better understand the environment of Vitis software, it also helps us to understand the designing flow of an application project using RTL kernel on Vitis.

- **Development technical details**

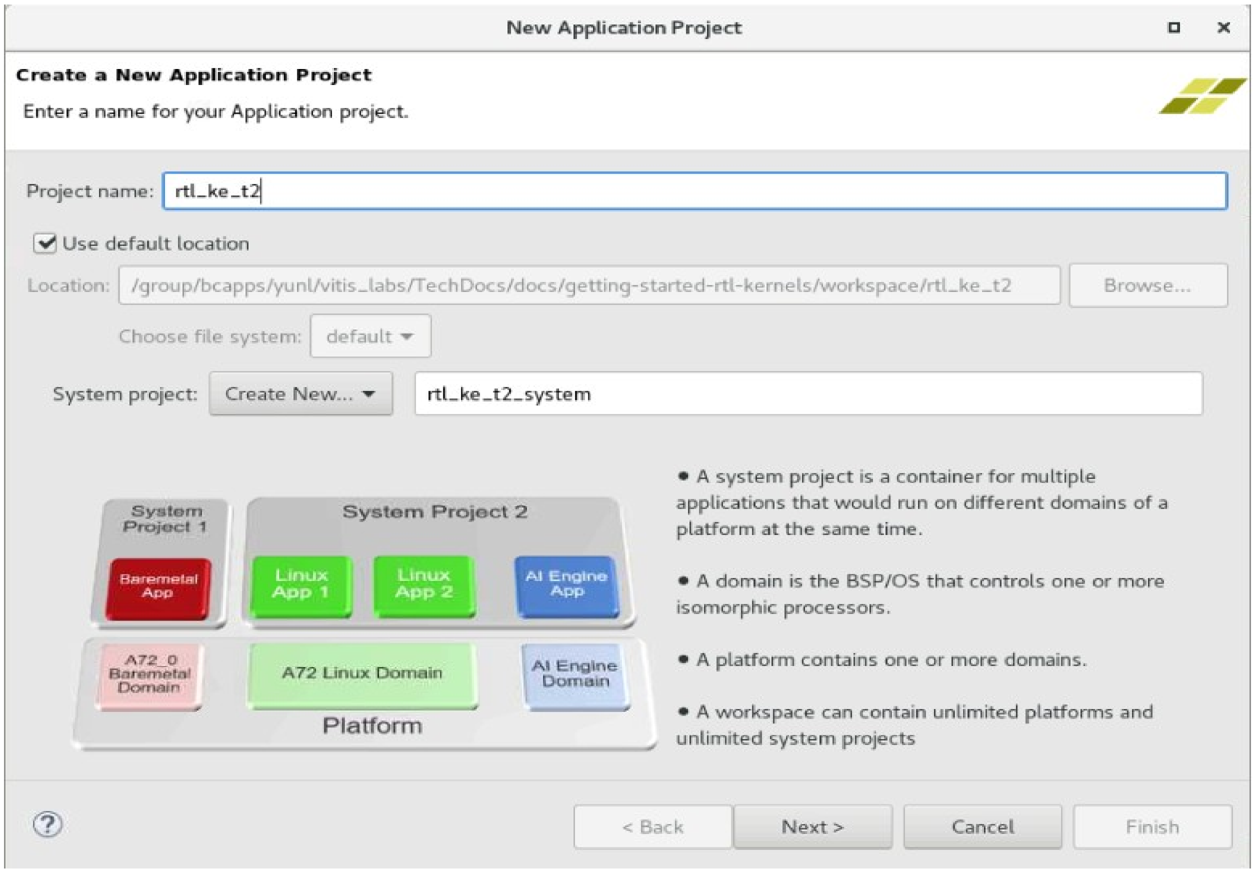
Before starting, the figure below (figure 1.2) shows us the FPGA Binary Build Process. Here we can see that the source code of the kernel is written in a different format as OpenCL, C/C++, and RTL. OpenCL and C/C++ are compiled using v++ -c-command while C/C++ can also package to .xo (Xilinx object) file using Vivado HLS, here RTL are package to (Xilinx object file) .xo through package _xo. After the kernel source are compiled, the Xilinx object file are linked with the target platform to generate the .xclbin



In this tutorial the host code and kernel code are already created in the reference file, to access the reference file from our laptop we have to write this command in our terminal:
git clone https://github.com/Xilinx/Vitis-Tutorials after we enter this command in our terminal we can access the reference file name Vitus-Tutorials through our laptop. Now we can start creating our first project.

➔ *Create a Vitis project*

- First click the icon of Vitis in the laptop, then the window of eclipse launcher will open, then select workspace through the browse on the right side and then click launch.
- Vitis IDE window will open.



New Application Project

Create a New Application Project

Enter a name for your Application project.

Project name: rtl_ke_t2

☑ Use default location

Location: /group/bcapps/yunl/vitis_labs/TechDocs/docs/getting-started-rtl-kernels/workspace/rtl_ke_t2

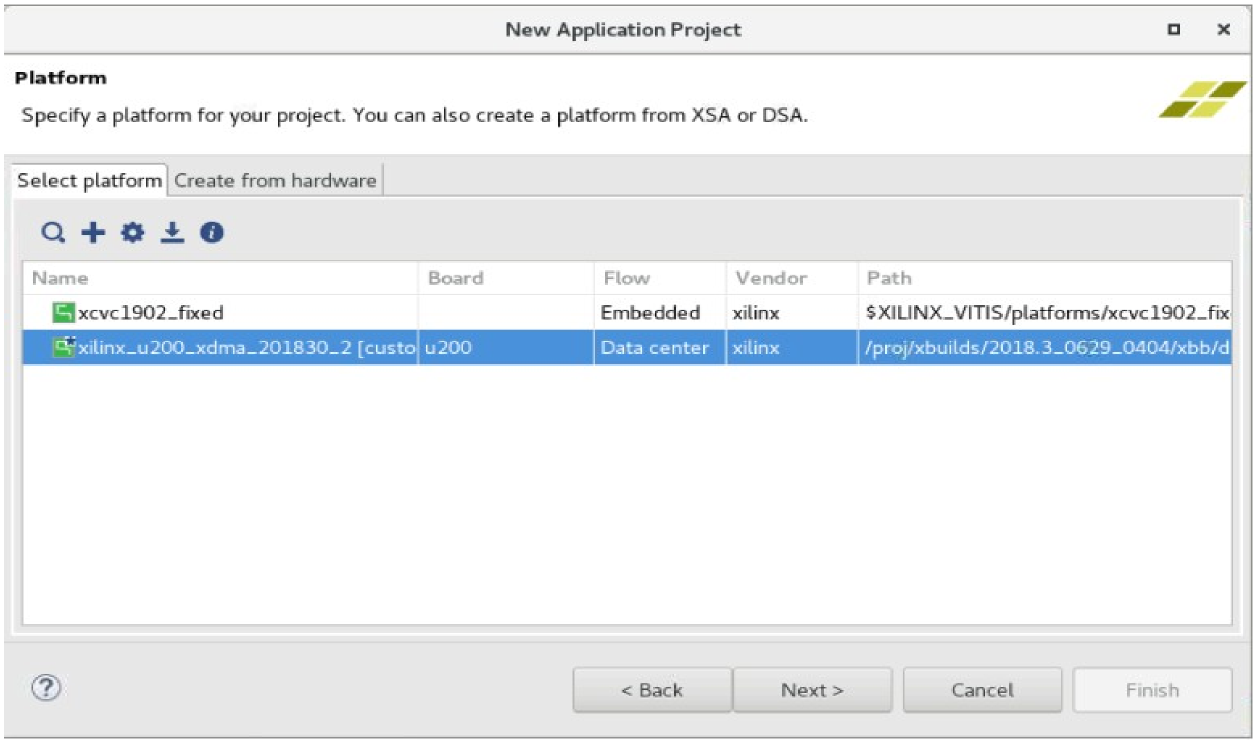
Choose file system: default

System project: Create New... rtl_ke_t2_system

- A system project is a container for multiple applications that would run on different domains of a platform at the same time.
- A domain is the BSP/OS that controls one or more isomorphic processors.
- A platform contains one or more domains.
- A workspace can contain unlimited platforms and unlimited system projects

When the Vitis IDE open, give a project name as we want, here we use rtl_ke_t2 for the project name, then click next

• Platform window will open, here we used xilinx_u200_xdma_201830_2 as our platform


New Application Project

Platform

Specify a platform for your project. You can also create a platform from XSA or DSA.

Select platform | Create from hardware

Name	Board	Flow	Vendor	Path
xcvc1902_fixed		Embedded	xilinx	$XILINX_VITIS/platforms/xcvc1902_fix
xilinx_u200_xdma_201830_2 [custo	u200	Data center	xilinx	/proj/xbuilds/2018.3_0629_0404/xbb/d

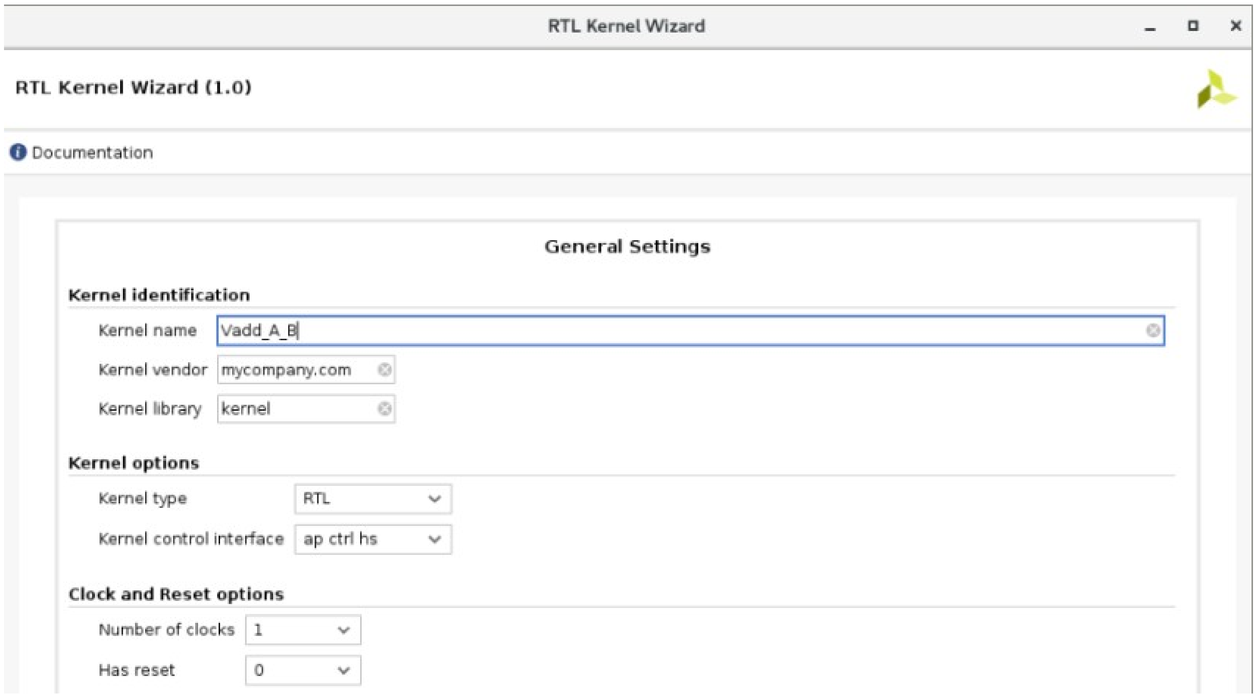
- After selection of platform click next
- The templates window will open

Select the empty application and click finish. The new project wizard closes and takes us back to the Vitis IDE.

➔ *Create and configure RTL kernel wizard*

The RTL Kernel Wizard guides you through the process of specifying the interface characteristics for an RTL kernel

- In the top menu click Xilinx and select RTL kernel wizard
- RTL kernel wizard window will open , in the general setting give a kernel name here we used (Vadd_A_B), and keep the other configuration and then click next


RTL Kernel Wizard

RTL Kernel Wizard (1.0)

Documentation

General Settings

Kernel identification

Kernel name	Vadd_A_B
Kernel vendor	mycompany.com
Kernel library	kernel

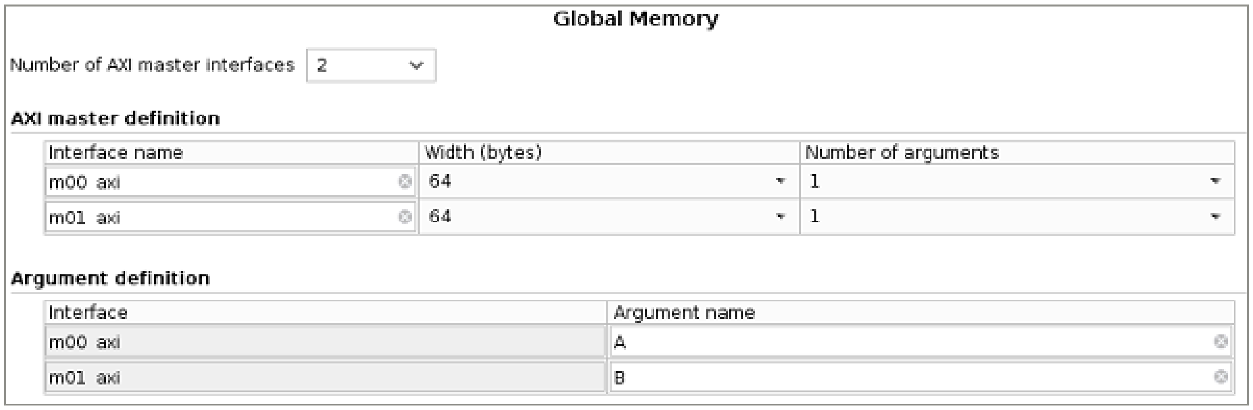
Kernel options

Kernel type	RTL
Kernel control interface	ap ctrl hs

Clock and Reset options

Number of clocks	1
Has reset	0

- The scalar argument window will open and keep the default value and click next
- The global memory window will open,for Number of AXI master interfaces, select **2** because the Vector-Accumulate kernel has two AXI4 interfaces (for A and for B), in the argument name change them into A and B and then click next


Global Memory

Number of AXI master interfaces: 2

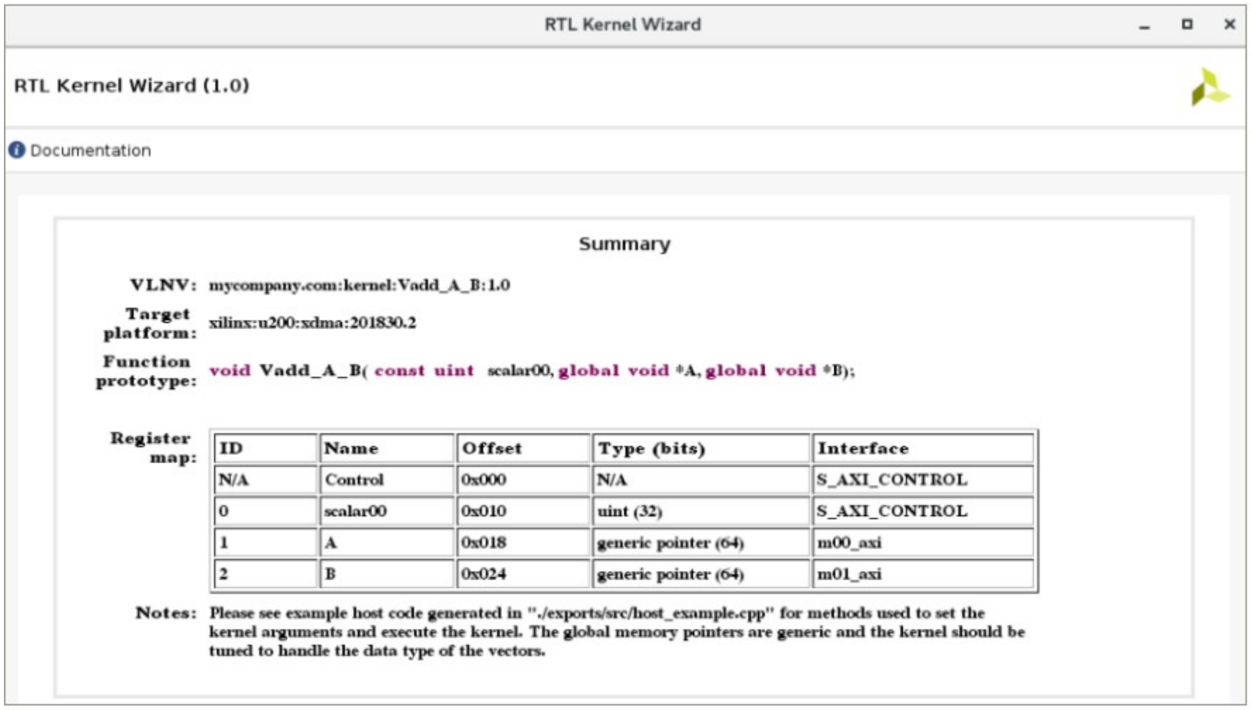
AXI master definition

Interface name	Width (bytes)	Number of arguments
m00 axi	64	1
m01 axi	64	1

Argument definition

Interface	Argument name
m00 axi	A
m01 axi	B

- Streaming interfaces will open , here we do not need a streaming interface(interconnect a kernel with another kernel), keep the default value and then click next
- Example summary page will open



Summary

VLNV: mycompany.com:kernel:Vadd_A_B:1.0

Target platform: xilinx:u200:xdma:201830.2

Function prototype: `void Vadd_A_B( const uint scalar00, global void *A, global void *B);`

Register map:

ID	Name	Offset	Type (bits)	Interface
N/A	Control	0x000	N/A	S_AXI_CONTROL
0	scalar00	0x010	uint (32)	S_AXI_CONTROL
1	A	0x018	generic pointer (64)	m00_axi
2	B	0x024	generic pointer (64)	m01_axi

Notes: Please see example host code generated in "./exports/src/host_example.cpp" for methods used to set the kernel arguments and execute the kernel. The global memory pointers are generic and the kernel should be tuned to handle the data type of the vectors.

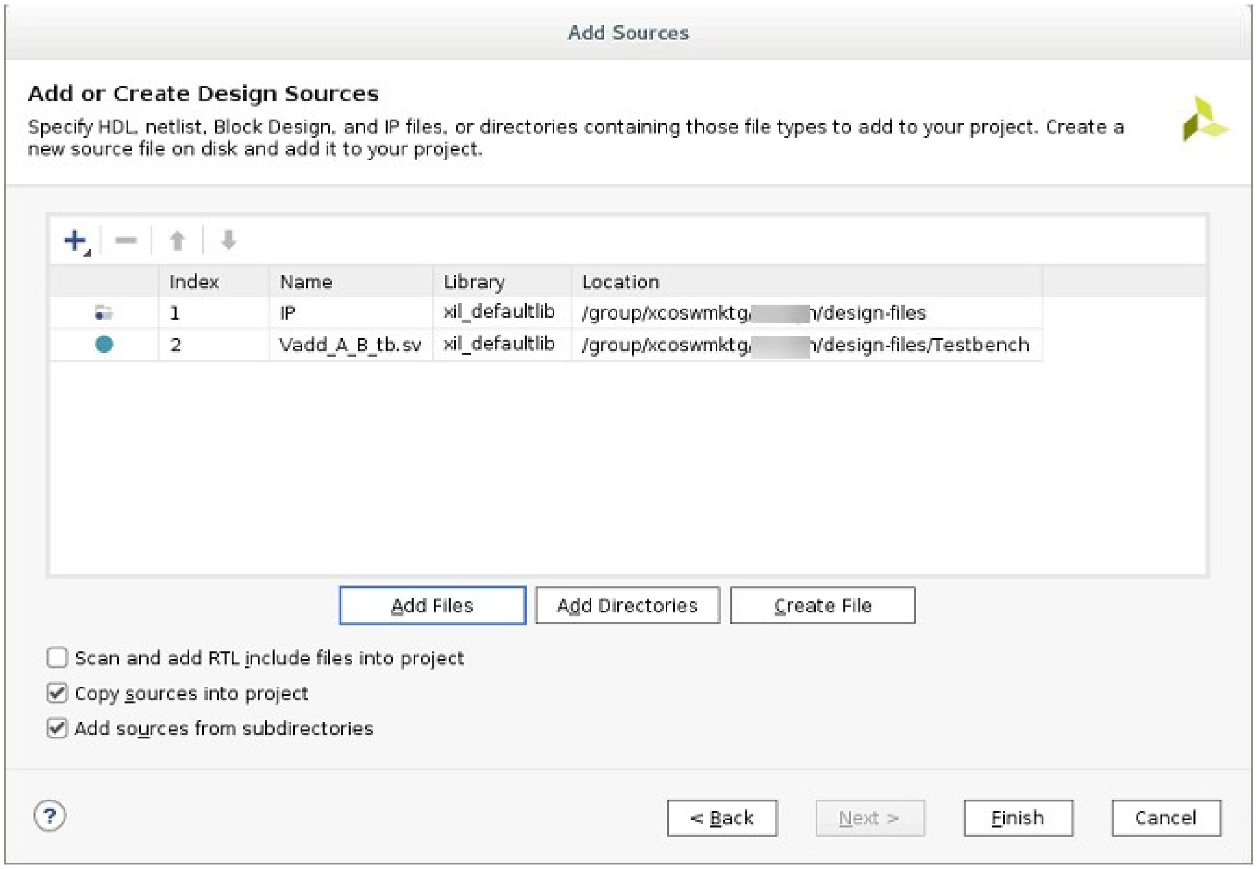
The new project wizard closes and takes you back to the Vitis IDE.

- To launch vivado design suite to package the RTL IP , click OK
- To launch Vivado Design Suite to package the RTL IP and create the kernel, click **OK**.

➔*RTL DESIGN Vivado design suite*

After configuration of the RTL kernel wizard the vivado design suite opened

- select **Compile Order** > **Synthesis**, and then expand the Design Sources tree.
- Select all of 8 example sources generated by the RTL kernel wizard and remove it
- Then go to simulation and remove the Vadd_A_b_tb.sv and remove it
- After that we can import our IP from the reference file, right click in the design source and click import and then import the directory of IP in the reference file and import also the file in the testbench


Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
	1	IP	xil_defaultlib	/group/xcoswmktg/[redacted]/design-files
●	2	Vadd_A_B_tb.sv	xil_defaultlib	/group/xcoswmktg/[redacted]/design-files/Testbench

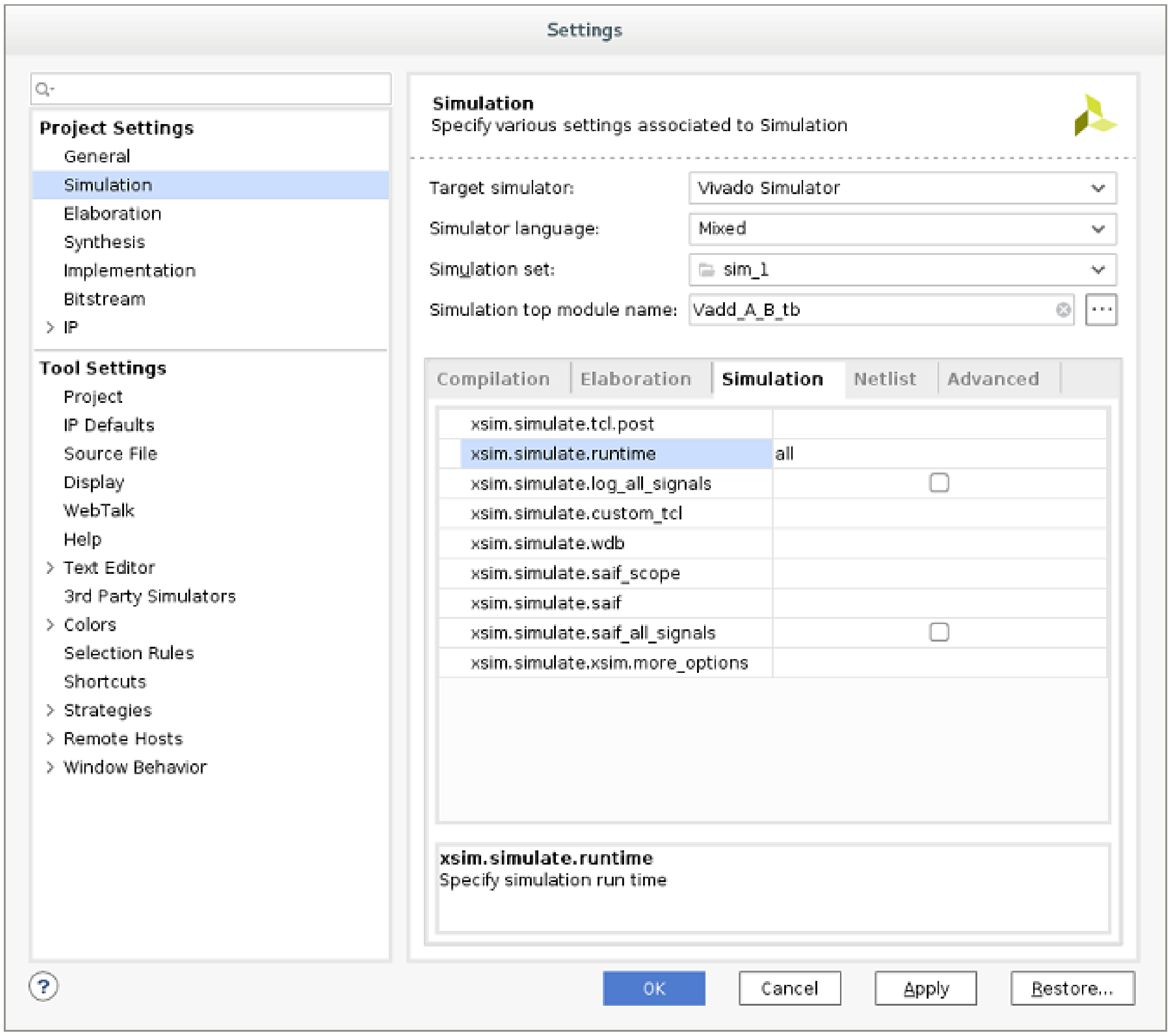
☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

- After that click finish
- Right-click **Vadd_A_B_tb.sv**, and then select **Move to Simulation Sources**.

➔ ***Simulation***

To use this test bench for verifying the Vector addition kernel:

- In the Flow Navigator, right-click **Run Simulation**, and then select **Simulation Settings**.
- In the settings dialog box, select simulation
- Change the **xsim.simulate.runtime** value to all as shown in the following figure



Simulation
Specify various settings associated to Simulation

Target simulator:	Vivado Simulator
Simulator language:	Mixed
Simulation set:	sim_1
Simulation top module name:	Vadd_A_B_tb

xsim.simulate.tcl.post	
xsim.simulate.runtime	all
xsim.simulate.log_all_signals	☐
xsim.simulate.custom_tcl	
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

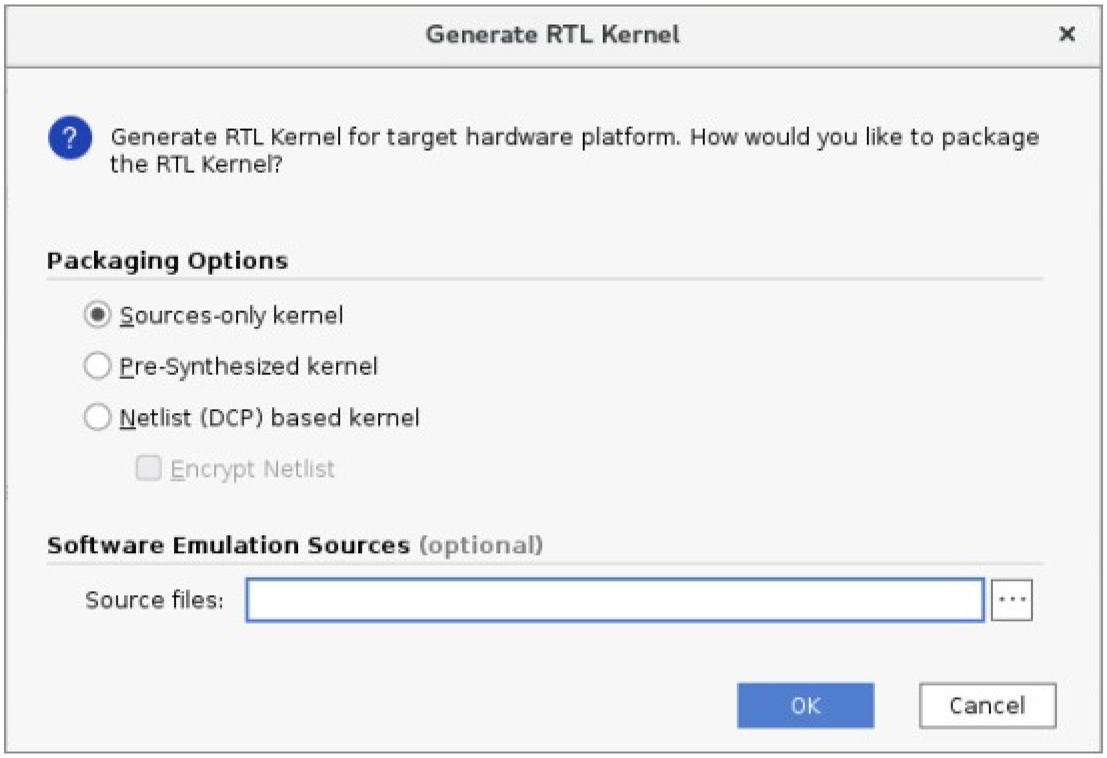
xsim.simulate.runtime
Specify simulation run time

- Click ok and run simulation

➔ *Package the RTL Kernel IP*

Now we can package the RTL kernel IP into .xo file

- From the Flow Navigator, click **Generate RTL Kernel**. The Generate RTL Kernel dialog box is opened, as shown in the following figure.


Generate RTL Kernel

Generate RTL Kernel for target hardware platform. How would you like to package the RTL Kernel?

Packaging Options

[✓ Sources-only kernel]

Pre-Synthesized kernel

Netlist (DCP) based kernel

Encrypt Netlist

Software Emulation Sources (optional)

Source files:

- Select sources only kernel
- Click ok the vivado design suite will close and return control to vitis IDE.RTL kernel is imported to the Vitis IDE

➔ *Using RTL kernel in vitis IDE*

The following files are added into a vitis IDE

- Vadd_A_B.xo compile kernel file
- Host_example.cpp example host application

In this tutorial we already had a host application in the reference file so we can remove the host example

1 Right click the host example and remove it

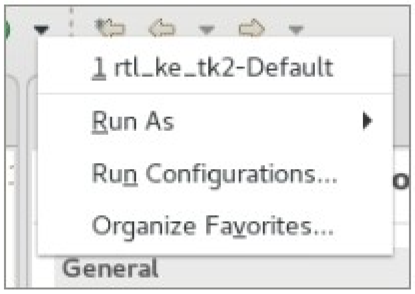
2 In the project explorer, right click and import the host.cpp from the reference and place it in the Vadd_A_B directory

3 After we import the host.cpp we can view the code by double click the file.

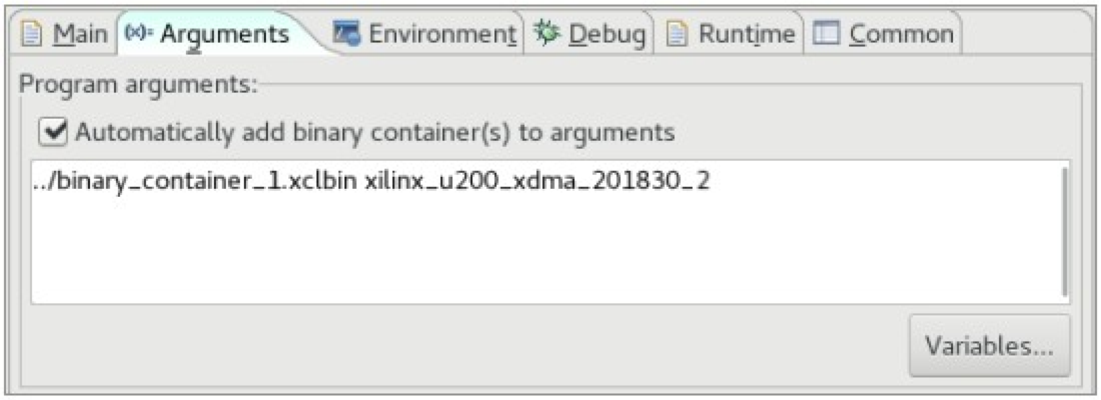
Now we are ready to build and run the application program

➔ *Build and run*

- To create a binary container, select the RTL kernel as the hardware function. In the Hardware Functions window, click the small lightning icon which is used for adding hardware functions into the project and select the Vadd_A_B function.
- In the vitis application project setting, change active build configuration to emulation hardware.
- At the top select run and go to run configuration



- When run configuration window open go to argument and select **automatically and binary container**



Program arguments:

☑ Automatically add binary container(s) to arguments

```
../binary_container_1.xclbin xilinx_u200_xdma_201830_2
```

When the .xclbin add into argument add the platform (xilinx_u200_xdma_201830_2) as the second input arguments.

- Then click the build icon (it will build the application project)
- And then run

ANNEX 3

PYNQ BOARD DESCRIPTIONS

➔ *Features of PYNQ:*

- **ZYNQ XC7Z020-1CLG400C**
    - 650MHz dual-core Cortex-A9 processor
    - DDR3 memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
    - High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
    - Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
    - Programmable from JTAG, Quad-SPI flash, and microSD card
    - Programmable logic equivalent to Artix-7 FPGA
        - 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops
        - 630 KB of fast block RAM
        - 4 clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM)
        - 220 DSP slices
        - On-chip analog-to-digital converter (XADC)
- **Memory**
    - 512MB DDR3 with 16-bit bus @ 1050Mbps
    - 16MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64™ compatible identifier
    - microSD slot
- **Power**
    - Powered from USB or any 7V-15V external power source
- **USB and Ethernet**
    - Gigabit Ethernet PHY
    - USB-JTAG Programming circuitry
    - USB-UART bridge

    - USB OTG PHY (supports host only)
- **Audio and Video**
    - HDMI sink port (input)
    - HDMI source port (output)
    - Microphone with PDM interface
    - PWM driven mono audio output with 3.5mm jack
- **Switches, Push-buttons, and LEDs**
    - 4 push-buttons
    - 2 slide switches
    - 4 LEDs
    - 2 RGB LEDs
- **Expansion Connectors**
    - Two standard Pmod ports
        - 16 Total FPGA I/O
    - Arduino/chipKIT Shield connector
        - 49 Total FPGA I/O
        - 6 Single-ended 0-3.3V Analog inputs to XADC
        - 4 Differential 0-1.0V Analog inputs to XADC

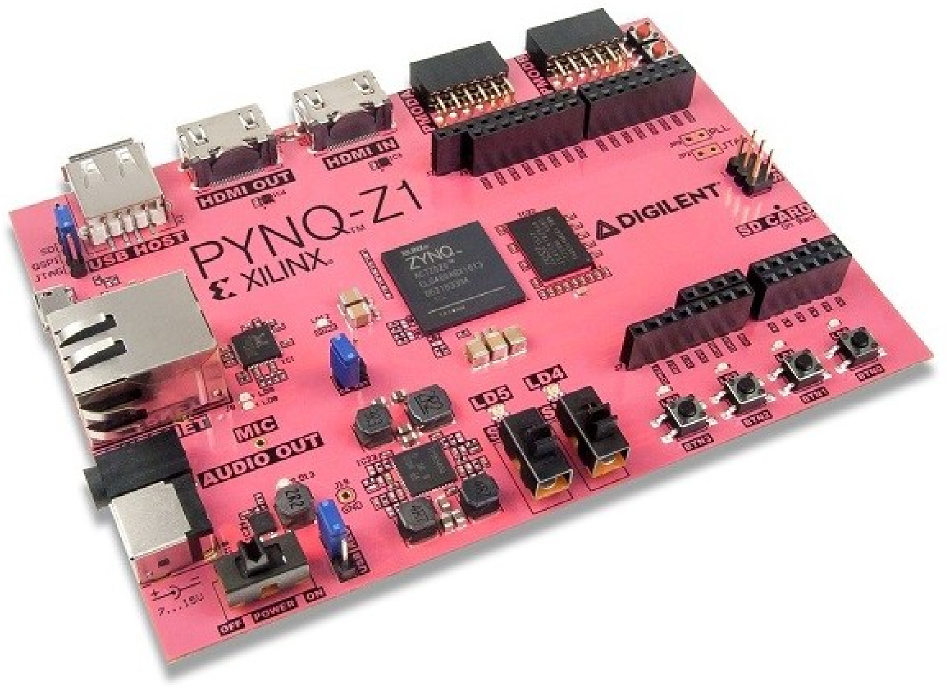


Fig 3.2.2 PYNQ BOARD [17]

ANNEX 4

BUILDING THE HOST AND THE KERNEL

➔ *Introduction*

compiled and linked is one of the important step we should do before we run the project into hardware or hardware emulation or software emulation. this tutorial help us to better understand all of the points we should do during compiling and linking. V++ is the command used in kernel function and g++ for the host.

- Compiler : convert the high level code into lower level code example (.cpp to .xo)
- link : as its name implies its used to link the host (xo file for hardware) with the platform to generate an executable file as (.exe or .xclbin)

➔ *Objective*

This tutorial show us how to compile, link and run an application project (host code and kernel function) using command v++ and g ++ and then emulate it into an hardware or emulation.

➔ *Development technical details*

the following step we have to do before compile and link :

- create a project using vitis software (the name of project here   example_of_firstP)

- then select platform

-when project create in vitis

- import the host code and kernel function in the reference file following this path

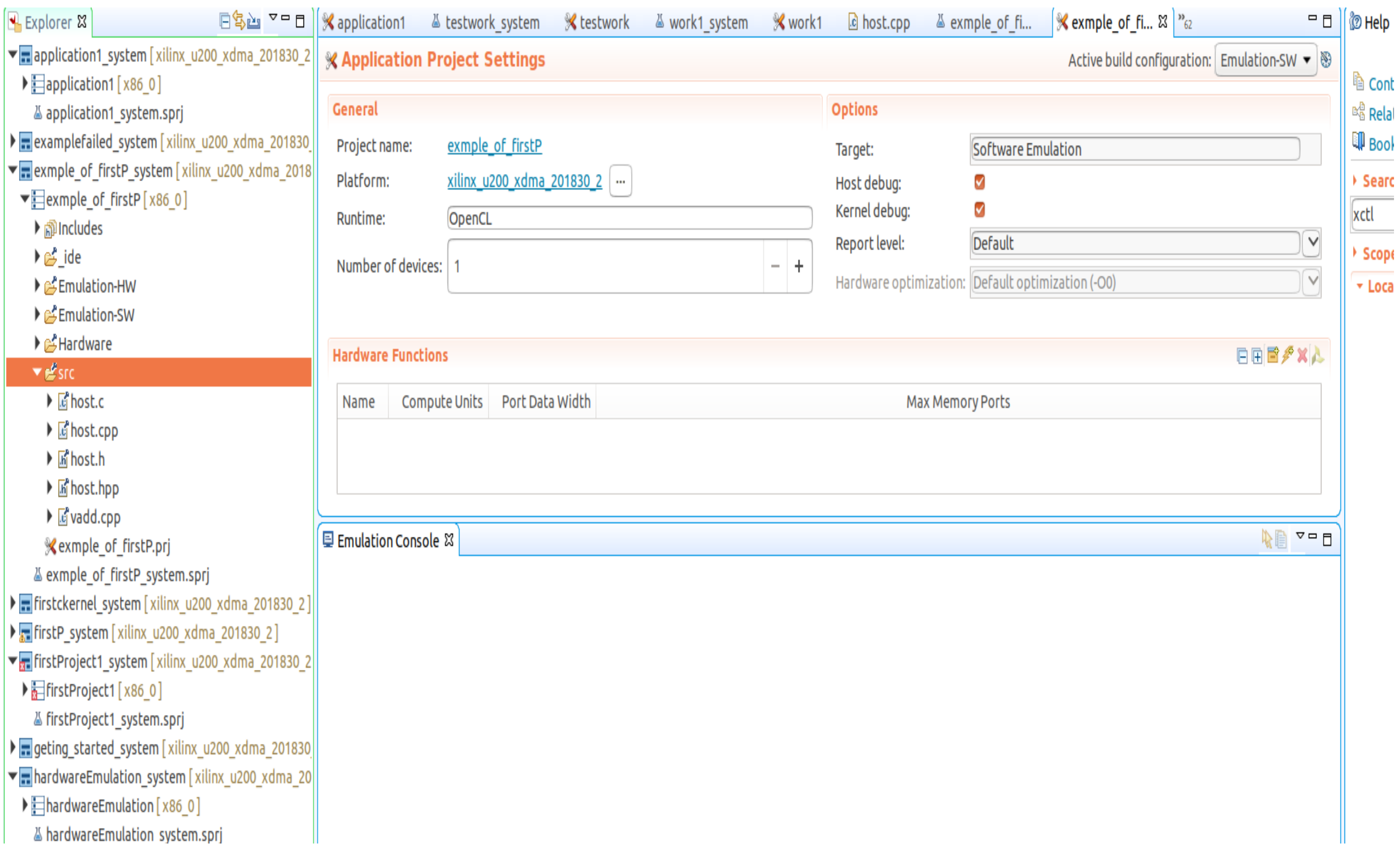
  Vitiis-Tutorials → Docs → my-first-program → reference-files → src.

-we are ready to compile and link the source code of host and kernel function after imported the file

When we already import all in the vitis software we can start compile and link our code .

➔ *Compile and link host code*

When we have imported all of the files include in src , see the figure bellow



Explorer

- application1_system [xilinx_u200_xdma_201830_2
  - application1 [x86_0]
  - application1_system.sprj
- examplefailed_system [xilinx_u200_xdma_201830
- exmple_of_firstP_system [xilinx_u200_xdma_2018
  - exmple_of_firstP [x86_0]
    - Includes
    - _ide
    - Emulation-HW
    - Emulation-SW
    - Hardware
    - src
      - host.c
      - host.cpp
      - host.h
      - host.hpp
      - vadd.cpp
    - exmple_of_firstP.prj
  - exmple_of_firstP_system.sprj
- firstckernel_system [xilinx_u200_xdma_201830_2]
- firstP_system [xilinx_u200_xdma_201830_2]
- firstProject1_system [xilinx_u200_xdma_201830_2
  - firstProject1 [x86_0]
  - firstProject1_system.sprj
- geting_started_system [xilinx_u200_xdma_201830
- hardwareEmulation_system [xilinx_u200_xdma_20
  - hardwareEmulation [x86_0]
  - hardwareEmulation_system.sprj

Application Project Settings

Active build configuration: Emulation-SW

General

Project name: exmple_of_firstP

Platform: xilinx_u200_xdma_201830_2

Runtime: OpenCL

Number of devices: 1

Options

Target: Software Emulation

Host debug: ✓

Kernel debug: ✓

Report level: Default

Hardware optimization: Default optimization (-O0)

Hardware Functions

Name	Compute Units	Port Data Width	Max Memory Ports

Emulation Console

We can combine the command of compile and link using g++ , **make sure that you are in the folder that had the host code** because in command bellow the path mention that our source code inside of src

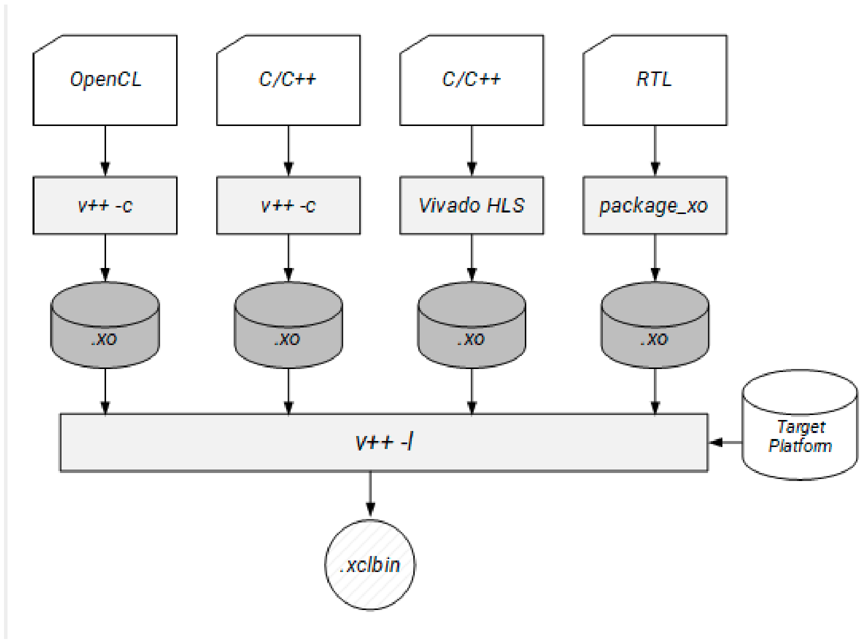
open terminal and go to the path you have your source code inside of your workspace here my path to go in src is **/home/semi/workspace/exmple_of_firstP/src**

command

```
$> g++ -I$XILINX_XRT/include/ -I$XILINX_VIVADO/include/ -Wall -O0 -g -std=c++11
./nameOfYourhost.cpp  -o 'host'  -L$XILINX_XRT/lib/ -lOpenCL -lpthread -lrt -lstdc++
```

➔ *Build kernel code*

we can compile the Kernel function with V++ compiler or through Vivado HLS or using package_xo.



➔ *Compiling kernel with vitis compiler*

Before starting make sure to setting up the vitis and xrt . To setting up the vitis and xrt open the terminal and run the command on terminal

```
#setup XILINX_VITIS and XILINX_VIVADO variables

        source <Vitis_install_path>/settings64.sh

        #setup XILINX_XRT

        source /opt/xilinx/xrt/setup.sh
```

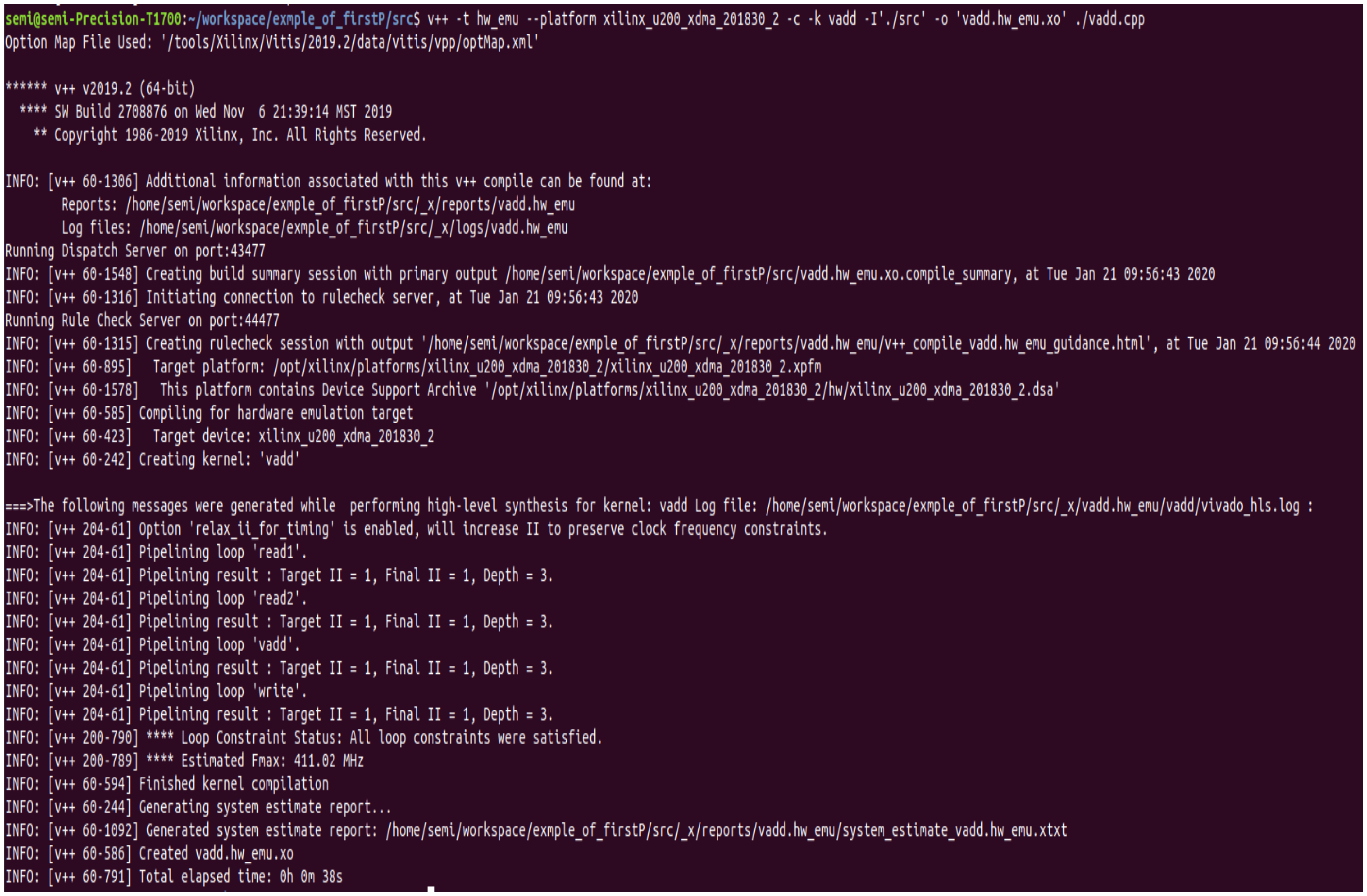
Before you compile your kernel source code check your location in terminal using command pwd, if you are not inside of ./src go to the ./src that you have your kernel code (krnl_vadd.cpp) as

: /home/semi/workspace/exmple_of_firstP/src

command to compile

$> v++ -t hw_emu --platform xilinx_u200_xdma_201830_2 -c -k vadd -I'./src' -o'vadd.hw_emu.xo' ./vadd.cpp

After you compile your kernel. You will get an output .xo file (krnl_vadd.hw_emu.xo) , now you are ready to linked your .xo file into .xclbin . You just type this command bellow to run your code in your terminal.

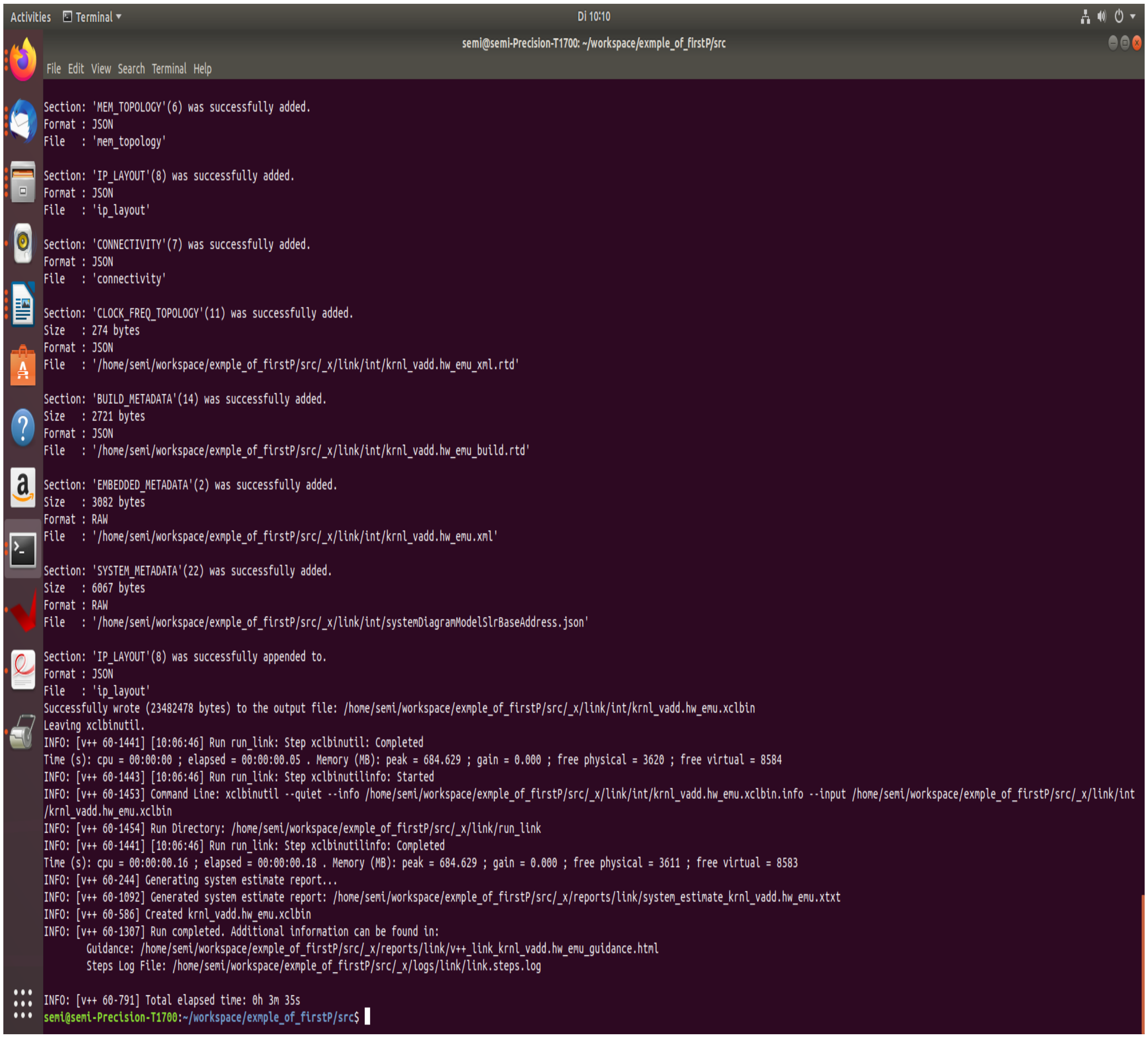


```
semi@semi-Precision-T1700:~/workspace/exmple_of_firstP/src$ v++ -t hw_emu --platform xilinx_u200_xdma_201830_2 -c -k vadd -I'./src' -o 'vadd.hw_emu.xo' ./vadd.cpp
Option Map File Used: '/tools/Xilinx/Vitis/2019.2/data/vitis/vpp/optMap.xml'

****** v++ v2019.2 (64-bit)
  **** SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: [v++ 60-1306] Additional information associated with this v++ compile can be found at:
        Reports: /home/semi/workspace/exmple_of_firstP/src/_x/reports/vadd.hw_emu
        Log files: /home/semi/workspace/exmple_of_firstP/src/_x/logs/vadd.hw_emu
Running Dispatch Server on port:43477
INFO: [v++ 60-1548] Creating build summary session with primary output /home/semi/workspace/exmple_of_firstP/src/vadd.hw_emu.xo.compile_summary, at Tue Jan 21 09:56:43 2020
INFO: [v++ 60-1316] Initiating connection to rulecheck server, at Tue Jan 21 09:56:43 2020
Running Rule Check Server on port:44477
INFO: [v++ 60-1315] Creating rulecheck session with output '/home/semi/workspace/exmple_of_firstP/src/_x/reports/vadd.hw_emu/v++_compile_vadd.hw_emu_guidance.html', at Tue Jan 21 09:56:44 2020
INFO: [v++ 60-895]   Target platform: /opt/xilinx/platforms/xilinx_u200_xdma_201830_2/xilinx_u200_xdma_201830_2.xpfm
INFO: [v++ 60-1578]   This platform contains Device Support Archive '/opt/xilinx/platforms/xilinx_u200_xdma_201830_2/hw/xilinx_u200_xdma_201830_2.dsa'
INFO: [v++ 60-585] Compiling for hardware emulation target
INFO: [v++ 60-423]   Target device: xilinx_u200_xdma_201830_2
INFO: [v++ 60-242] Creating kernel: 'vadd'

===>The following messages were generated while  performing high-level synthesis for kernel: vadd Log file: /home/semi/workspace/exmple_of_firstP/src/_x/vadd.hw_emu/vadd/vivado_hls.log :
INFO: [v++ 204-61] Option 'relax_ii_for_timing' is enabled, will increase II to preserve clock frequency constraints.
INFO: [v++ 204-61] Pipelining loop 'read1'.
INFO: [v++ 204-61] Pipelining result : Target II = 1, Final II = 1, Depth = 3.
INFO: [v++ 204-61] Pipelining loop 'read2'.
INFO: [v++ 204-61] Pipelining result : Target II = 1, Final II = 1, Depth = 3.
INFO: [v++ 204-61] Pipelining loop 'vadd'.
INFO: [v++ 204-61] Pipelining result : Target II = 1, Final II = 1, Depth = 3.
INFO: [v++ 204-61] Pipelining loop 'write'.
INFO: [v++ 204-61] Pipelining result : Target II = 1, Final II = 1, Depth = 3.
INFO: [v++ 200-790] **** Loop Constraint Status: All loop constraints were satisfied.
INFO: [v++ 200-789] **** Estimated Fmax: 411.02 MHz
INFO: [v++ 60-594] Finished kernel compilation
INFO: [v++ 60-244] Generating system estimate report...
INFO: [v++ 60-1092] Generated system estimate report: /home/semi/workspace/exmple_of_firstP/src/_x/reports/vadd.hw_emu/system_estimate_vadd.hw_emu.xtxt
INFO: [v++ 60-586] Created vadd.hw_emu.xo
INFO: [v++ 60-791] Total elapsed time: 0h 0m 38s
```

➔ *linking kernel with vitis compiler*

After you compile your kernel. You will get an output .xo file (vadd.hw_emu.xo) , now you are ready to linked your .xo file into .xclbin . You just type this command bellow to run your code in your terminal.

```
>$ v++ -t hw --platform xilinx_u200_xdma_201830_2 --link vadd.hw_emu.xo \

-o 'krnl_vadd.hw_emu.xclbin' --config ./connectivity.cfg
```


Activities
Terminal ▾
Di 10:10
semi@semi-Precision-T1700: ~/workspace/exmple_of_firstP/src
File Edit View Search Terminal Help

```
Section: 'MEM_TOPOLOGY'(6) was successfully added.
Format : JSON
File   : 'mem_topology'

Section: 'IP_LAYOUT'(8) was successfully added.
Format : JSON
File   : 'ip_layout'

Section: 'CONNECTIVITY'(7) was successfully added.
Format : JSON
File   : 'connectivity'

Section: 'CLOCK_FREQ_TOPOLOGY'(11) was successfully added.
Size   : 274 bytes
Format : JSON
File   : '/home/semi/workspace/exmple_of_firstP/src/_x/link/int/krnl_vadd.hw_emu_xml.rtd'

Section: 'BUILD_METADATA'(14) was successfully added.
Size   : 2721 bytes
Format : JSON
File   : '/home/semi/workspace/exmple_of_firstP/src/_x/link/int/krnl_vadd.hw_emu_build.rtd'

Section: 'EMBEDDED_METADATA'(2) was successfully added.
Size   : 3082 bytes
Format : RAW
File   : '/home/semi/workspace/exmple_of_firstP/src/_x/link/int/krnl_vadd.hw_emu.xml'

Section: 'SYSTEM_METADATA'(22) was successfully added.
Size   : 6067 bytes
Format : RAW
File   : '/home/semi/workspace/exmple_of_firstP/src/_x/link/int/systemDiagramModelSlrBaseAddress.json'

Section: 'IP_LAYOUT'(8) was successfully appended to.
Format : JSON
File   : 'ip_layout'
Successfully wrote (23482478 bytes) to the output file: /home/semi/workspace/exmple_of_firstP/src/_x/link/int/krnl_vadd.hw_emu.xclbin
Leaving xclbinutil.
INFO: [v++ 60-1441] [10:06:46] Run run_link: Step xclbinutil: Completed
Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.05 . Memory (MB): peak = 684.629 ; gain = 0.000 ; free physical = 3620 ; free virtual = 8584
INFO: [v++ 60-1443] [10:06:46] Run run_link: Step xclbinutilinfo: Started
INFO: [v++ 60-1453] Command Line: xclbinutil --quiet --info /home/semi/workspace/exmple_of_firstP/src/_x/link/int/krnl_vadd.hw_emu.xclbin.info --input /home/semi/workspace/exmple_of_firstP/src/_x/link/int
/krnl_vadd.hw_emu.xclbin
INFO: [v++ 60-1454] Run Directory: /home/semi/workspace/exmple_of_firstP/src/_x/link/run_link
INFO: [v++ 60-1441] [10:06:46] Run run_link: Step xclbinutilinfo: Completed
Time (s): cpu = 00:00:00.16 ; elapsed = 00:00:00.18 . Memory (MB): peak = 684.629 ; gain = 0.000 ; free physical = 3611 ; free virtual = 8583
INFO: [v++ 60-244] Generating system estimate report...
INFO: [v++ 60-1092] Generated system estimate report: /home/semi/workspace/exmple_of_firstP/src/_x/reports/link/system_estimate_krnl_vadd.hw_emu.xtxt
INFO: [v++ 60-586] Created krnl_vadd.hw_emu.xclbin
INFO: [v++ 60-1307] Run completed. Additional information can be found in:
        Guidance: /home/semi/workspace/exmple_of_firstP/src/_x/reports/link/v++_link_krnl_vadd.hw_emu_guidance.html
        Steps Log File: /home/semi/workspace/exmple_of_firstP/src/_x/logs/link/link.steps.log

INFO: [v++ 60-791] Total elapsed time: 0h 3m 35s
semi@semi-Precision-T1700:~/workspace/exmple_of_firstP/src$
```

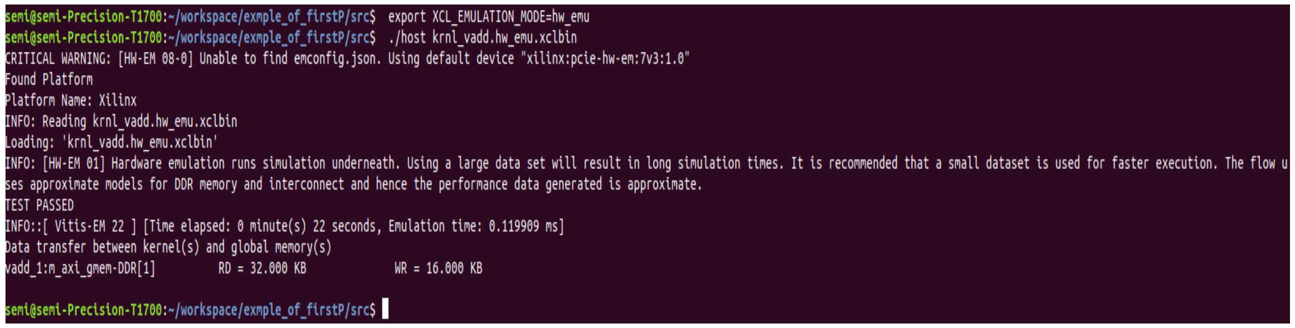
- From this result we can see that the krn_vadd.hw_emu.xclbin is created.

➔ *Run emulation hardware*

Run the emulation on hardware is the step after we had the executable file as .xclbin and .exe src to run the emulation hardware we have to follow this step bellow

in the terminal the same path bellow write those command

- emconfigutil --platform xilinx_u200_xdma_201830_2
- export XCL_EMULATION_MODE=hw_emu



```
semi@semi-Precision-T1700:~/workspace/exmple_of_firstP/src$  export XCL_EMULATION_MODE=hw_emu
semi@semi-Precision-T1700:~/workspace/exmple_of_firstP/src$  ./host krnl_vadd.hw_emu.xclbin
CRITICAL WARNING: [HW-EM 08-0] Unable to find emconfig.json. Using default device "xilinx:pcie-hw-em:7v3:1.0"
Found Platform
Platform Name: Xilinx
INFO: Reading krnl_vadd.hw_emu.xclbin
Loading: 'krnl_vadd.hw_emu.xclbin'
INFO: [HW-EM 01] Hardware emulation runs simulation underneath. Using a large data set will result in long simulation times. It is recommended that a small dataset is used for faster execution. The flow u
ses approximate models for DDR memory and interconnect and hence the performance data generated is approximate.
TEST PASSED
INFO::[ Vitis-EM 22 ] [Time elapsed: 0 minute(s) 22 seconds, Emulation time: 0.119909 ms]
Data transfer between kernel(s) and global memory(s)
vadd_1:m_axi_gmem-DDR[1]         RD = 32.000 KB             WR = 16.000 KB

semi@semi-Precision-T1700:~/workspace/exmple_of_firstP/src$ 
```

- ./hostname.exe kernelName.xclbin

After runing the emulation hardware we should have the same result lshow above.

ANNEX 5

PUBLICATION

1. Survey on Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array

Journal	International Journal of Research and Analytical Reviews (IJRAR)
Manuscript ID	Paper ID : 212763
Manuscript types	Review Paper
Date Submitted by the Author:	December 2019
Complete List of Authors:	Rakotojaona Nambinina, Haresh Suthar
Key word:	FPGA, neural network, activation function, look up table, piece wise linear approximation

2. Enhancement of Night Time Video Using Dark Channel Prior IP Accelerator

Journal	International Journal of Engineering and Advanced Technology (IJEAT)
Manuscript ID	Paper ID: D6707049420
Manuscript types	Research Paper
Date Submitted by the Author:	04 march 2020
Complete List of Authors:	Rakotojaona Nambinina, Haresh Suthar, Yogesh Parmar , Carlos Valderrama
Key word:	Image enhancement, DCP, FPGA, Vitis Xilinx software