



Helwan University

Faculty of Engineering

Communications and Electronics department

Hardware /Software Co-Design of Automotive Lane Detection

Project Documentation Year (2021-2022)

Supervised by:

Dr. Mohamed EL Dakroury

Sponsored by:

One Lab and IP Valley inc. Canada

Project team:

Abdelrahman Ahmed Fouda

Basem Diaa

Nancy Osama Ibrahim

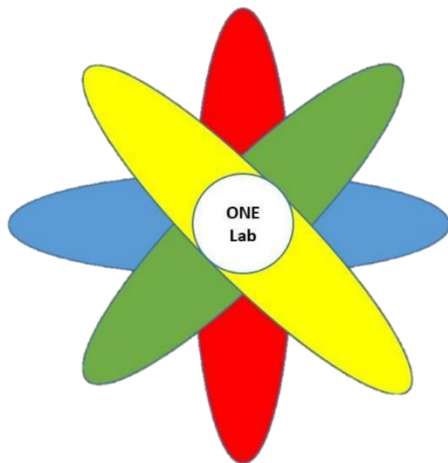
Shaimaa kamal

Sarah Muhammed Khairat

Acknowledgment

First of all, we would like to express our deepest gratitude and full thanks to almighty Allah for giving us the strength and the composure to complete our senior project, with its complexities and difficulties, it was not an easy task at all. This venture would not have been possible without the generosity of many people who contributed their time and talents to the various phases of this work. It's our proud privilege to release the feelings of our gratitude for the immense contribution of these people who directly or indirectly made this project a success. We convey our sincere gratitude to our graduation project supervisor Dr. Mohamed El Dakrouy for his advice, knowledge and the insightful discussions and suggestions throughout the whole venture, and for patiently guiding us through its processes. Also, We would like to thank Dr. Hassan Moustafa for his dedication, his efforts and his great support to our team.

Finally, we would like to thank One Lab and IP Valley inc. Canada for giving us the chance to work comfortably in its institute and provide us with the newest tools.



Abstract

Traffic accidents have become one of the most serious problems all over the world. Increase in the number of vehicles, human errors towards traffic rules and the difficulty to oversee situational dangers by drivers are contributing to most accidents on the road. With the increase in the number of vehicles, many intelligent systems have been developed to help drivers to drive safely. Lane detection is a vital element of driver assistance systems. Lane detection process is an essential component for autonomous vehicles, it has several major challenges, such as attaining robustness to inconsistencies in lighting and background clutter. Advanced Driver Assistance System (ADAS) provides safe and better driving, it ensures safety and reduces driver workload. It helps to automate, adapt and enhance the driving experience. Most of the road accidents occur due to carelessness of driver. Whenever a dangerous situation is encountered, the system either warns the driver or takes active role by performing necessary corrective action to avoid an accident.

Lane detection is an application of environmental perception, which aims to detect lane areas or lane lines by camera or lidar. In recent years, gratifying progress has been made in detection accuracy.

The lane detection algorithms, which are an advanced driver assistance system, are also one of the algorithms that using image processing techniques. Cars always need high processing speed to make accurate decisions. so we need to construct more robust detection systems and overcome the challenges and problems which causes failures and delay.

Unfortunately, the computer vision and image processing algorithms are slow on processors and High-priced hardware such as "Graphics Processing Unit" (GPU) because of the processing intensity of image processing techniques. This situation reduces the usability of image processing techniques and applications in daily life. In this project, A lane detection algorithm has been developed as an image processing project, and it has been implemented in the FPGA after this developed algorithm was broken down as software and hardware.

Key words:

Lane detection, OpenCV, xfpencv, Xilinx SDSOC platform, HLS.

Table of Contents

Acknowledgement	2
Abstract	3
Table of Contents	4
List of Figures	7
List of Tables	10
Table of Abbreviations	11
Chapter 1 Introduction	12
1.1. Introduction	13
1.2. Problem Statement	14
1.3. Objective	15
1.4. Previous work	15
1.5. proposed solution	16
Chapter 2 Development tools and environment	18
2.1. Software tools	19
2.1.1. Computer Vision Libraries	19
2.1.2. Languages	20
2.1.3. Platforms	22
2.1.2. Hardware tools	25
2.1.1. Raspberry pi	25
2.1.2. system on chip (SOC)	26
Chapter 3 Necessary Background	28
3.1. Computer vision	29
3.1.1. what is computer vision?	29
3.1.2. How Does Computer Vision Work?	29
3.1.3. History of Computer Vision	31
3.1.4. Computer vision Applications	31
3.1.5. Benefits of economic impacts of CV	311
3.2 Deep learning	32
3.2.1. What is Deep Learning?	32
3.2.2. Advantages of Computer Vision	32

3.2.3. Difference between Computer Vision and Deep Learning ..	33
3.2.4. Computer Vision VS Deep Learning	32
3.2.2. Comparison Chart.....	32
3.3. Camera Calibration and Image Distortion Removal	32
3.3. Edge detection techniques	36
3.3.1. Canny Rdge Detection.....	32
3.3.2. Hough Tranform	40
3.3.3. Perspective Transform.....	44
3.3.4. Sliding Window.....	46
Chapter 4 Implementation on Raspberry pi.....	49
4.1. Hardware used	50
4.1.1. Raspberry Pi 4	50
4.1.2. The Processor of Raspberry pi 4	50
4.2. Implementation	51
4.2.1. Without implementing the curved lanes.....	51
4.2.2. With Curved Lanes.....	53
Chapter 5 Digital Design	61
5.1. Introduction.....	62
5.2. MATLAB.....	62
5.2.1HDL Coder	62
5.3. Vivado HLS (Vivado High level synthesis)	64
5.3.1. High-Level synthesis methodology design	64
5.3.2. Vivado HLS flow	67
5.3.3. Gray scale on Vivado HLS.....	68
5.3.3. Challenges we faced	70
5.4. SDSOC (software defined System-On-Chip).....	70
5.4.1. Overview	70
5.4.2. SDSOC target	71
5.4.3. Advantages of SDSOC	71
5.4.4. SDSOC environment design flow	71
5.4.5. About the project	73
5.4.6. SDSOC Environment	73
5.5. Implementation using SDSoc	75

5.5.1. Hardware resources utilization	75
5.5.2. Power estimations.....	75
5.5.3. Hardware accelerated cycles	75
5.5.4. Implementation combinations	76
5.5.5. Comparison between RTL and SDSoC HW implementation	76
5.5.6. Sample of generated reports	76
5.6. SDSOC Build Process	77
5.7. Hardware and Software separation.....	82
5.7.1 Time Analysis.....	82
5.7.2 Design Notes	82
5.8. The Lane Detection Algorithm.....	84
5.8.1. PetaLinux	85
5.9. Zynq 7000 board.....	92
5.9.1. ZC702 Board	94
5.9.2. USB 2.0 ULPI Transceiver	97
5.9.3. SD Card Interface	99
5.9.4. USB-to-UART Bridge.....	99
Chapter 6 Results.....	106
6.1. Raspberry pi.....	107
6.2. Vivado HLS.....	112
6.4. SDSOC.....	113
Chapter 7 Conclusion and Future work.....	117
Conclusion	118
Future work.....	118
Chapter 8 References.....	119

List of Figures

Figure 1: Lane detection	13
Figure 2: Raspberry pi	25
Figure 3: SoC chip	26
Figure 4: (a) Traditional Computer Vision workflow vs. (b) Deep Learning workflow.	34
Figure 5: chessboard corners traced on a sample image	36
Figure 6: chessboard corners traced on a sample image	36
Figure 7: Original image Figure 8: Gradient Magnitude Figure 9: Non-maximum suppression	38
Figure 10: Canny: Hysteresis threshold.....	39
Figure 11: step1: After grayscale and gaussian filter, step 2: compute magnitude and angle	39
Figure 12: Step 3: Non-maximum suppression, step 4: Hysteresis thresholding.....	40
Figure 13: Canny Edge Detection Algorithm.	40
Figure 14: Mapping from edge points to the Hough Space.....	41
Figure 15: The equation to calculate a slope of a line	42
Figure 16: An alternative representation of a straight line and its corresponding Hough Space.	42
Figure 17: The process of detecting lines in an image	43
Figure 18: Output of Hough transform.....	44
Figure 19: Perspective transform 1	45
Figure 20: Perspective transform image	46
Figure 21: Example of the sliding a window approach, where we slide a window from left-to-right and top-to-bottom.	47
Figure 22: Raspberry pi 4 Model B	50
Figure 23 : Flowchart of straight lane lines algorithm	52
Figure 24 output 1 of the Raspberry pi.....	53
Figure 25 flowchart1 of curved lines algorithm	56
Figure 26: flowchart 2 of curved lines algorithm	57
Figure 27: bird's eye view	59
Figure 28: Wrapping image	59
Figure 29: Lane line boundaries warped back onto original image	60
Figure 30: Detected Lane lines overlapped on to the original image along with curvature radius and position of the car	60

Figure 31: HDL coder design on MATLAB	63
Figure 32: grayscale output.....	63
Figure 33: HLS design flow.....	65
Figure 34: HLS flow chart	66
Figure 35: flow design in Vivado HLS.....	67
Figure 36: reports generated by Vivado	68
Figure 37: original image.....	69
Figure 38: image after grayscale in Vivado.....	69
Figure 39: Xilinx blocking mail	70
Figure 40: SDSoC flow chart	72
Figure 41 projects can be implemented on SDSOC	74
Figure 42: System on chip (SoC).....	74
Figure 43: Reports generated by SDSOC	77
Figure 44: SDSoC flow.....	78
Figure 45: SDSOC Platform.....	80
Figure 46: Linking openCV and xfopenCV library.....	81
Figure 47: Linking the OpenCV library	81
Figure 48: The Executable and Linkable Format (ELF) file.....	81
Figure 49: partition errors	82
Figure 50: reports 1 by SDSOC.....	82
Figure 51: reports 2 by SDSOC.....	82
Figure 52: xfopencv libraries	83
Figure 53: OpenCV library error 1	83
Figure 54: OpenCV library error 2	83
Figure 55: OpenCV library solution	83
Figure 56: sdx log after running	84
Figure 57: Sds compiler error	84
Figure 58: opencv library error.....	84
Figure 59: command for installation.....	86
Figure 60: license of PetaLinux	86
Figure 61: add setting files to bash shell	87
Figure 62: create a normal project in PetaLinux	88
Figure 63: Create a bsp project on PetaLinux	88
Figure 64: command to show configuration.....	89
Figure 65: configuration window	89
Figure 66: steps on PetaLinux tool	90
Figure 67: Petalinux error 1	90

Figure 68: Errors solution	91
Figure 69: files generated after build the project.....	91
Figure 70: Block Diagram of ZYNQ7000.....	92
Figure 71: Block Diagram of ZC702.....	96
Figure 72: high level Block Diagram	96
Figure 73: USB 2.0 ULPI Transceiver	98
Figure 74: USB Controller Block Diagram.....	99
Figure 75: SDI/O	100
Figure 76: SD card interface.....	100
Figure 77: SD port	101
Figure 78: SW 16 in ZC702.....	102
Figure 79: SW1 in ZC702.....	102
Figure 80: UART PORT in ZC702	103
Figure 81: serial terminal setting	104
Figure 82: terminal setting in SDx.....	104
Figure 83: results after marking.....	105
Figure 84: output 1	107
Figure 85: performance of raspberry pi in case 1	107
Figure 86: performance of raspberry pi in case 1	108
Figure 87: output 2.....	109
Figure 88: performance of curvature	109
Figure 89: performance 2 of curvature	110
Figure 90: performance on pc.....	110
Figure 91: performance 2 on pc.....	111
Figure 92: Reports of grayscale code on Vivado	112
Figure 93: preprocessing stages.....	115
Figure 94: perspective transform	115
Figure 95: space usage in the system.....	116
Figure 96: power consumption result in the system	116

List of tables

Table 1: Computer Vision VS Deep Learning.....	34
Table 2: Results from Vivado.....	68
Table 3: Sample of generated reports.....	76
Table 4: Lane keeping system time analysis.....	82
Table 5: Switch SW16 configuration option setting.....	97
Table 6: Configuring the Board for SD Card Boot.....	101
Table 7: xf::Sobel resource usage.....	113
Table 8: xf::absdiff resource usage.....	113
Table 9: xf::Threshold resource usage.....	113
Table 10: xf::bitwise or resource usage.....	114
Table: 11: Xf::wrap Transform resource usage.....	114
Table 12: SDSoC ZC702 processor + hardware time analysis.....	114

Table of abbreviations

Word	Abbreviation
System on chip	SOC
Software defined system on chip	SDSOC
High level synthesis	HLS
Processing system	PS
Programmable logic	PL
LUTS	known as Lookup Table
Hardware description language	HDL
RTL	Register-transfer level
USB	Universal Serial Bus



Chapter 1

Introduction

1.1. Introduction:

Traffic accidents are mainly caused by human mistakes such as inattention, misbehavior, and distraction. Many companies and institutes have proposed methods and techniques for the improvement of driving safety and reduction of traffic accidents. Automobile accidents injure between 20 to 50 million people and kill at least 1.2 million individuals worldwide each year. Among these accidents, approximately 60% are due to driver inattentiveness and fatigue.

According to American Association of State Highway and Transportation Officials (AASHTO), almost 60% of the fatal accidents are caused by an unintentional lane drifting of a vehicle on major roads. Similarly, in a Minnesota crash study, it was reported that 25 to 50 % of the severe road departure crashes in Minnesota occur on curves, even though curves account for only 10 % of the total system mileage. Systems that predict the driver's attentive state and intent of lane change and provide map-based route guidance and/or warning about unintentional lane departure are all useful to reduce major road crashes.



Figure 1: Lane detection

Most of these crashes involve crossing of an edge line, centerline, or otherwise leaving the intended lane or trajectory. According to a recent study, which compared crashes with and without an LDWS, it was found that an in-vehicle LDWS was helpful in reducing crashes of all severities by 18%, with injuries by 24%, and with fatalities by 86% without considering for driver demographics.

1.2. Problem Statement:

Lane departure warning systems have been in development by industry for over 20 years. LDWS are generally visual devices that look at the lane line markers to compute a predicted moment of lane departure and alert the driver when unintended lane departures are about to occur without causing undue false warnings due to subtle lateral lane position changes. Beginning with simple line scan video, LDW has developed into sophisticated lane marker identification and lane boundary projection systems that provide the driver with a warning if the vehicle has a trajectory that will take it out of lane.

Among these techniques, road perception and lane marking detection play a vital role in helping drivers avoid mistakes. The lane detection is the foundation of many advanced driver assistance systems (ADASs) such as the lane departure warning system (LDWS) and the lane keeping assistance system (LKAS). Some successful ADAS or automotive enterprises, such as Mobileye, BMW, and Tesla, etc. have developed their own lane detection and lane keeping products and have obtained significant achievements in both research and real-world applications.

Almost all the current mature lane assistance products use vision-based techniques since the lane markings are painted on the road for human visual perception. The utilization of vision-based techniques detects lanes from the camera devices and prevents the driver from making unintended lane changes. Therefore, the accuracy and robustness are two most important properties for lane detection systems. Lane detection systems should have the capability to be aware of unreasonable detections and adjust the detection and tracking algorithm accordingly. When a false alarm occurs, the ADAS should alert the driver to concentrate on the driving task. On the other hand, vehicles with high levels of automation continuously monitor their environments and should be able to deal with low-accuracy detection problems by themselves. Hence, evaluation of lane detection systems becomes even more critical with increasing automation of vehicles. Most vision-based lane detection systems are commonly designed based on image processing techniques within similar frameworks. With the development of high-speed computing devices and advanced machine

learning theories such as deep learning, lane detection problems can be solved in a more efficient fashion using an end-to-end detection procedure.

The problem of our project is that the code should be written in C/C++ in order to be able to implement the algorithm on SOC (system on chip).

Some resources say that we should write it with MATLAB and then do the modifications and the optimizations process after converting the code from MATLAB into C++ (but I guess it will be hard to make this), other resources say that we should use OpenCV C++ library to implement the algorithm on the SoC, others also say that we can write the algorithm in C++ without using any library which I think will be hard.

we want to try the best and the applicable way and if there is another way which will be better than these.

1.3. Objective:

Identifying lanes on the road is a common task performed by all human drivers to ensure their vehicles are within lane constraints when driving, to make sure traffic is smooth and minimize chances of collisions with other cars in nearby lanes. Similarly, it is a critical task for an autonomous vehicle to perform. It turns out that recognizing lane markings on roads is possible using well known computer vision techniques. We will cover how to use various techniques to identify and draw the inside of a lane, compute lane curvature, and even estimate the vehicle's position relative to the center of the lane.

The Aim of the project is to compare between the normal microcontrollers and microprocessors used in embedded systems and the hardware/software system represented in implementation of the algorithm on SOC to prove that the performance of the computer vision and image processing algorithms on SOC is much better as the results is more accurate and the processing speed increases.

1.4. Previous work:

Lane detection and Lane tracking algorithms, which are a driver support system, are also one of the algorithms developed using image processing techniques. We can perform these algorithms by using many

microcontrollers such as ARM, Raspberry pi and many more. However, due to the processing intensity caused by image processing techniques, it works slowly on processors. To solve this slow working problem, high-priced hardware such as "Graphics Processing Unit" (GPU) is used. The use of GPU in projects increases the cost of the project and is offered to the end user as a product at a high price. This situation reduces the usability of image processing techniques and applications in daily life.

This project can be made using MATLAB on FPGA. We find that FPGA solution achieves the speedup of over 10 times faster than traditional CPU platform for image/video processing. We find that FPGA accelerations is a cost-efficient, high-performance video processing solution for those applications with low power and real-time requirements. However, we found that MATLAB on FPGA has low accuracy in comparison to system on chip (SOC).

1.5. proposed solution:

The utilization of vision-based techniques detects lanes from the camera devices and prevents the driver from making unintended lane changes. Therefore, the accuracy and robustness are two most important properties for lane detection systems.

Lane detection systems should have the capability to be aware of unreasonable detections and adjust the detection and tracking algorithm accordingly. These systems can prevent drivers from making mistakes on the road and can reduce traffic accidents. An effective DAS should satisfy the following requirements: accuracy, reliability, robustness, low cost, compact design, low dissipation, and applicability in real time, etc.

To solve this problem we use computer vision to try detecting the lane lines, computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos, machines can accurately identify and classify objects — and then react to what they “see.”

Computer Vision and Machine Learning are witnessing tremendous growth over the years due to the application of ML methods to computer vision tasks that include image registration, 3D reconstruction,

segmentation, and classification, motion tracking, and object detection. Difficult computational data analytics problems are solved best by ML algorithms based on training data.

The current level of computer vision allows the detection and tracking of single objects (Lanes, faces, pedestrians, cars) classes in an unconstrained setting. It enables the realization of smart cameras to identify smiling persons, pedestrian detection, surveillance applications, including image-based web searches.

The lane detection algorithms, which are an advanced driver assistance system, are also one of the algorithms that using computer vision and image processing techniques. Unfortunately, the computer vision algorithms are slow on processors because of the processing intensity of image processing techniques. This situation reduces the usability of computer vision and image processing techniques and applications in daily life. In this project, the ZC702 FPGA development card of the ZYNQ-7000 series of Xilinx Company is used as “System on Chip” (SoC) to solve this problem.



Chapter 2

Development tools and environment

2.1. Software tools

2.1.1. Computer Vision Libraries

2.1.1.1. OpenCV library

Open-Source Computer Vision (OpenCV) is an open-source image processing library. The OpenCV library was first started to be developed by INTEL in 1999, and later, with the support of various companies, it continued to be developed as open-source software under the Berkeley Software Distribution (BSD) license. Having a BSD license means that the OpenCV library can be used free of charge in any project. OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full featured CUBA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

2.1.1.2. xfOpenCV library

Xilinx OpenCV (also known as *xfOpenCV*) is a template library optimized for FPGA High-Level Synthesis (HLS), allowing to create image processing pipelines easily in the same fashion that you may do it with the well-known OpenCV library. Inside this function, you might find some common algorithms such as

- Color space conversion
- Image resizing
- Border and edge detection algorithms (Canny, Sobel)
- Warp transformation
- Hough transform
- Matrix-matrix operations (addition, weighted-addition)

Xilinx OpenCV is open source and free completely supported and maintained by the community under the BSD-3 license. That means that XfOpenCV can be used in projects by recognizing the author when distributing the application.

The Xilinx xfopenCV library is intended for application developers using Zynq-7000 SoC and Zynq® Ultra Scale+™ MPSoC and PCIE based (Virtex and U200 ...) devices. xfopenCV library has been designed to work in the SDx™ development environment and provides a software interface for computer vision functions accelerated on an FPGA device. xfopenCV library functions are mostly similar in functionality to their OpenCV equivalent. Any deviations, if present, are documented.

2.1.2. Languages

2.1.2.1. Python

Python programming Language Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

2.1.2.2. C++

C++ is a general-purpose programming language created by Danish computer scientist Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms. It was designed with an orientation toward systems programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights.

2.1.2.3. HLS

High-Level Synthesis tools (HLS) has renewed the High-Performance Computing (HPC) community's interest in Field Programmable Gate Arrays (FPGA) for accelerating HPC applications. HLS tools hide the complexity of FPGA programming through raising the abstraction level. They offer environments where traditional HPC programmers can use high-level languages such as C/C++ and OpenCL to implement kernels. The ability to program FPGAs at a high level of abstraction.

Vivado is a tool provided by AMD Xilinx, Vivado High-Level Synthesis accelerates design implementation by enabling C, C++ and System C specifications to be directly targeted into Xilinx devices without the need to manually create RTL.

We will discuss it later with more details in Chapter 5.

2.1.2.4. C

C programming Language C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions.

2.1.3. Platforms



2.1.3.1. SDSOC

SDSoC (Software-Defined System on Chip) environment is an Eclipse-based Integrated Development Environment (IDE) for implementing heterogeneous embedded systems using the Zynq-7000 All Programmable SoC (System on Chip) platform. The SDSoC system compilers (sdscc/sds++) transform C/C++ programs into complete hardware/software systems based on command line options that specify target platform, and functions within the program to compile into programmable hardware. The SDSoC system compilers generate hardware and software components that preserve program semantics and ensure synchronization between hardware and software threads, while enabling pipelined computation and communication. Each hardware function runs as an independent thread to achieve high performance with the minimum design time. We will discuss it later with more details on Chapter 5.

2.1.3.2. jupyter notebook

It is a Python (and also R) distribution. A Python distribution is a program that allows you to use Python. It may contain more than one program in it. Anaconda contains multiple programs that let you use Python. Jupiter Notebook and Spyder are two of these programs. These programs in Anaconda are specialized in data science and machine learning.

It is a web-based program under Anaconda distribution and it let you code Python. You can also call it a web application under Anaconda. It is good for data analysis and machine learning. You can visualize data easily. It is very interactive and lets you run partial codes.

2.1.3.3. Visual studio 2019

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development.

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic, .NET, C#, F#, TypeScript, XML, JavaScript, XSLT, HTML, and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. Java (and J#) were supported in the past.

2.1.3.4. VMware

VMware is a virtualization and cloud computing software provider based in Palo Alto, Calif. Founded in 1998, VMware is a subsidiary of Dell Technologies. EMC Corporation originally acquired VMware in 2004; EMC was later acquired by Dell Technologies in 2016. VMware bases its virtualization technologies on its bare-metal hypervisor ESX/ ESXi in x86 architecture.

With VMware server virtualization, a hypervisor is installed on the physical server to allow for multiple virtual machines (VMs) to run on the same physical server. Each VM can run its own operating system (OS), which means multiple OSes can run on one physical server. All the VMs on the same physical server share resources, such as networking and RAM. In 2019, VMware added support to its hypervisor to run containerized workloads in a Kubernetes cluster in a similar way. These types of workloads can be managed by the infrastructure team in the same way as virtual machines and the DevOps teams can deploy containers as they were used to.

2.1.3.5. Virtual box

When using a traditional you need to install the operating system on a physical machine for evaluating software that cannot be installed on your current operating system. Oracle VirtualBox is what you need in this case, instead of reinstalling software on your physical machine. VirtualBox is designed to run virtual machines on your physical machine without reinstalling your OS that is running on a physical machine. One more

VirtualBox advantage is that this product can be installed for free. A virtual machine (VM) works much like a physical one. An OS and applications installed inside a VM “think” that they are running on a regular physical machine since emulated hardware is used for running VMs on VirtualBox. Virtual machines are isolated from each other and from the host operating system. Thus, you can perform your tests in isolated virtual machines without any concerns of damaging your host operating system or other virtual machines.

2.1.3.6. Peta Linux

Peta Linux is a tool used to create your personal system which is compatible with hardware you work on it.

The Peta Linux Tools help you create and deliver a custom Linux distribution. They allow you to work easily with available software which is independently available from the Xilinx GIT or open source communities.

The Peta Linux Tools offers everything necessary to customize, build and deploy Embedded Linux solutions on Xilinx processing systems. Tailored to accelerate design productivity, the solution works with the Xilinx hardware design tools to ease the development of Linux systems for Zynq® UltraScale+™ MPSoC, Zynq®-7000 SoCs, and MicroBlaze.

Custom BSP Generation Tools

Peta Linux tools enable developers to synchronize the software platform with the hardware design as it gains new features and devices.

Peta Linux consists of three key elements: pre-configured binary bootable images, fully customizable Linux for the Xilinx device, and Peta Linux SDK which includes tools and utilities to automate complex tasks across configuration, build, and deployment. We will discuss it with more details in chapter 5.

2.2. Hardware tools

2.2.1. Raspberry pi



Figure 2: Raspberry pi

From building a single board computer for educational purposes and personal entertainment to selling more than 40 million boards around the globe, Raspberry Pi has come a long way. Raspberry Pi devices are developed by a UK-based charity that aims to deliver the power of digital computing to people across all sections of the world. Raspberry Pi foundations empower low-cost and high-power single-board PCs and software.

Most of the schools and colleges prefer Raspberry Pi units for general purposes. However, Raspberry Pi was not intended as a charity program earlier. It was a small team of the computer laboratory at the University of Cambridge that discovered a declining interest in computers due to increasing costs and tough maintenance of typical computer systems. This is where they decided to get a solution to this problem and thus, Raspberry Pi was born. Let's discuss the journey of Raspberry Pi from 2012 until now.

The Raspberry Pi- a credit card sized single board computer developed by Raspberry Pi Foundation, United Kingdom. The board is a miniature marvel, packs extreme computing power and capable to develop amazing projects. The computer costs ranging from \$5 to \$35 and is perfect to perform all sort of computing tasks and interfacing various sorts of devices via GPIO.

The Raspberry Pi board contains Broadcom based ARM Processor, Graphics Chip, RAM, GPIO and other connectors for external devices. The

operating procedure of Raspberry Pi is very similar as compared to PC and requires additional hardware like Keyboard, Mouse, Display Unit, Power Supply, SD Card with OS Installed (Acting like Hard Disk) for operation. Raspberry Pi also facilitates USB ports, Ethernet for Internet/Network-Peer to Peer Connectivity. Like any other computer, where Operating system acts as backbone for operation. Raspberry Pi facilitates open source operating system's based on Linux. Till date more than 30 operating systems based on different flavors of Linux is being launched. Raspberry Pi foundation has also launched various accessories like Camera, Gertboard and Compute Model Kit for deploying add-on hardware modules.

2.2.2. System on chip(SOC)

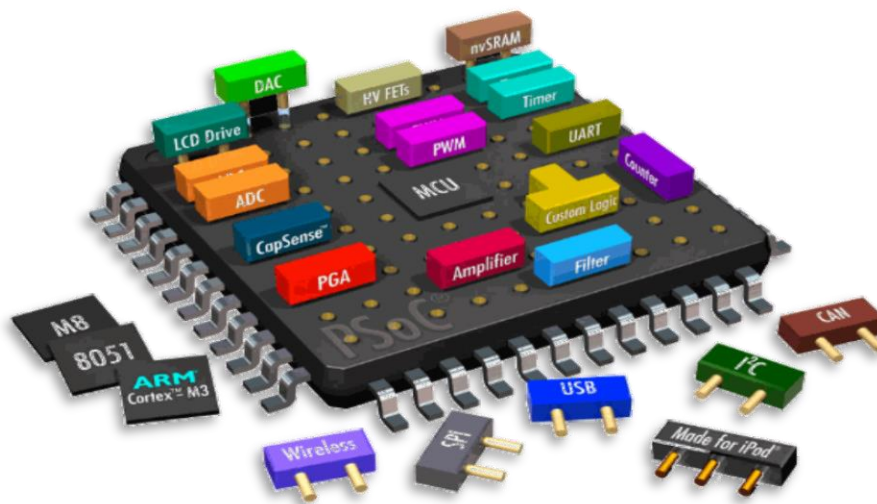


Figure 3: SoC chip

As you may be aware, the concept has been around for a while; the implication is that a single silicon chip can be used to implement the functionality of an entire system, rather than several different physical chips being required. In the past, the term SoC has usually referred to an Application Specific Integrated Circuit (ASIC), which can include digital, analogue and radio frequency components, together with mixed signal blocks for implementing analogue-to-digital and digital-to-analogue converters (ADCs and DACs). Focusing on the digital aspect for a moment, an SoC can combine all aspects of a digital system: processing,

high-speed logic, interfacing, memory, and so on. All of these functions might otherwise be realized using physically separate devices, and combined together into a system at the Printed Circuit Board (PCB) level. The SoC solution is lower cost, enables faster and more secure data transfers between the various system elements, has higher overall system speed, lower power consumption, smaller physical size, and better reliability. In fact, there are a number of compelling reasons for preferring SoCs over discrete component equivalent systems! For a simple graphical comparison of the system-on-a-board and the system-on-chip

A **SoC**, also is essentially an integrated circuit or an IC that takes a single platform and integrates an entire electronic or computer system onto it. It is, exactly as its name suggests, an entire system on a single chip. The components that an SoC generally looks to incorporate within itself include a central processing unit, input and output ports, internal memory, as well as analog input and output blocks among other things. Depending on the kind of system that has been reduced to the size of a chip, it can perform a variety of functions including signal processing, wireless communication, artificial intelligence and more.



Chapter 3

Necessary Background

3.1. Computer vision

3.1.1. What is Computer Vision?

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos, and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image.

Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

Computer vision is used in industries ranging from energy and utilities to manufacturing and automotive – and the market is continuing to grow. It is expected to reach USD 48.6 billion by 2022.

3.1.2. How Does Computer Vision Work?

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be

fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN).

Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will “look” at the data and teach itself to tell one

image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.

A CNN helps a machine learning or deep learning model “look” by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is “seeing.” The neural network runs convolutions and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

Much like a human making out an image at a distance, a CNN first discerns hard edges and simple shapes, then fills in information as it runs iterations of its predictions. A CNN is used to understand single images. A recurrent neural network (RNN) is used in a similar way for video applications to help computers understand how pictures in a series of frames are related to one another.

3.1.3. History Of Computer Vision

1974 saw the introduction of optical character recognition (OCR) technology, which could recognize text printed in any font or typeface. Similarly, intelligent character recognition (ICR) could decipher hand-written text using neural networks. Since then, OCR and ICR have found their way into document and invoice processing, vehicle plate recognition, mobile payments, machine translation and other common applications.

In 1982, neuroscientist David Marr established that vision works hierarchically and introduced algorithms for machines to detect edges, corners, curves and similar basic shapes. Concurrently, computer scientist Kunihiko Fukushima developed a network of cells that could recognize patterns. The network, called the Neocognitron, included convolutional layers in a neural network.

By 2000, the focus of study was on object recognition, and by 2001, the first real-time face recognition applications appeared. Standardization of how visual data sets are tagged and annotated emerged through the 2000s. In 2010, the ImageNet data set became available. It contained millions of tagged images across a thousand object classes and provides a foundation for CNNs and deep learning models used today. In 2012, a team from the University of Toronto entered a CNN into an image recognition contest.

The model, called AlexNet, significantly reduced the error rate for image recognition. After this breakthrough, error rates have fallen to just a few percent.

3.1.4. Computer vision Applications

Now we have a very large number of applications that we use computer vision in, and here some examples of fields and applications:

- Medicine
- Military
- Autonomous vehicles (like self-driving cars)
- Industry
- Sports

Each track and field of this you can use computer vision in many things for a lot of applications.

3.1.5. Benefits and economic impacts of CV

- It makes the computer (machine) act like human, can see and take an action control or monitor.
- Classify and identify the objects.
- In any application, it makes our life easier.
- Increasing human safety in dangerous situations.

3.2. Deep learning

3.2.1. What is Deep Learning?

Deep learning is a subset of machine learning and AI based on artificial neural networks that seeks to mimic the functioning of the human brain so that computer would learn what comes naturally to humans. Deep learning is concerned with algorithms inspired by the structure of the human brain that enables machines to gain some level of understanding and knowledge just the way human brain filters information. It defines model parameters for decision making process mimicking the understanding process in the human brain. It is a way of data inference in machine learning and together, they are among the major tools of modern AI. It was initially developed as a machine learning approach to deal with complex input-output mappings. Today, deep learning is a state of the art system used across many industries for various applications.

3.2.2. Advantages of Deep Learning

Rapid progressions in DL and improvements in device capabilities including computing power, memory capacity, power consumption, image sensor resolution, and optics have improved the performance and cost-effectiveness of further quickened the spread of vision-based applications. Compared to traditional CV techniques, DL enables CV engineers to achieve greater accuracy in tasks such as image classification, semantic segmentation, object detection and Simultaneous Localization and Mapping (SLAM). Since neural networks used in DL are trained rather than programmed, applications using this approach often require less expert analysis and fine-tuning and exploit the tremendous amount of video data available in today's systems. DL also provides superior flexibility because CNN models and frameworks can be re-trained using a custom dataset for any use case, contrary to CV algorithms, which tend to be more domain-specific.

3.2.3. Difference between Computer Vision and Deep Learning

Concept

Computer vision is a subset of machine learning that deals with making computers or machines understand human actions, behaviors, and languages similarly to humans. The idea is to get machines to understand and interpret the visual world so that they make sense out of it and derive some meaningful insights. Deep learning is a subset of AI that seeks to mimic the functioning of the human brain based on artificial neural networks.

Purpose

The purpose of computer vision is to program a computer to interpret visual information contained within image and video data in order to make better sense of the digital data. The idea is to translate this data into meaningful insights, using contextual information provided by humans in order to make better business decisions and solve complex problems. Deep learning has been introduced with the objective of moving machine learning closer to AI. DL algorithms have revolutionized the way we deal with data. The goal is to extract features from raw data based on the notion of artificial neural networks.

3.2.4. Computer Vision vs. Deep Learning

The traditional approach is to use well-established CV techniques such as feature descriptors for object detection. Before the emergence of DL, a step called feature extraction was carried out for tasks such as image classification. Features are small “interesting”, descriptive, or informative patches in images. Several CV algorithms, such as edge detection, corner detection or threshold segmentation may be involved in this step. As many features as practicable are extracted from images and these features form a definition (known as a bag-of-words) of each object class. At the deployment stage, these definitions are searched for in other images. If a significant number of features from one bag-of-words are in another image, the image is classified as containing that specific object (i.e. chair, horse, etc.). The difficulty with this traditional approach is that it is necessary to choose which features are important in each given image. As the number

of classes to classify increases, feature extraction becomes more and more cumbersome.

It is up to the CV engineer's judgment and a long trial and error process to decide which features best describe different classes of objects. Moreover, each feature definition requires dealing with a plethora of parameters, all of which must be fine-tuned by the CV engineer. DL introduced the concept of end-to-end learning where the machine is just given a dataset of images which have been annotated with what classes of object are present in each image. Thereby a DL model is 'trained' on the given data, where neural networks discover the underlying patterns in classes of images and automatically works out the most descriptive and salient features with respect to each specific class of object for each object. It has been well-established that DNNs perform far better than traditional algorithms, albeit with trade-offs with respect to computing requirements and training time. With all the state-of-the-art approaches in CV employing this methodology, the workflow of the CV engineer has changed dramatically where the knowledge and expertise in extracting hand-crafted features has been replaced by knowledge and expertise in iterating through deep learning architectures as depicted in the development of CNNs has had a tremendous influence in the field of CV in recent years and is responsible for a big jump in the ability to recognize objects

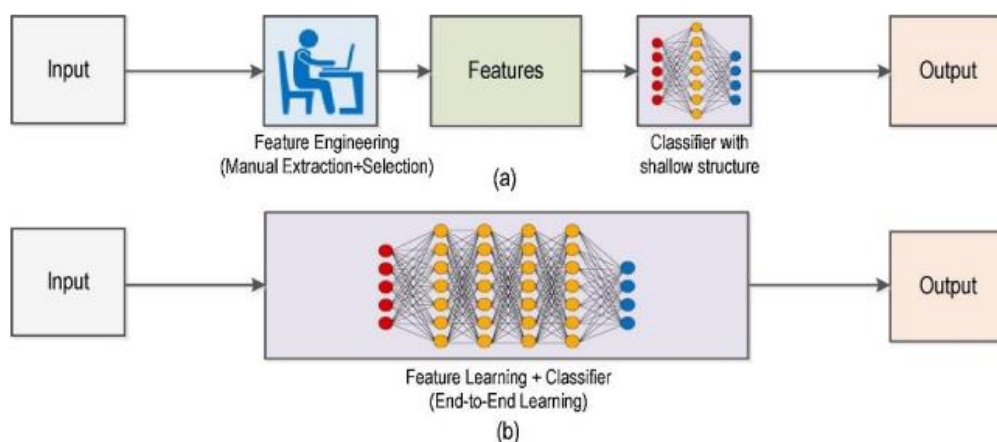



Figure 4: (a) Traditional Computer Vision workflow vs. (b) Deep Learning workflow.

3.2.5. Comparison Chart

Table 1: Computer Vision VS Deep Learning

Computer Vision	Deep Learning
It is a subset of machine learning that enables computers to process, analyze and interpret the visual world.	It is a subset of AI that seeks to mimic the functioning of the human brain based on artificial neural networks.
The purpose is to program a computer to interpret visual information contained within image and video data and derive meaningful insights.	The goal is to enable machines to gain some level of understanding and knowledge just the way human brain filters information.
Applications include defect detection, image labeling, face recognition, object detection, image classification, and more.	Applications include self driving cars, natural language processing, visual recognition, image and speech recognition, virtual assistants, etc.



3.3. Camera Calibration & Image Distortion Removal

Image distortion occurs when a camera looks at 3D objects in the real world and transforms them into a 2D image

This transformation isn't always perfect and distortion can result in a change in apparent size, shape or position of an object, So we need to correct this distortion to give the camera an accurate view of the image.

This is done by computing a camera calibration matrix by taking several chessboard pictures of a camera and using `cv2.calibrateCamera()`

function.

To compute the camera, the transformation matrix and distortion coefficients, we use multiple pictures of a *chessboard* on a flat surface

taken by the same camera. OpenCV has a convenient method called `findChessboardCorners` that will identify the points where black and white squares intersect and reverse engineer the distortion matrix this way. We use camera calibration techniques when using camera to take our input image to apply our algorithm on it.



Figure 5: chessboard corners traced on a sample image

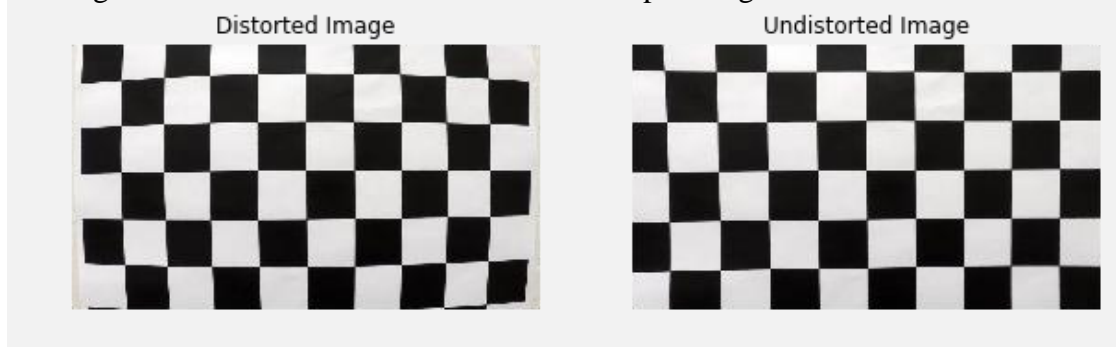


Figure 6: chessboard corners traced on a sample image

3.4. Edge detection techniques

The edge representation of an image significantly reduces the quantity of data to be processed, yet it retains essential information regarding the shapes of objects in the scene. This explanation of an image is easy to incorporate into a large amount of object recognition algorithms used in computer vision along with other image processing applications. The major property of the edge detection technique is its ability to extract the exact edge line with good orientation as well as more literature about edge detection has been available in the past three decades. On the other hand, there is not yet any common performance directory to judge the performance of the edge detection techniques. The performance of an edge

detection techniques are always judged personally and separately dependent to its application.

Edge detection is a fundamental tool for image segmentation. Edge detection methods transform original images into edge images benefits from the changes of grey tones in the image. In image processing especially in computer vision, the edge detection treats the localization of important variations of a gray level image and the detection of the physical and geometrical properties of objects of the scene. It is a fundamental process detects and outlines of an object and boundaries among objects and the background in the image. Edge detection is the most familiar approach for detecting significant discontinuities in intensity values. Edges are local changes in the image intensity. Edges typically occur on the boundary between two regions. The main features can be extracted from the edges of an image. Edge detection has major feature for image analysis. These features are used by advanced computer vision algorithms. Edge detection is used for object detection which serves various applications like medical image processing, biometrics etc. Edge detection is an active area of research as it facilitates higher level image analysis. There are three different types of discontinuities in the grey level like point, line, and edges. Spatial masks can be used to detect all the three types of discontinuities in an image. There are many edge detection techniques in the literature for image segmentation. The most commonly used discontinuity-based edge detection techniques are reviewed in this section. Those techniques are Roberts edge detection, Sobel Edge Detection, Prewitt edge detection, Kirsh edge detection, Robinson edge detection, Marr-Hildreth edge detection, LoG edge detection and Canny Edge Detection.

3.4.1. Canny Edge Detection

In industry, the Canny edge detection technique is one of the standard edge detection techniques. It was first created by John Canny for his Master's thesis at MIT in 1983, and still outperforms many of the newer algorithms that have been developed. To find edges by separating noise from the image before find edges of image the Canny is a very important method. Canny method is a better method without disturbing the features of the

edges in the image afterwards it , applying the tendency to find the edges and the serious value for threshold.

The algorithmic steps are as follows:

- Convolve image $f(r, c)$ with a Gaussian function to get smooth image $f^{\wedge}(r, c)$. $f^{\wedge}(r,c)=f(r,c)*G(r,c,6)$
- Apply first difference gradient operator to compute edge strength then edge magnitude and direction are obtained as before.
- Apply non-maximal or critical suppression to the gradient magnitude.
- Apply threshold to the non-maximal suppression image. Unlike Roberts and Sobel, the Canny operation is not very susceptible to noise. If the Canny detector worked well it would be superior.

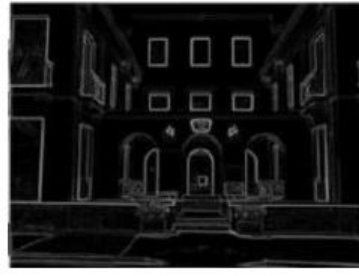


Figure 7: Original image Figure 8: Gradient Magnitude Figure 9: Non-maximum suppression

Then, A threshold is applied on the edge pixels, by so-called hysteresis thresholding, which is based on using two thresholds, $T1$ and $T2$, with $T1 < T2$. Ridge pixels with values greater than $T2$ are said to be "strong" edge pixels. Ridge pixels with values between $T1$ and $T2$ are said to be "weak" edge pixels.

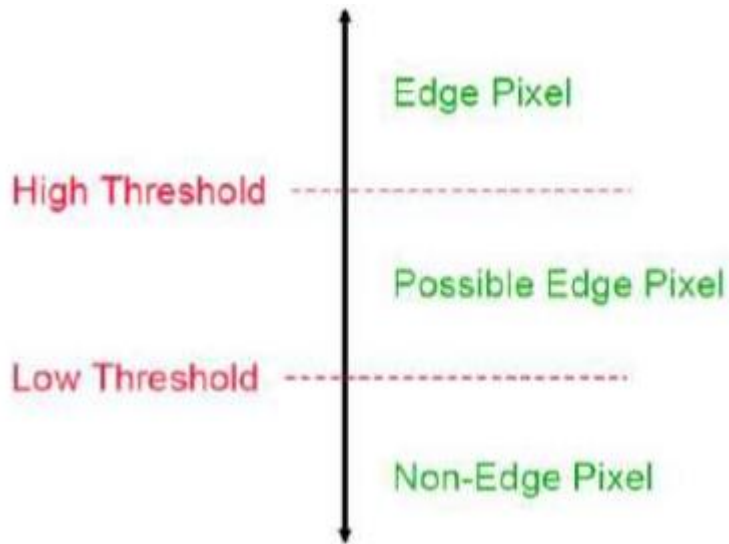


Figure 10: Canny: Hysteresis threshold

Algorithm of canny edge:

1. convert pic to gray scale.
2. Noise reduction using Gaussian filter.
3. Compute gradient magnitude and angle
4. Non-maximum suppression
5. Hysteresis thresholding.



Figure 11: step1: After grayscale and gaussian filter, step 2: compute magnitude and angle



Figure 12: Step 3: Non-maximum suppression, step 4: Hysteresis thresholding

3.4.2. Hough Transform

The Hough Transform is an algorithm patented by Paul V. C. Hough and was originally invented to recognize complex lines in photographs (Hough, 1962). Since its inception, the algorithm has been modified and enhanced to be able to recognize other shapes such as circles and quadrilaterals of specific types. In order to understand how the Hough Transform algorithm works, it is important to understand four concepts: edge image, the Hough Space, and the mapping of edge points onto the Hough Space, an alternate way to represent a line, and how lines are detected.

Edge Image



Figure 13: Canny Edge Detection Algorithm.

An edge image is the output of an edge detection algorithm. An edge detection algorithm detects edges in an image by determining where the brightness/intensity of an image changes drastically. Examples of edge detection algorithms include: Canny, Sobel, Laplacian, etc. It is common for an edge image to be binarized meaning all of its pixel values are either a 1 or a 0. Depending on your situation, either a 1 or a 0 can signify an edge pixel. For the Hough Transform algorithm, it is crucial to perform edge detection first to produce an edge image which will then be used as input into the algorithm.

The Hough Space and the Mapping of Edge Points onto the Hough Space

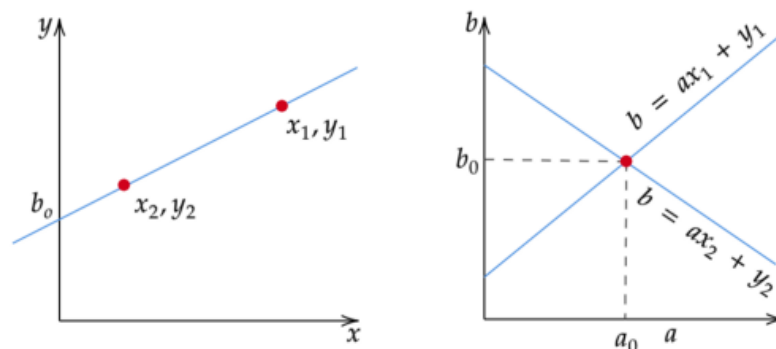


Figure 14: Mapping from edge points to the Hough Space.

The Hough Space is a 2D plane that has a horizontal axis representing the slope and the vertical axis representing the intercept of a line on the edge image. A line on an edge image is represented in the form of $y = ax + b$ (Hough, 1962). One line on the edge image produces a point on the Hough Space since a line is characterized by its slope a and intercept b . On the other hand, an edge point (x_i, y_i) on the edge image can have an infinite number of lines pass through it. Therefore, an edge point produces a line in the Hough Space in the form of $b = ax_i + y_i$ (Leavers, 1992). In the Hough Transform algorithm, the Hough Space is used to determine whether a line exists in the edge image.

An Alternate Way to Represent a Line

$$m = \frac{\text{rise}}{\text{run}} = \frac{y_2 - y_1}{x_2 - x_1}$$

m = slope

(x_1, y_1) = first point

(x_2, y_2) = second point

Figure 15: The equation to calculate a slope of a line

There is one flaw with representing lines in the form of $y = ax + b$ and the Hough Space with the slope and intercept. In this form, the algorithm won't be able to detect vertical lines because the slope a is undefined/infinity for vertical lines (Leavers, 1992). Programmatically, this means that a computer would need an infinite amount of memory to represent all possible values of a . To avoid this issue, a straight line is instead represented by a line called the normal line that passes through the origin and perpendicular to that straight line. The form of the normal line is $\rho = x \cos(\theta) + y \sin(\theta)$ where ρ is the length of the normal line and θ is the angle between the normal line and the x axis.

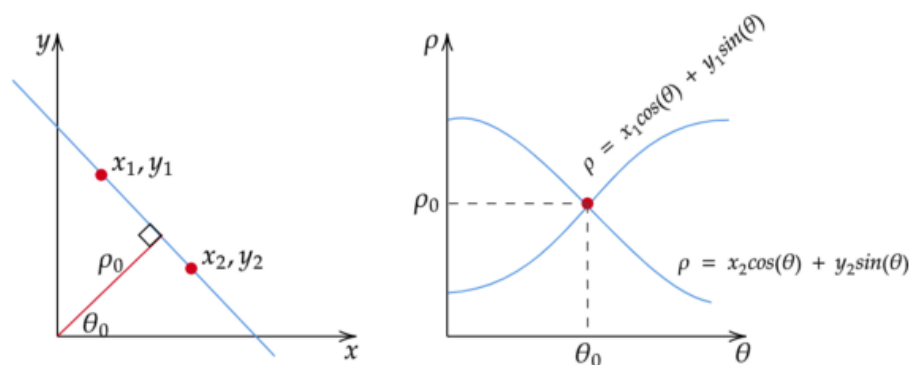


Figure 16: An alternative representation of a straight line and its corresponding Hough Space.

Using this, instead of representing the Hough Space with the slope a and intercept b , it is now represented with ρ and θ where the horizontal axis is for the θ values and the vertical axis are for the ρ values. The mapping of

edge points onto the Hough Space works in a similar manner except that an edge point (x_i, y_i) now generates a cosine curve in the Hough Space instead of a straight line (Leavers, 1992). This normal representation of a line eliminates the issue of the unbounded value of a that arises when dealing with vertical lines.

Line Detection

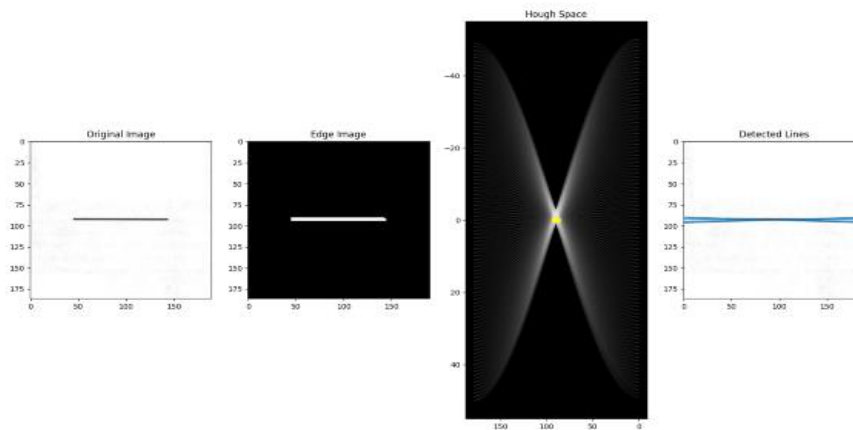


Figure 17: The process of detecting lines in an image

The yellow dots in the Hough Space indicate that lines exist and are represented by the θ and ρ pairs.

As mentioned, an edge point produces a cosine curve in the Hough Space. From this, if we were to map all the edge points from an edge image onto the Hough Space, it will generate a lot of cosine curves. If two edge points lay on the same line, their corresponding cosine curves will intersect each other on a specific (ρ, θ) pair. Thus, the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that have a number of intersections larger than a certain threshold. It is worth noting that this method of thresholding might not always yield the best result without doing some preprocessing like neighborhood suppression on the Hough Space to remove similar lines in the edge image.

Steps of The Algorithm

1. Decide on the range of ρ and θ . Often, the range of θ is $[0, 180]$ degrees and ρ is $[-d, d]$ where d is the length of the edge image's diagonal. It is important to quantize the range of ρ and θ meaning there should be a finite number of possible values.

2. Create a 2D array called the accumulator representing the Hough Space with dimension (num_rhos, num_thetas) and initialize all its values to zero.
3. Perform edge detection on the original image. This can be done with any edge detection algorithm of your choice.
4. For every pixel on the edge image, check whether the pixel is an edge pixel. If it is an edge pixel, loop through all possible values of θ , calculate the corresponding ρ , find the θ and ρ index in the accumulator, and increment the accumulator base on those index pairs.
5. Loop through all the values in the accumulator. If the value is larger than a certain threshold, get the ρ and θ index, get the value of ρ and θ from the index pair which can then be converted back to the form of $y = ax + b$.

```
<matplotlib.image.AxesImage at 0x1a9e161f888>
```

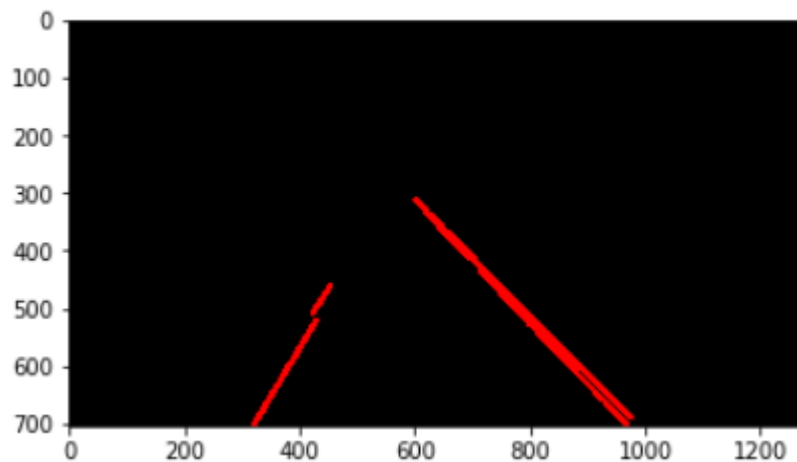


Figure 18: Output of Hough transform

3.4.3. Perspective transform

When human eyes see near things, they look bigger as compared to those who are far away. This is called perspective in a general way. Whereas transformation is the transfer of an object etc. from one state to another.

So overall, the perspective transformation deals with the conversion of 3d world into 2d image. The same principle on which human vision works and the same principle on which the camera works.

We will see in detail about why this happens, that those objects which are near to you look bigger, while those who are far away, look smaller even though they look bigger when you reach them.

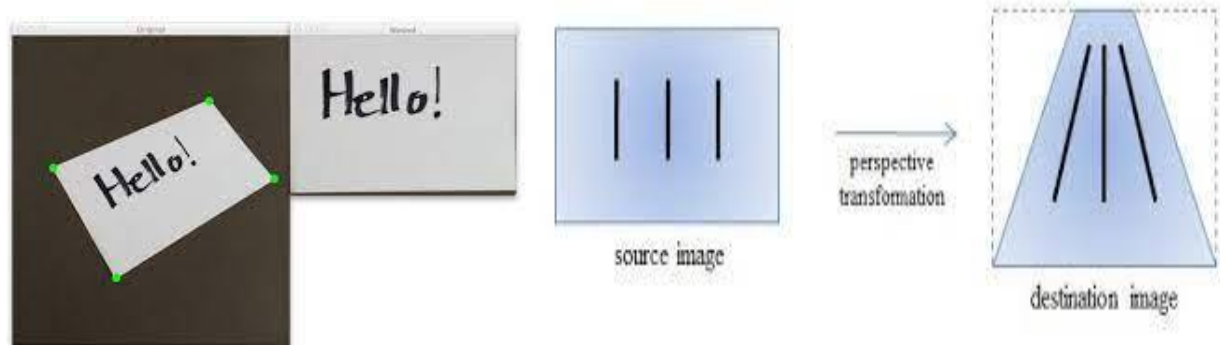


Figure 19: Perspective transform 1

- Sometimes we have the required images or videos but the necessary information is not properly visible. In such cases, we may have to change the alignment of the images or videos or change the perspective of the images or videos to obtain better insights into the required information from the image or video.
- Then we make use of a function called Perspective Transform() function in OpenCV.
- The PerspectiveTransform() function takes the coordinate points on the source image which is to be transformed as required and the coordinate points on the destination image that corresponds to the points on the source image as the input parameters.

```
cv2.PerspectiveTransform(source coordinates,  
destination_coordinates)
```

- The PerspectiveTransform() function returns the matrix of the transformed perspective as the output.
- Then the original source image and the resulting matrix from the PerspectiveTransform() function along with the size of the required output image is passed to the warpPerspective() function to obtain the transformed image.

```
cv2.warpPerspective(source_image, destination_image,  
destination_imagesize)
```

- The aligned or transformed image of the source image as per the required size is returned by the warpPerspective() function.



Figure 20: Perspective transform image

3.4.4. Sliding window

Sliding windows play an integral role in object classification, as they allow us to localize exactly “*where*” in an image an object resides. We utilize a sliding window to detect objects in images at various scales and locations.

Normally, we’d use an image classifier on the window region to see if it contains an object of interest. By combining sliding windows with object classification, we can build classification systems for images that can identify objects of various sizes and positions. Despite their simplicity, these techniques are the foundation of modern neural network architectures for identifying objects in images.

what is a sliding window?

In the context of computer vision (and as the name suggests), a sliding window is a rectangular region of fixed width and height that “slides” across an image, such as in the following figure:

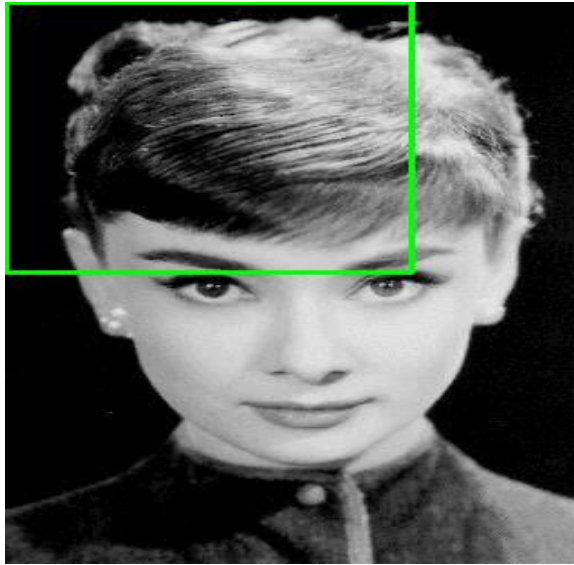


Figure 21: Example of the sliding a window approach, where we slide a window from left-to-right and top-to-bottom.

For each of these windows, we would normally take the window region and apply an *image classifier* to determine if the window has an object that interests us — in this case, a face.

How the sliding window on the lane works?

The selection of initial sliding window is very important and hence previous lane starting points are saved in the cache to efficiently detect the moving lanes. Basic sliding window approach is not suitable in sharp curves and dashed lanes. Multiple sliding window technique is able to detect the sharp curves and dashed lanes. Once the left and right lanes are detected, the center of the left lane and right lane is found; vehicle's deviation from actual center of the lane is calculated.

After, edge detection, perspective transformation is performed in order to get a top view of the image. The image coordinates are shifted, and the image is rotated followed by projecting the image onto a 2D plane. Most of the research using feature-based techniques for lane detection use inverse perspective transform. It is very useful especially when dealing with curved road scenarios. It is also called bird's-eye view image. Lane detection algorithm comprises of the basic image preprocessing steps followed by image warping and line fitting using B-Spline model and Random Sample Consensus (RANSAC) algorithm. After warping the

image, to detect the best lane points sliding window approach is used instead of searching the entire image. Sliding window technique can be used for any application to reduce the search area there by reducing the computational complexity. This technique also reduces detecting points outside the lanes. The lane detection using combined thresholding (Gradient and Hue, Saturation and Lightness (HSL) thresholding), perspective transform, sliding window approach and second order polynomial fitting.

A sliding window technique is used to find the edges of rail lines. The edge points are detected by filtering the points in sliding windows based on their gradient value and mean square error approximation. A sliding window technique is used to detect lanes on the road. Generally, in a sliding window approach, after image preprocessing steps, initially two windows (left and right) are considered using peak values in the histogram of the image. Subsequent sliding window positions are determined based on mean of the points in the current sliding window. Several open-source resources implemented sliding window technique. But this algorithm works only for slightly curved lanes but not for sharp curves because, the windows are considered assuming that the next lane points will be above the previous window.



Chapter 4

Implementation on Raspberry pi

4.1. Hardware used

4.1.1. Raspberry Pi 4

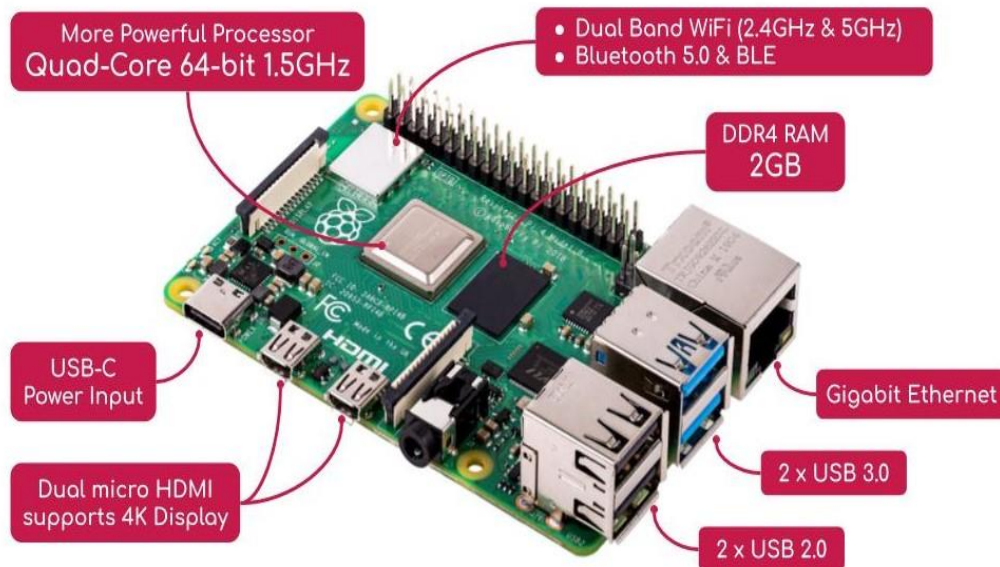


Figure 22: Raspberry pi 4 Model B

Raspberry Pi 4 was launched in June 2019. This edition has a 64-bit CPU, Wi-Fi, and Bluetooth, and boasts improved performance speed. It has been specifically designed for experimentation with the circuit board and to help users develop new computing skills. The main reason to choose raspberry pi 4 as a beginner is because this is the latest (till now) bug-free version and faster than all old pi boards.

Raspberry Pi 4 comes with- Broadcom BCM2711 Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz Processor, **4GB LPDDR4-3200 SDRAM** and OpenGL ES 3.0 graphics H.265 (4kp60 decode). It has Gigabit Ethernet, along with Wi-Fi of 5.0 GHz IEEE 802.11ac wireless & Bluetooth 5.0, BLE. Micro-SD card slot for loading operating system and data storage. 2-lane MIPI DSI display port and 2-lane MIPI CSI camera port. And also 4-pole stereo audio and composite video port.

4.1.2. The Processor of Raspberry pi 4

The encapsulated processor, which uses the same heat spreader for better thermal control as the last model, may look the same from the outside. But while the Raspberry Pi 3 was built around the Broadcom BCM2837 processor, a quad-core Arm Cortex-A53 clocked at 1.4GHz, the new board

is built around the Broadcom BCM2711, a 64-bit quad-core Arm Cortex-A72 clocked at 1.5GHz. Though that might not seem significant, there are some big differences between the core architectures of these two processors.

While the A53 was designed as a mid-range core, and for efficiency, the A72 is a performance core, so despite the apparently small difference in clock speed, the real performance difference between the cores is really rather significant.

We use Raspberry Pi 4 Model B to implement the algorithm and use Motors to Move the wheels of Demo of the project.

4.2. Implementation

4.2.1. Without implementing the curved lanes

First we take the input from the camera, the camera takes some pictures of a chess board before taking a picture of the lane in order to know the camera distortion and trying to correct it.

This camera distortion results from the transformation of the image from 3D to 2D and results in change in the shape or position of an object in an image.

After calibrating the distortion in the image, we apply this error to raw images to apply distortion correction.

After this we take the input from the camera (video) and convert the image from 3 channel image (R, G, B) to grayscale.

We then filter the image from noises and unneeded information by applying gaussian filter, after this we use canny edge detector in order to detect the edges in our image then the output of canny is the input of ROI step. This step specifies the region which we want to detect the lane by adjusting the parameters and the coordinates to specify the area of the lane only to complete the processing on the image.

We try multiple times until the correct region of the lane is specified and no error exist in this region.

We then apply the output image to the Hough transform which draw the lines on the lanes of the image in order to detect them, then we average these lines to output only one line on the image.

We then apply these lines (the two lanes) on the original image which is 3 Channel image to draw the lines on the original lanes as shown in the following figure.

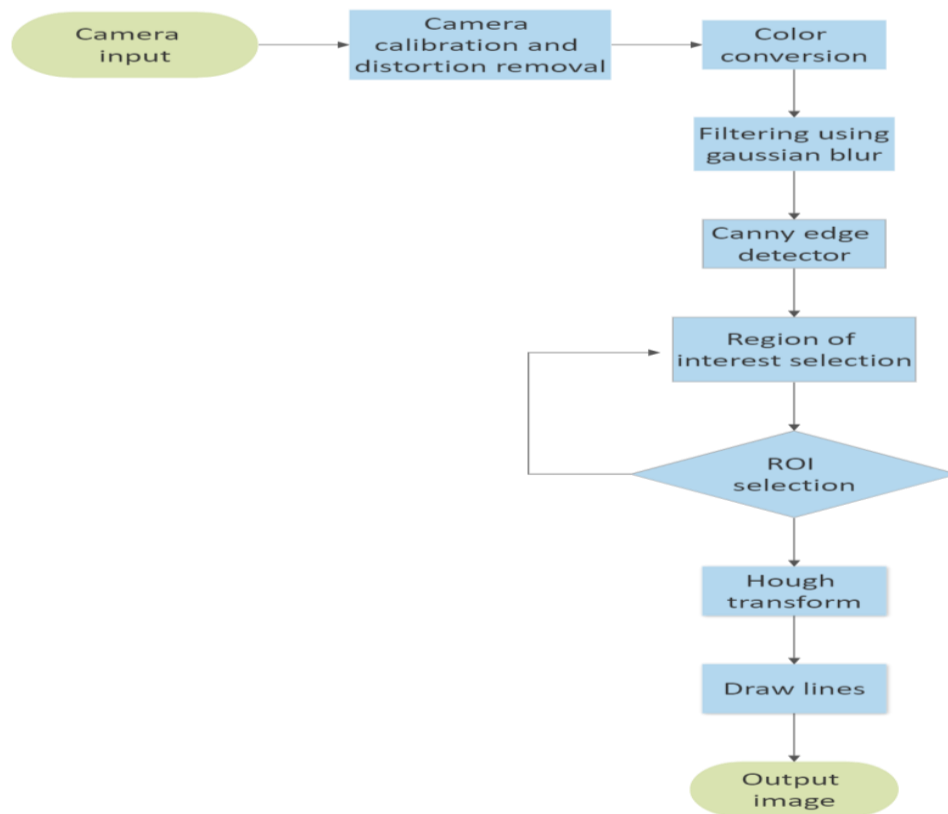


Figure 23 : Flowchart of straight lane lines algorithm

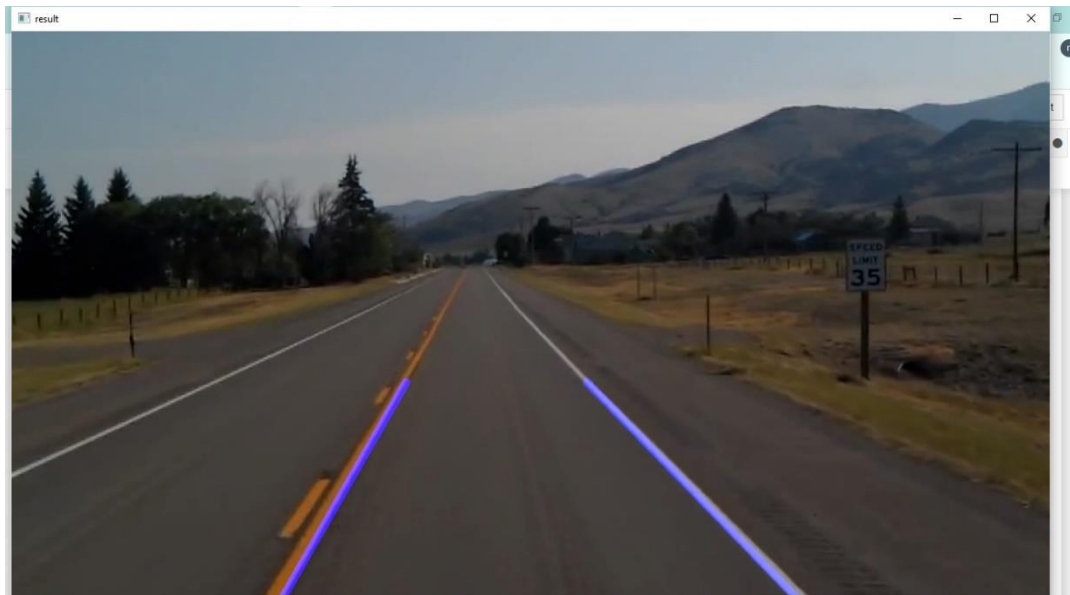


Figure 24 output 1 of the Raspberry pi

4.2.2. With Curved Lanes

Identifying lanes on the road is a common task performed by all human drivers to ensure their vehicles are within lane constraints when driving, so as to make sure traffic is smooth and minimize chances of collisions with other cars in nearby lanes. Similarly, it is a critical task for an autonomous vehicle to perform. It turns out that recognizing lane markings on roads is possible using well known computer vision techniques

The input data set for the algorithm is collected by recording video sequences of the white lanes, which are formed inside an indoor facility. The input video is converted to frames. The input image consists of white lanes. Hence, detecting those white lanes using color thresholding helps in eliminating most of the unnecessary points in the frame. The input RGB image is converted to hue, saturation and value (HSV) image. The lower and upper thresholds are selected depending upon the lighting conditions, which vary the color of the lanes. Thus, white components are extracted. Now, the image is converted to gray scale image. Gray scale image conversion reduces the computational complexity. In real time environment, small stones or debris that can be seen beside lanes, could be accidentally considered as edge points. This can be reduced using Gaussian Blur.

Next, canny edge detection is performed to segment the dominant pixels. Edge detection finds the boundaries of the objects within images. Canny Edge detection is a multi-stage algorithm. A filtered gray scale image is the input to the canny edge detection. It deals with the calculation of gradients, non- maximum suppression and thresholding with hysteresis. There are two thresholds, which should be selected based on the input data set. If the pixel gradient value is higher than the upper threshold, the pixel is accepted as an edge. If the pixel value is below the lower threshold and is not connected to any of the pixel values above the upper threshold, it is rejected.

When the lane is curved, top view image produces more information about the lane than a normal image whose lanes intersect at horizon. With this view, the left and right lanes are not closer to each other at the horizon. This also helps in the sliding window technique in the categorization of left and right lanes. If the image is not perspective transformed, there is a good probability that the sliding windows of the left lane might be accidentally merged into the windows of the right lane. The image coordinates are shifted, rotated and projected to a bird's- eye view image using perspective transform. The coordinates of the perspective transformation are chosen carefully so that a perfect transformed image is obtained. This helps us retain all the lane data in the image.

After canny edge detection, the image warping is done. This warped canny image is the input to the sliding window technique. To draw multiple sliding windows, the starting point of the windows should be known. To find the initial point, histogram for the bottom part of the image is calculated. Based on the peak value of the histogram, the initial window is selected and the mean of the non-zero points inside the window is determined. For the first half of the image, left lane peak is obtained and the other right half gives the peak of the right lane. Thus, left and right starting sliding windows are formed, and then left lane center and right lane center are calculated. This kind of selection works fine as long as both lanes are in ideal places that is left lane on the left side of the image and right lane on the right side of the image. In some cases, for example where the vehicle is gradually steered more towards right, then we might see the right lane in the left half of the image. In such situations, improper detection is possible.

To avoid such situations, a variable cache is defined to save the starting

point windows of previous lanes. The histogram is not calculated throughout the detection process but only for first few frames and later it will be dynamically tracked using the cache. For each initial sliding window, mean of the points inside each window is calculated. Two windows to the left and right of the mean point and three more windows on top of the mean point are selected as next sliding windows. This kind of selection of windows helps to detect the sharp curves and dashed lanes. The window width and height are fixed depending upon the input data set. The width of the sliding window should be adjusted depending on the distance between the lanes. The left and right sliding windows on top help track the lane points turning left and right respectively. The windows need to have a relatively well-tuned size to make sure the left and right curved lanes are not tracked interchangeably when lanes have sharp turn and become horizontally parallel to each other. The detected points inside the sliding window are saved.

The process of finding the mean point and next set of sliding windows based on valid points inside the respective sliding windows for left and right lanes is continued until no new lane points are detected. Points detected in previous sliding windows are discarded when finding points in next set of sliding windows. This makes sure that the algorithm can stop tracking when no new points are found. Once, the left and right points are detected, these points are processed to polynomial fitting to fit the respective lanes. Average of polynomial fit values of past few frames is used to avoid any intermittent frames, which may have unreliable lane information. The lane starting points are retrieved from the polynomial fitting equation. This approach helps increase the confidence of lane's starting point detection based on lanes rather than relying on starting sliding windows. The deviation of the car from center of the lanes is calculated. Then the image is unwrapped, and the lines are fitted onto the input image.

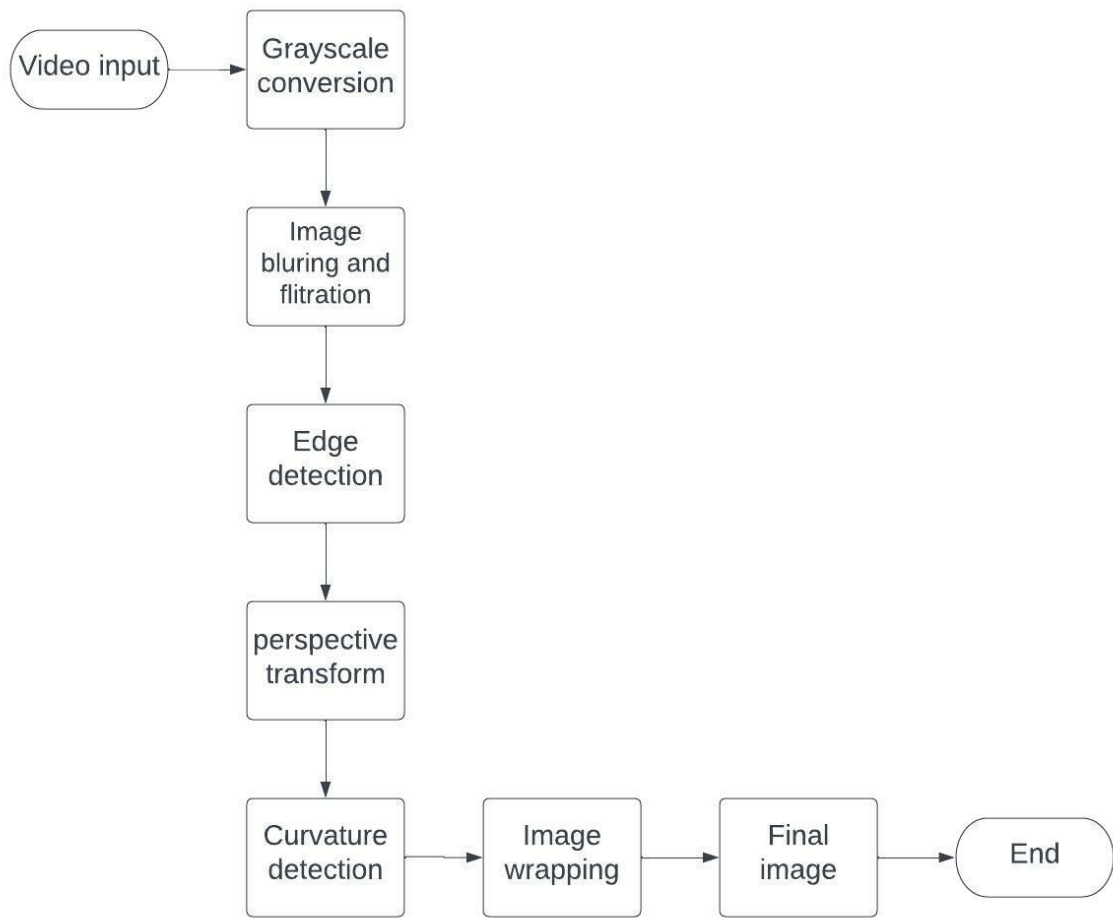


Figure 25 flowchart1 of curved lines algorithm

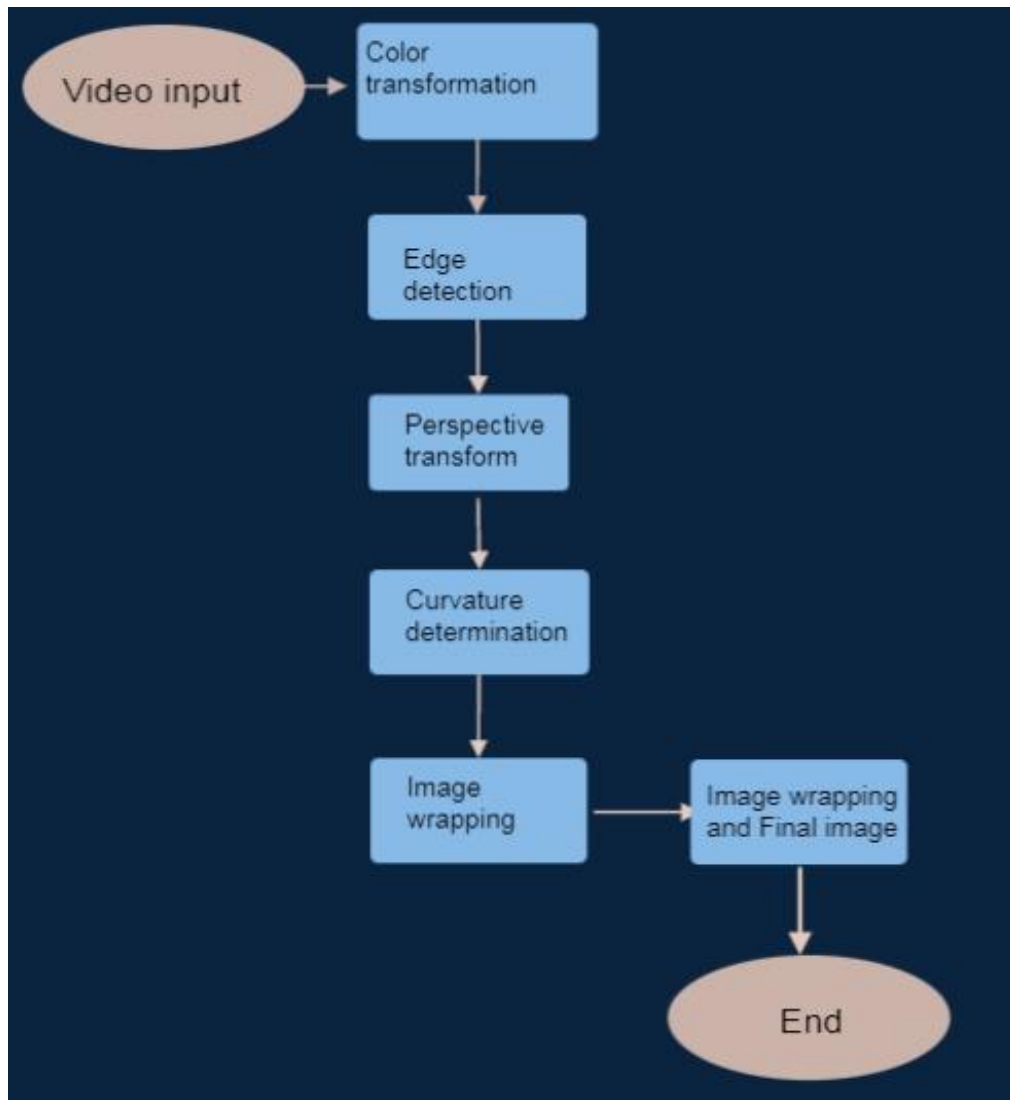


Figure 26: flowchart 2 of curved lines algorithm

we build a pipeline consisting of the following steps:

- Computation of camera calibration matrix and distortion coefficients from a set of chessboard images
- Distortion removal on images
- Application of color and gradient thresholds to focus on lane lines
- Production of a bird's eye view image via perspective transform
- Use of sliding windows to find *hot* lane line pixels
- Fitting of second-degree polynomials to identify left and right lines composing the lane.
- Computation of lane curvature and deviation from lane center

- Warping and drawing of lane boundaries on image as well as lane curvature information.

4.2.2.1. Video or image input

Here we take a video to be our input and to do our image processing and computer vision techniques on it.

4.2.2.2. Use color transforms

We use color transforms, gradients, etc., to create a thresholded binary image. The idea behind this step is to create an image processing pipeline where the lane lines can be clearly identified by the algorithm. There are a number of different ways to get to the solution by playing around with different gradients, thresholds and color spaces. I experimented with a number of these techniques on several different images and used a combination of thresholds, color spaces, and gradients. I settled on the following combination to create my image processing pipeline: S channel thresholds in the HLS color space and V channel thresholds in the HSV color space, along with gradients to detect lane lines. An example of a final binary thresholded image is shown in the fig below, where the lane lines are clearly visible.

4.2.2.3. Apply a perspective transform

We apply perspective transform to generate a “bird’s-eye view” of the image. Images have perspective which causes lanes lines in an image to appear like they are converging at a distance even though they are parallel to each other. It is easier to detect curvature of lane lines when this perspective is removed. This can be achieved by transforming the image to a 2D Bird’s eye view where the lane lines are always parallel to each other. Since we are only interested in the lane lines, I selected four points on the original un-distorted image and transformed the perspective to a Bird’s eye view as shown in below.

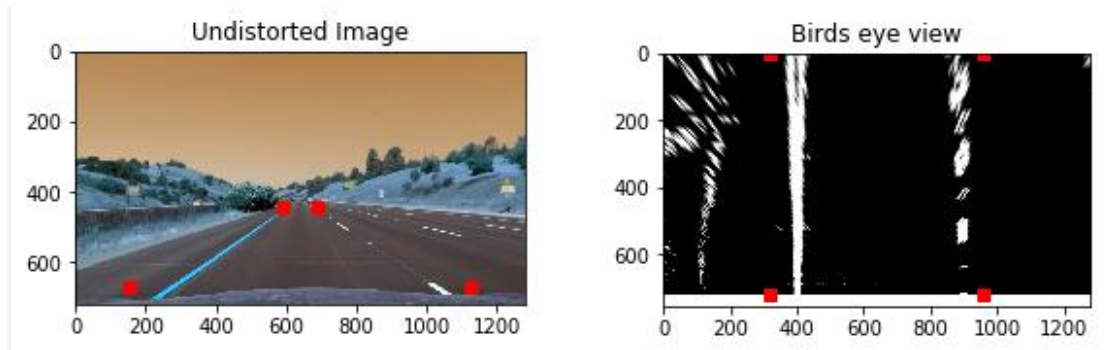


Figure 27: bird's eye view

4.2.2.4. Detect lane pixels

To detect the lane lines and fit to find the lane boundary, there are a number of different approaches. I used convolution which is the sum of the product of two separate signals: the window template and the vertical slice of the pixel image.

I used a sliding window method to apply the convolution, which will maximize the number of hot pixels in each window. The window template is slid across the image from left to right and any overlapping values are summed together, creating the convolved signal. The peak of the convolved signal is where the highest overlap of pixels is and it is the most likely position for the lane marker. Methods have been used to identify lane line pixels in the rectified binary image. The left and right lines have been identified and fit with a curved polynomial function. Example images with line pixels identified with the sliding window approach and a polynomial fit overlapped are shown.

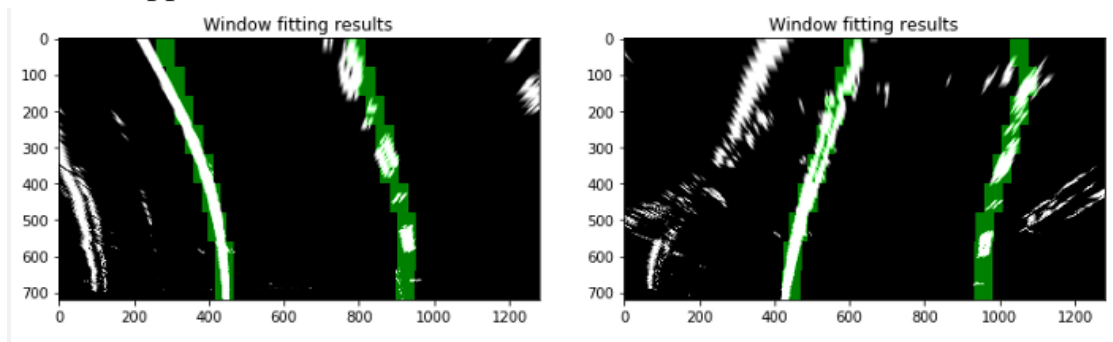


Figure 28: Wrapping image

4.2.2.5. The curvature determination

To determine the curvature of the lane and vehicle position with respect to the center of the car I took the measurements of where the lane lines are

and estimated how much the road is curving, along with the vehicle position with respect to the center of the lane. I assumed that the camera is mounted at the center of the car.

4.2.2.6. Image wrapping

In this step we warp the detected lane boundaries back onto the original image and display numerical estimation of lane curvature and vehicle position.

The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. Fig 7 demonstrates that the lane boundaries were correctly identified and warped back on to the original image. An example image with lanes, curvature, and position from center is shown.

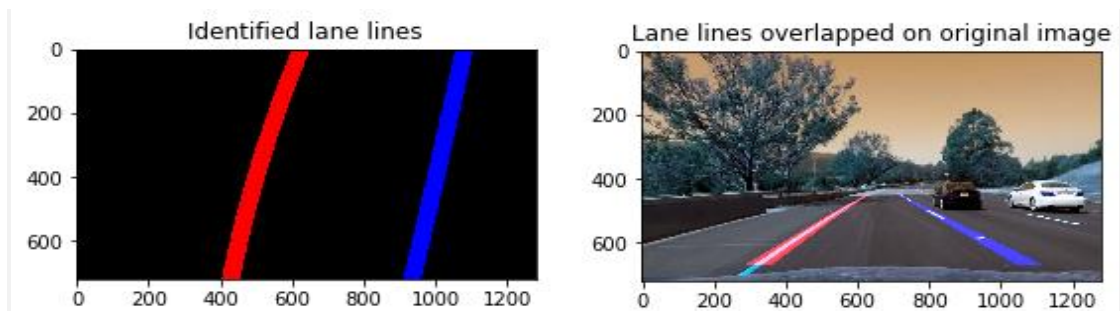


Figure 29: Lane line boundaries warped back onto original image

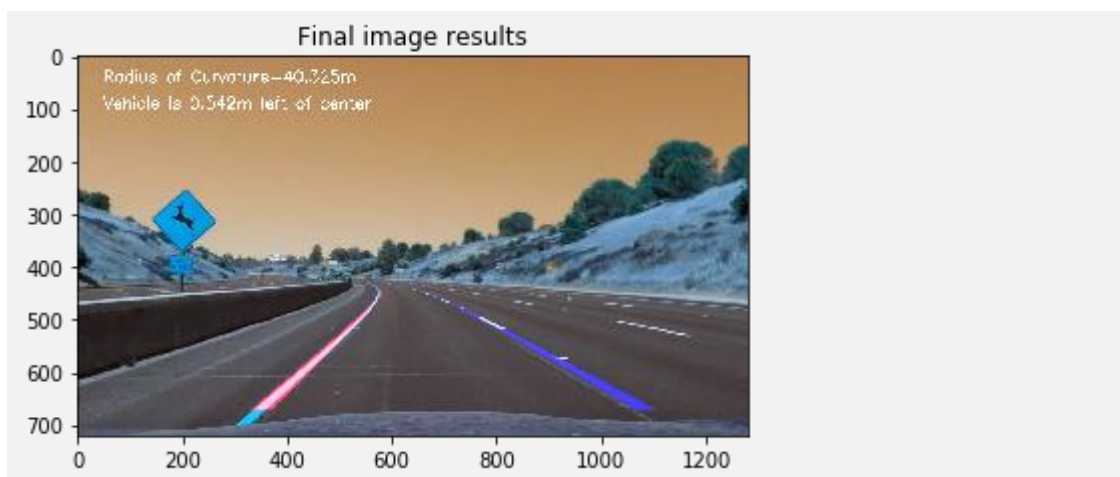
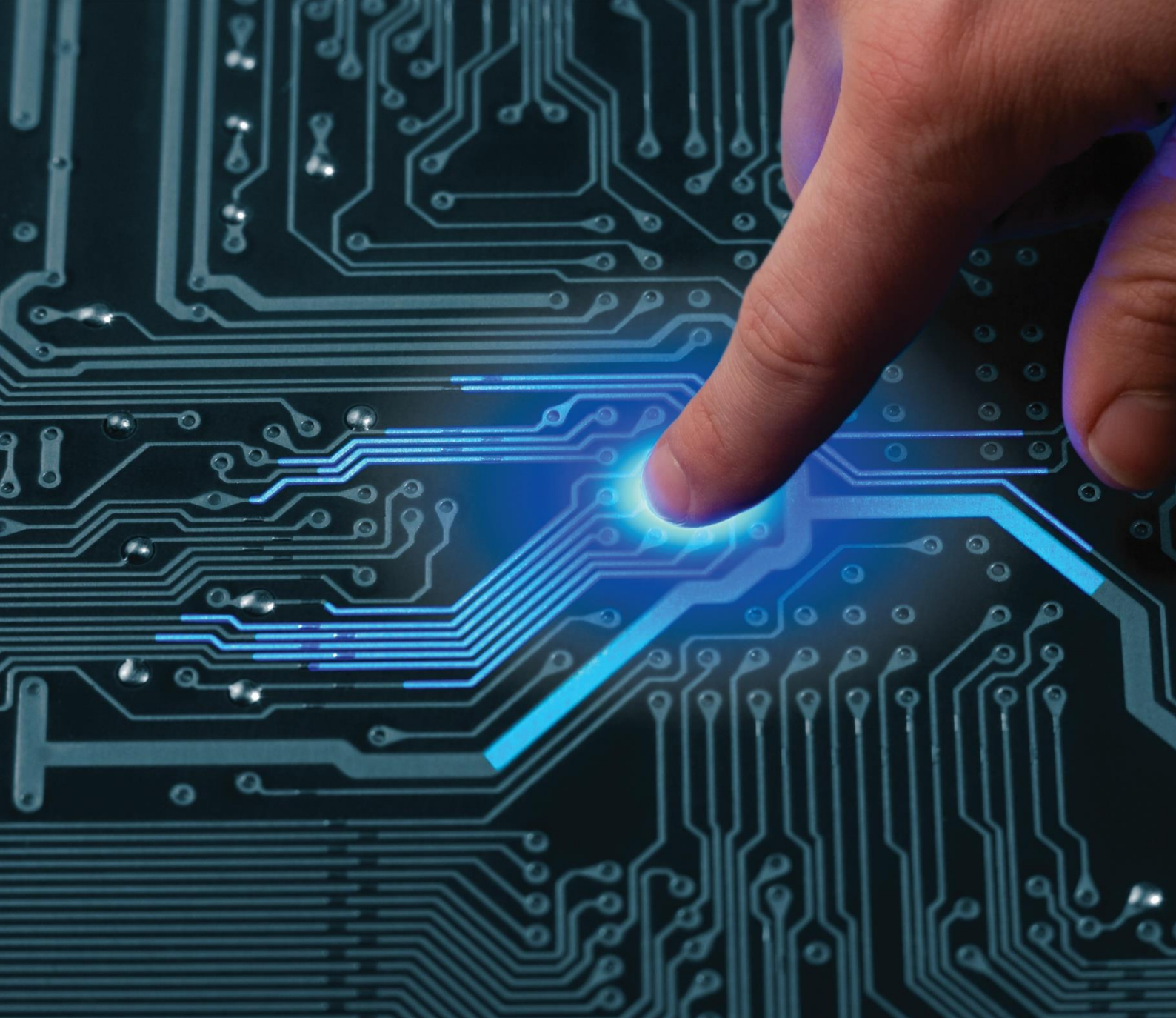


Figure 30: Detected Lane lines overlapped on to the original image along with curvature radius and position of the car



Chapter 5

Digital Design

5.1. Introduction

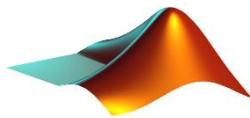
The usage rate of image processing algorithms has gained a great impetus with the "OpenCV" library, which is implemented as open source. The lane detection algorithms, which are an advanced driver assistance system, are also one of the algorithms that using image processing techniques. Unfortunately, the image processing algorithms are slow on processors because of the processing intensity of image processing techniques. This situation reduces the usability of image processing techniques and applications in daily life.

As FPGA solution achieves the speedup of over 10 times faster & less power than traditional CPU platform for image/video processing but it take much time to design any circuit (ex: using a research paper published in Germany in lane detection acceleration VHDL this design take 3 years, this IC gave better performance but it take long time compared with 4 to six months using GPU), so try design high performance IC in the shortest time. First, we decided to use MATLAB & SIMULINK to convert the lane detection algorithm to HDL code (VHDL / Verilog) but after some research we found 2 problems.

- 1-The low performance of the output HDL code.
- 2- The output HDL code isn't readable so we can't modify it.
- 3-The accuracy of the output of MATLAB is poor.

We then decided to change the MATLAB to a new tool which gives accurate output. This tool is SDSOC (software defined system on chip).

5.2. MATLAB



First we desire to use HDL coder to the lane detection algorithm to HDL (VHDL/Verilog)

5.1.1. HDL Coder

HDL Coder provides a workflow advisor that automates the programming of Xilinx®, Microsemi®, and Intel® FPGAs. You can control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between your Simulink model and the generated Verilog and VHDL code,

enabling code verification for high-integrity applications adhering to DO-254 and other standards.

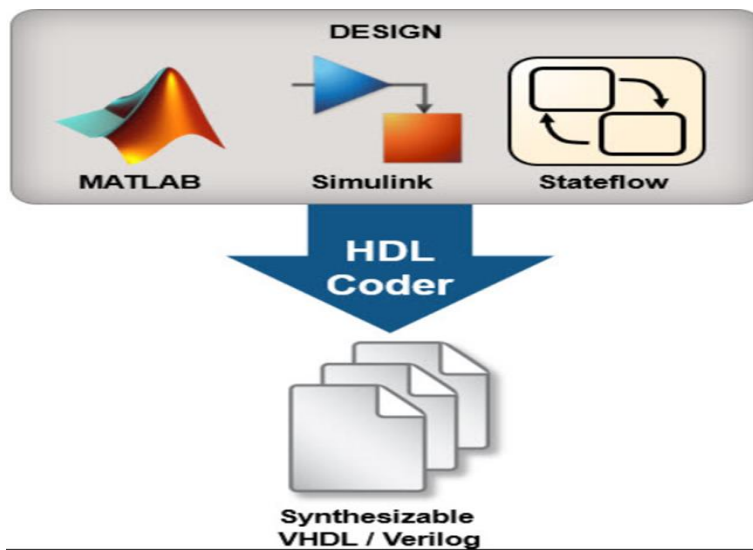


Figure 31: HDL coder design on MATLAB

DO-254 is a requirements-driven process-oriented safety standard used on commercial electronics that go into aircraft. (Conceptually speaking, this standard applies to all electronics in anything that flies or could crash and pose a hazard to the public.

After working about one month with MATLAB we started in applying some parts of the algorithm as grayscale.



Figure 32: grayscale output

But after research and some experiments we discover the low performance of the HDL coder, need long time to make the time analysis for the code and the main problem is the code is not readable so, if we can't modify any part in the algorithm to decrease of the area or for any purpose so the sponsor suggest using the SDSOC but due to the lack of resources and the long time it need to build a project (the small project can take half an hour to built) so, we used Vivado HLS as the two tools work with the same HLS and it can build project in few minutes.

5.3. Vivado HLS (Vivado High level synthesis)

5.3.1. High-Level synthesis methodology design:

Vivado High-Level Synthesis also known as Electronic System Level Synthesis and C Synthesis is an automated design procedure, that converts the algorithmic description of a system into the corresponding hardware circuit. In this process, which is actually a part of the high-level design flow, the system behavior is described at a very high level of abstraction. This method improves productivity and reduces the chance of error.

Synopsys introduced Behavioral Compiler, the first-generation behavioral synthesis tool, in 1994. Verilog was used as the input language. 10 years later, various next-generation High-Level Synthesis (HLS) tools were introduced in the market. These tools offered circuit synthesis, described in a high-level language and Register Transfer Level. Manufacturers of these tools provided extensive PC support for a wide range of tool issues.

The first step in HLS is to implement the system algorithm in a high-level language, such as ANSI C, C++, System C, etc. After that, the synthesis tool generates the technical details, which is required for hardware implementation.

Most of the HLS design methods use conventional logic synthesis tools by generating a Register Level Transfer (RTL) logic implementation from the system algorithm. The RTL logic is used by the traditional logic synthesis tools to generate a gate-level design. The HLS tools convert the partially

timed functional code into a fully timed RTL design. The basic objective of HLS is to enable the designers to develop and test the hardware efficiently. It also gives the designers better control over the design architecture optimization.

Hardware design could be developed at multiple levels of abstraction. The most common abstraction levels are Algorithmic Level, Register Transfer Level and Gate Level.

It allows the users who are not familiar with logic design to develop hardware accelerators for complex ML algorithms on FPGA. Xilinx included Vivado HLS on their development framework so that any user can use HLS for custom hardware development. Although HLS enables rapid prototyping for any random algorithms, there exists limitations on its achievable performance, memory bandwidth, and logic count compared to the ones from manual designs by domain experts.

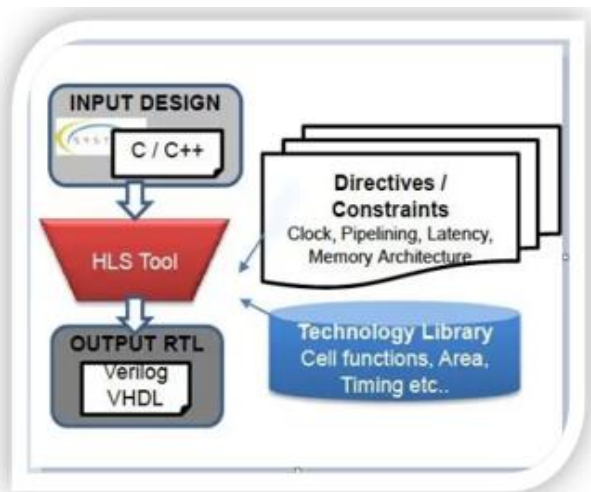


Figure 33: HLS design flow

For ML applications, which usually have regular compute and memory access patterns, HLS provides a decent performance, short development time, and easier debugging environment. The objective of HLS is to extract parallelism from the input description and construct a micro architecture that is faster and cheaper than simply executing the input description as a program on a microprocessor. The goal of HLS is to let hardware designers efficiently build and verify hardware, by giving them better control over optimization of their design architecture, and through the nature of allowing the designer to describe the design at a higher level of abstraction

while the tool does the RTL implementation. Verification of the RTL is an important part of the process. The high-level synthesis tools handle the micro-architecture and transform untimed or partially timed functional code into fully timed RTL implementations, automatically creating cycle-by-cycle detail for hardware implementation. The (RTL) implementations are then used directly in a conventional logic synthesis flow to create a gate-level implementation.

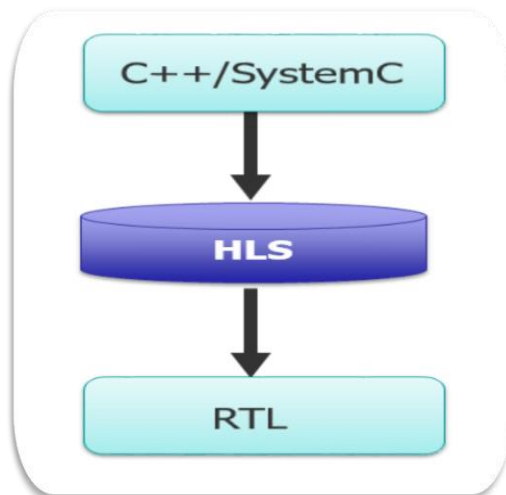


Figure 34: HLS flow chart

The HLS design description is ‘high level’ compared to RTL in two aspects: design abstraction, and specification language:

I. High level of abstraction: HLS input is an untimed (or partially timed) dataflow or computation specification of the design. This is higher level than RTL because it does not describe a specific cycle by cycle behavior and allows HLS tools the freedom to decide what to do in each clock cycle.

II. High level specification language: HLS input is specified in languages like C, C++, System C, or even MATLAB, and allows use of advanced language features like loops, arrays, structs, classes, pointers, inheritance, overloading, template, polymorphism, etc. This is higher level than (synthesizable subset of) RTL description languages and allows concise, reusable, and readable design descriptions.

The objective of HLS is to extract parallelism from the input description and construct a micro architecture that is faster and cheaper than simply executing the input description as a program on a microprocessor. The

micro architecture contains a pipelined data path and a cycle-by-cycle description of how data is routed through this data path.

5.3.2. Vivado HLS flow :

1- we use C/C++ but with changes in the syntax as there are some functions and libraries in C/C++ is not supported in HLS (can't be accelerated or have some design issues as dynamic access memory).

2- We build the project then after building it, Vivado generates RTL scheme (VHDL/Verilog).

3- we use Vivado AXI to produce IP core.

4- finally the project implementation is completed.

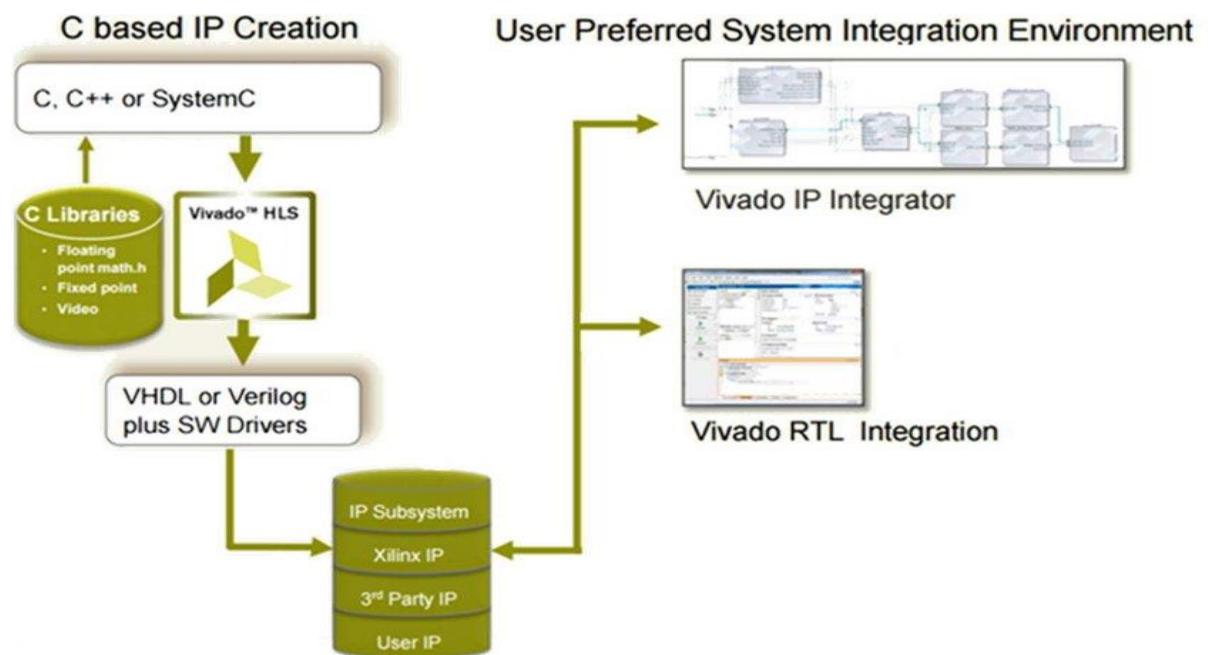


Figure 35: flow design in Vivado HLS

We chose to use Vivado HLS before using SDSOC due to the presence of resources and shorter build time.

5.3.3. Gray scale on Vivado HLS

We apply the Grayscale algorithm on HLS. We put an input image which contains 3 channels (R, G, B) to the code and it converts the image to an image with one channel (grayscale). The Vivado HLS tool outputs reports which explain the performance of the

The reports

The screenshot displays two sections of a Vivado report. The first section, 'General Information', lists project details: Date (Mon Jul 4 00:40:19 2022), Version (2018.3), Project (gradu), Solution (solution1), Product family (zynq), and Target device (xc7z020clg484-1). The second section, 'Performance Estimates', includes a 'Timing (ns)' summary table and a 'Latency (clock cycles)' summary table.

General Information

Date: Mon Jul 4 00:40:19 2022
 Version: 2018.3 (Build 2405991 on Thu Dec 06 23:56:15 MST 2018)
 Project: gradu
 Solution: solution1
 Product family: zynq
 Target device: xc7z020clg484-1

Performance Estimates

▣ **Timing (ns)**

▣ **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.283	1.25

▣ **Latency (clock cycles)**

▣ **Summary**

Latency		Interval		
min	max	min	max	Type
1	1	1	1	none

Figure 36: reports generated by Vivado

Table 2: Results from Vivado

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	83
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	15
Register	-	-	10	-
Total	0	1	10	98
Available	280	220	106400	53200
Utilization (%)	0	~0	~0	~0

The output of the test bench



Figure 37: original image



Figure 38: image after grayscale in Vivado

5.3.4. Challenges we faced

One of the Challenges we faced in our project while using Vivado HLS is making Egypt a blocked area by Xilinx so

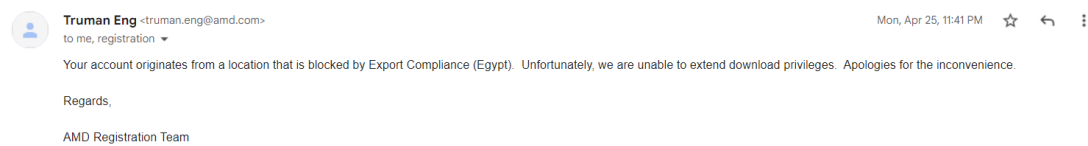


Figure 39: Xilinx blocking mail

the absence of mark function of Vivado Which allows you to choose some functions to build on the ARM cortex of the SOC so we go to use

5.4. SDSOC (software defined System-On-Chip)

5.4.1. Overview

SDSOC is an environment offered by AMD Xilinx is an Eclipse-based Integrated Development Environment (IDE) for implementing heterogeneous embedded systems using the Zynq-7000 SoC and Zynq UltraScale+ MPSoC platforms which support the design using HLS (high level synthesis c/c++).

The SDSoC development environment provides a familiar embedded C/C++/OpenCL application development experience including an easy to use Eclipse IDE and a comprehensive design environment for heterogeneous Zynq SoC and MPSoC deployment. Complete with the industry's first C/C++/OpenCL full-system optimizing compiler, SDSoC delivers system level profiling, automated software acceleration in programmable logic, automated system connectivity generation, and libraries to speed programming. It also enables end user and third party platform developers to rapidly define, integrate, and verify system level solutions and enable their end customers with a customized programming environment.

5.4.2. SDSOC target

The acceleration is the target in this field nowadays for using these systems in real time applications. The Graphics Processing Units is the solution but

its high-power consumption prevents its utilization in daily-used equipment moreover the Field Programmable Gate Array (FPGA) has low power consumption and flexible architecture. We added a new methodology to compromise the area requirements with the speed and design time by using Xilinx SDSOC tool (including processor and FPGA on the same board). Implementing design using HW/SW partitioning will enhance time design based on high level language (C or C++) in Vivado HLS (High Level Synthesis). It also fits for larger designs than using FPGA only and faster in design time.

5.4.3. Advantages of SDSOC

The main advantage of using SDSOC is the ability to implement larger designs & give less time for design, but the generated RTL is not optimized so it will take more area, power, and may be delay too. To explore the design space, first we choose each function for hardware acceleration and all other functions for software to get the power, area, and delay for each function separately. Then eliminate the other partitioning possibilities based on the parameters of each function. This way guarantee that we run the possible partitions only and get the best among them. And this is what SDSOC usually does, it separates the hardware component from the software ones.

5.4.4. SDSOC environment design flow

As shown the SDSOC environment design flow

- the first step is to identify compute-intensive hot spots in the application that can be migrated to programmable logic to achieve higher performance
- isolate them into functions that you can compile for hardware. C/C++ code compiled for programmable logic with the SDSOC environment must conform to coding guidelines and must also conform to Vivado High-Level Synthesis (HLS) guidelines.
- The remaining flow of SDSOC is selecting the functions for hardware acceleration and running the code with choosing estimate performance and generation of SD-Card image.

- We then take the SD-Card image to evaluation board chosen for implementation.
- After placing and routing is done on board, the tool generates an estimation report for speed up for using selected function for hardware implementation.
- We do this flow several times until we get the best optimum solution, which gives us the hardware functions with their generated RTL code.

The input language for SDSOC is a C/C++ code written according to high level synthesis (HLS) instructions approved by Vivado HLS tool supported with SDx environment tools.

The support of CPU and FPGA together on the same board makes a lot of combinations for implementing design which function will be executed by CPU and which one will be implemented on FPGA with HDL code generated to it.

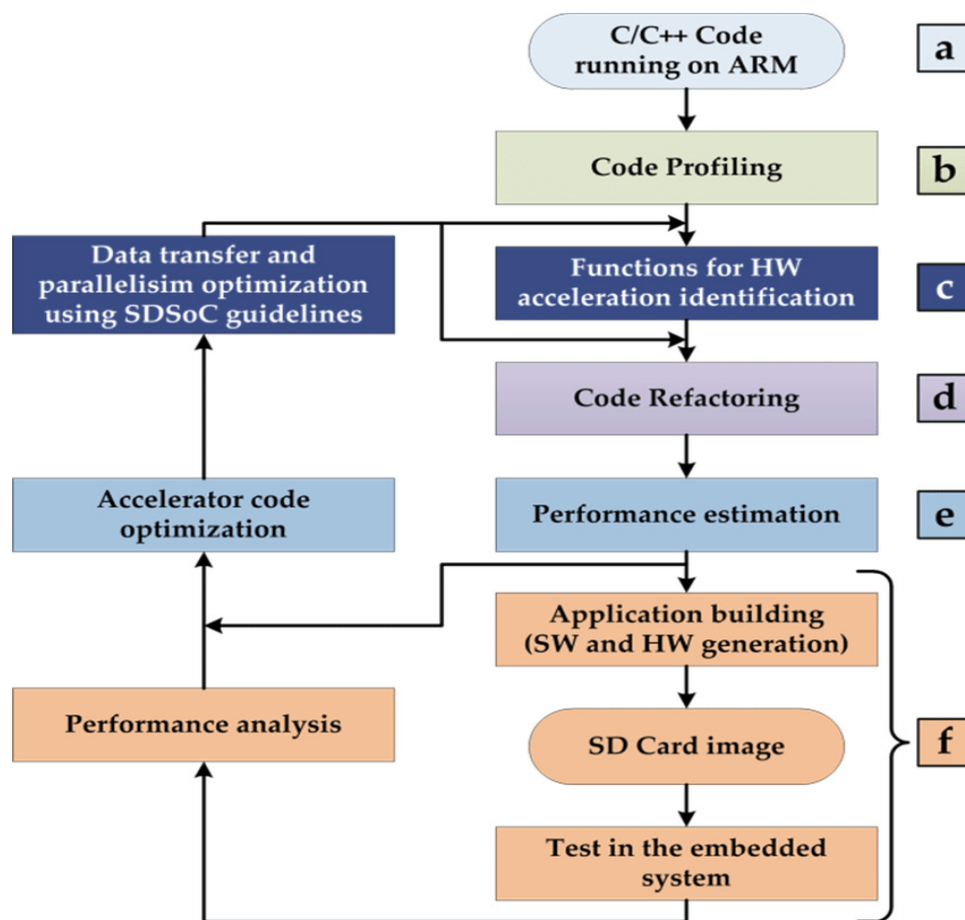


Figure 40: SDSoc flow chart

5.4.5. About the project

The design has been simulated on SDx 2018.3 environment using SDSoC, Vivado, and Vivado HLS. Our methodology is running the whole system by CPU only (SW solution) then takes each function to be implemented on FPGA and the other functions SW to get each function specifications (Hardware Resources, Latency, Power Estimations, Hardware accelerated cycles) separately then start to combine between the functions that can be fit on HW together and see the improvement in performance then finally decide the best combination of implementation to be done.

In this project, we use ZYNQ-7000 FPGA of "Xilinx" company, in order to show that it can use image processing algorithms at low cost.

"System of Chip - SoC" is defined on the development board. The lane tracking algorithm has been developed as an image processing project, and the hardware part has been implemented after this system has been broken down into software and hardware. Lane keeping algorithm was written in "C++" environment using "OpenCV" library and has 3 main functions. In the first function "pre-processing" section, meaningful information was tried to be extracted from the photo frame also some preprocessing is done like grayscale conversion, image filtering and binarization, in the second section, it was aimed to determine the lines belonging to the strip from this information extracted in the "stripe finding" function, and in the last section, the detected lines that would express the strip were painted and visualized with the "line drawing" function.

According to the time analysis, the first part, the preprocessing part, was implemented in hardware. SDSoC and Vivado platforms were used to implement the project on the ZYNQ-7000 development board.

When using high-level software (C++) on the SDSoC platform. Verilog, a hardware description language, is used on the Vivado platform.

5.4.6. SDSOC Environment

The concept of a platform is integral to the SDSoC environment as it defines the hardware, software, and meta-data components on which SDSoC applications are built. Multiple base platforms are available within the SDx IDE and can be used to create SDSoC applications without first having to create a custom platform. The SDx IDE utilizes the sds++ system

compiler to convert C/C++ code into high-performance hardware accelerators that attach to platform interfaces as determined by the platform designer and by application code pragmas. Declarations within the platform meta-data identify interface ports, clocks, interrupts, and reset blocks for use by the system compiler when it attaches hardware accelerators to the base platform.

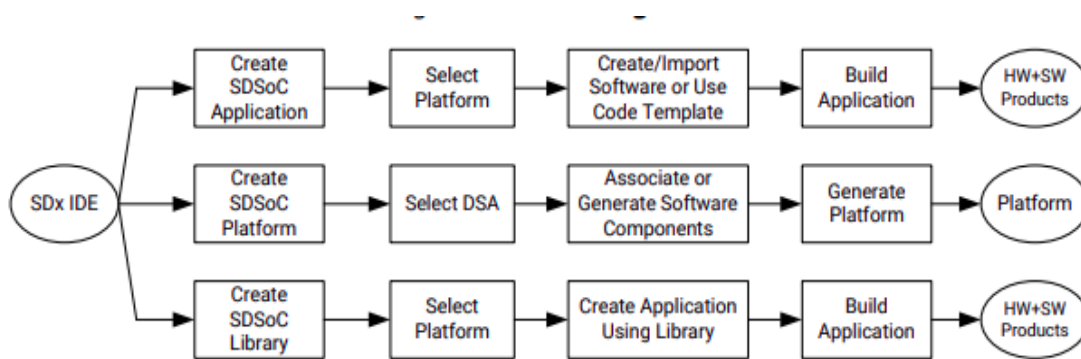


Figure 41 projects can be implemented on SDSOC

The system compiler analyzes a program to determine the dataflow between software and hardware functions and generates an application-specific system-on-chip. The sds++ system compiler generates hardware IP and software control code that implements data transfers and synchronizes the hardware accelerators with application software. Performance is achieved by pipelining communication and computation, thereby producing hardware functions that can run with maximum parallelism as illustrated in the following figure.

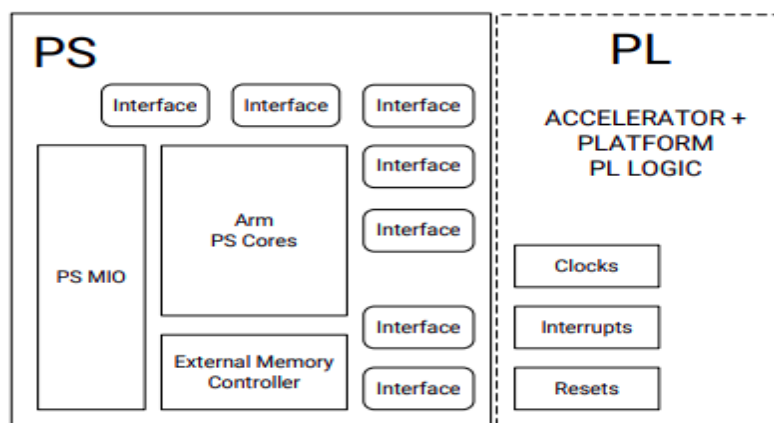


Figure 42: System on chip (SoC)

The sds++ system compiler invokes the Vivado® High-Level Synthesis

(HLS) tool to transform software functions into a bitstream that defines and configures the programmable logic (PL) portion of the SoC. In addition, stub functions are generated so application software compiled and linked using the standard GNU toolchain transparently uses the implemented hardware functions. All necessary drivers and libraries are automatically included in this system compilation process. The final output of system compilation is the generated `sd_card` directory, which at minimum is populated with a Zynq bootable `BOOT.BIN` file, the executable and linkable format (ELF) file application code, and a `README.txt` boot instructions file. The `BOOT.BIN` file contains any necessary bootloaders, bitstreams, and application code to boot the generated system on a target board. For systems that run Linux on the target board, the `sd_card` directory also contains the Linux image file used during the boot process.

5.5. Implementation using SDSoC

5.5.1. Hardware resources utilization

After performing Debug build for design with choosing only one hardware function and the other functions are executed by CPU, we get the detailed reports for synthesis and implementation of this function to get hardware utilizations (LUT, BRAMs, DSPs, and FFs).

5.5.2. Power estimations

The tool generates the estimated power (Watt) consumptions for the hardware partitioned functions including on chip power consists of dynamic power, programmable logic (PL), Processing system (PS), and static power (PL and PS). These numbers based on the automatic generated RTL from the tool. We get the numbers using Vivado power estimator for the generated RTL Project.

5.5.3. Hardware accelerated cycles

Hardware acceleration is metric defined by SDSoC tool. Hardware acceleration is the number of clock cycles improvement in execution of system if the function is implemented as hardware function on the programmable logic. the tool generates the hardware acceleration for the generated platforms of the synthesized hardware as estimated by debugging compilers of tool. The used performance estimation assumes

worst-case latency of hardware functions, it also assumes worst-case data transfer size for arrays so it could be the hardware function latency and data transfer size at run time is smaller than such assumptions).

5.5.4. Implementation combinations

There are several implementation combinations for the design as we said. We will choose some of them depends on the results of area of each functions to decide which functions will implement on HW and the others on SW. reports show the utilization of hardware resources for different implementations. the implemented combinations for HW show the accelerated clock cycles that will improve speed of system. Furthermore, the dynamic power is the dominant in on Chip power so the different solutions have a small effect on PL power, as it is much smaller than PS power. Hence, the PS power is the dominant term in dynamic power too. These results are very useful guide to choose and eliminate the combinations for implementation.

5.5.5. Comparison between RTL and SDSoC HW implementation

Here is a simple comparison based on the previous results of synthesis phase for both RTL and SDSOC. The optimized RTL has better results as SDSOC generates un-optimized RTL code RTL takes a lot of LUTs but with minimum use of BRAMs and FFs but SDSOC goes to use BRAMs more than LUTs and DSPs. On the other hand, the SDSOC flow is giving less time for design, more flexible for any change, the optimized RTL can replace for the generated RTL from Tool, which will be better comparable to optimized RTL regular flow. This is the main part of using SDSOC flow for fast design time and fitting larger designs.

5.5.6. Sample of generated reports

Table 3: Sample of generated reports

Resource utilization estimates for Hardware functions			
Resource	Used	Total	% Utilization
DSP	162	1728	9.38
BRAM	64	312	20.51
LUT	29965	230400	13.01
FF	38884	460800	8.44

```

Timing (ns):
* Summary:
+-----+-----+-----+-----+
| Clock | Target| Estimated| Uncertainty|
+-----+-----+-----+-----+
| ap_clk | 10.00| 6.437| 2.70|
+-----+-----+-----+-----+

Latency (clock cycles):
* Summary:
+-----+-----+-----+-----+
| Latency | Interval | Pipeline|
| min | max | min | max | Type |
+-----+-----+-----+-----+
| 1031| 1031| 1031| 1031| none |
+-----+-----+-----+-----+

```

Figure 43: Reports generated by SDSOC

Design guidelines

The guidelines for implementation using SDSOC.

- Write the input C++ code with optimized techniques supported by the SDSOC tool.
- Divide design into sub-functions considering the functions that will target hardware will be written in a good way that suits HW implementation.
- Using the useful pragmas that helps to decrease time and resources
- After getting the implementation of design try to replace RTL generated code with another optimized one that will enhance performance.
- Finally, one of possible work in this area is to using partial dynamic reconfiguration (PDR).

5.6. SDSOC Build Process

The SDSOC build process uses a standard compilation and linking process. Similar to g++, the sds++ system compiler invokes sub-processes to accomplish compilation and linking.

As shown in the following figure, compilation is extended not only to

object code that runs on the CPU, but it also includes compilation and linking of hardware functions into IP blocks using the Vivado High-Level Synthesis (HLS) tool, and creating standard object files (.o) using the target CPU toolchain. System linking consists of program analysis of caller/callee relationships for all hardware functions, and the generation of an application-specific hardware/software network to implement every hardware function call. The sds++ system compiler invokes all necessary tools, including Vivado HLS (function compiler), the Vivado Design Suite to implement the generated hardware system, and the Arm compiler and sds++ linker to create the application binaries that run on the CPU invoking the accelerator (stubs) for each hardware function by outputting a complete bootable system for an SD card.

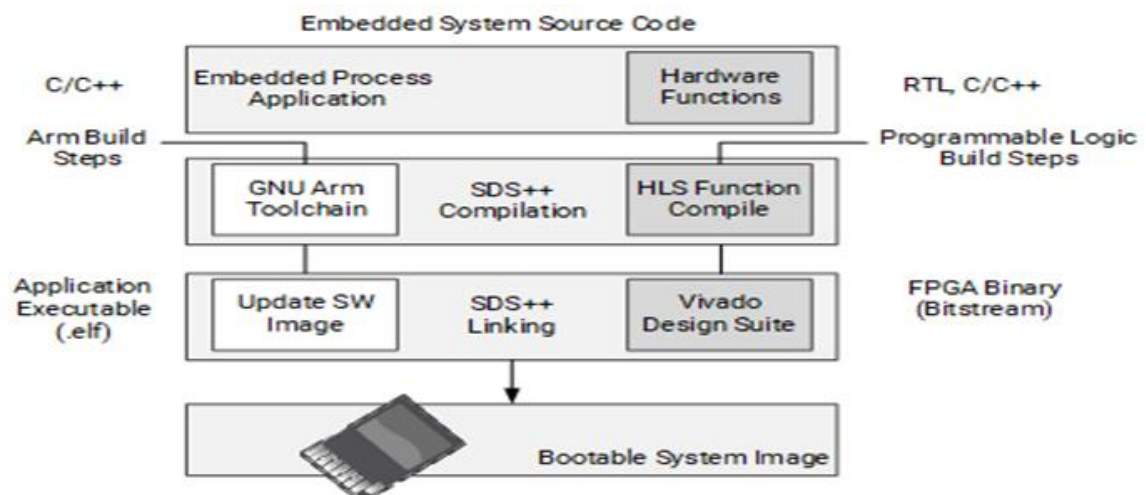


Figure 44: SDSoC flow

The compilation process includes the following tasks:

1. Analyzing the code and running a compilation for the main application on the Arm core, as well as a separate compilation for each of the hardware accelerators.
2. Compiling the application code through standard GNU Arm compilation tools with an object (.o) file produced as final output.
3. Running the hardware accelerated functions through the HLS tool to start the process of custom hardware creation with an object (.o) file as output.

After compilation, the linking process includes the following tasks:

1. Analyzing the data movement through the design and modifying the hardware platform to accept the accelerators.
2. Implementing the hardware accelerators into the programmable logic (PL) region using the Vivado Design Suite to run synthesis and implementation, and generate the bitstream for the device.
3. Updating the software images with hardware access APIs to call the hardware functions from the embedded processor application.
4. Producing an integrated SD card image that can boot the board with the application in an Executable and Linkable Format (ELF) file.

The project creation

We use the “XfOpenCV” library. It is a library which is derived from the “OpenCV” library and is an optimized library for hardware implementation of many OpenCV functions for Xilinx FPGAs [30]. Working with Xilinx SDSoC platform, this library acts as an accelerator in computer vision projects. The first version was released in June 2017, and this project uses the third version - the current latest version - released in December 2017.

Since SDSOC is an Eclipse IDE based development platform, there is a similarity in project creation stages. When the program is run for the first time, first the "workspace" file path where the projects will be saved is created, then the main section welcomes the user. In this area, new projects can be created, files containing new FPGA development cards can be transferred, previous projects can be imported, and educational documents about SDSoC can be accessed. A screenshot of this area is given in Figure. FPGA development to be used in SDSoC projects to be created.

After the card is selected, the operating system that the project will run on is selected under the system configuration title (Linux, Standalone, FreeRTOS). In this project, "Linux" was chosen as the system

configuration while using the ZC702 development board.

xfOpenCV library linking

In the project created to link the XfOpendcv library to the SDSoC platform, set the "C/C++ build" setting and go to the "Directories" tab under the "SDS ++ Compiler" heading and add the XfOpendcv library to the "Include Paths" path.

OpenCV library linking

In order for the OpenCV library to be used on the Zynq Board Development Board the OpenCV library must be installed according to the "AARCH32" architecture. After the installation in accordance with the processor architecture on the FPGA card used, the project belongs to:

- OpenCV library compiled for the computer environment (Windows in this project) used in the "Directories" tab under the "SDS ++ Compiler" heading in the "C/C++ build" setting is added to the "Include Paths" path.
- In addition, OpenCV cores used in the project in the "Libraries" path in the "SDS ++ Linker" tab under the "C/C++ build" setting; The OpenCV library, which is installed for the appropriate target architecture (AARCH32 in this project), is added to the "Library Search Path" path.



Figure 45: SDSOC Platform

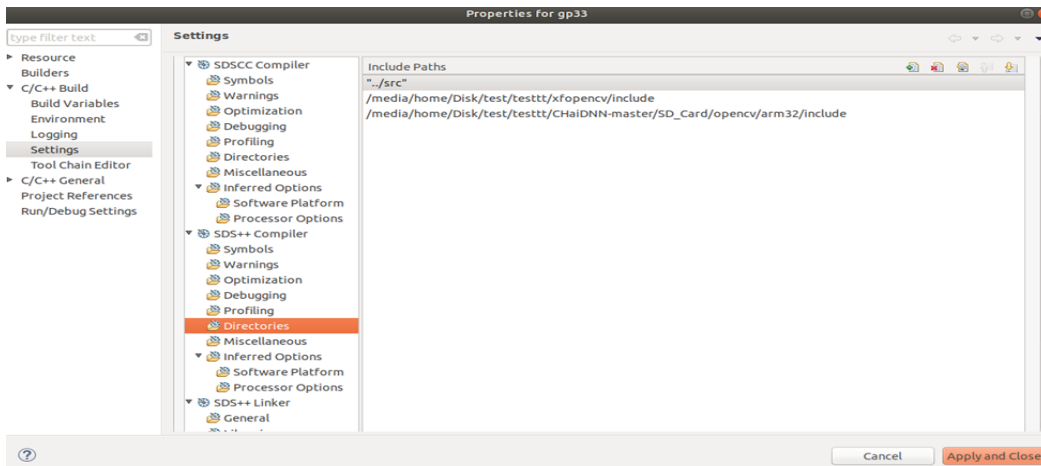


Figure 46: Linking openCV and xfopencv library

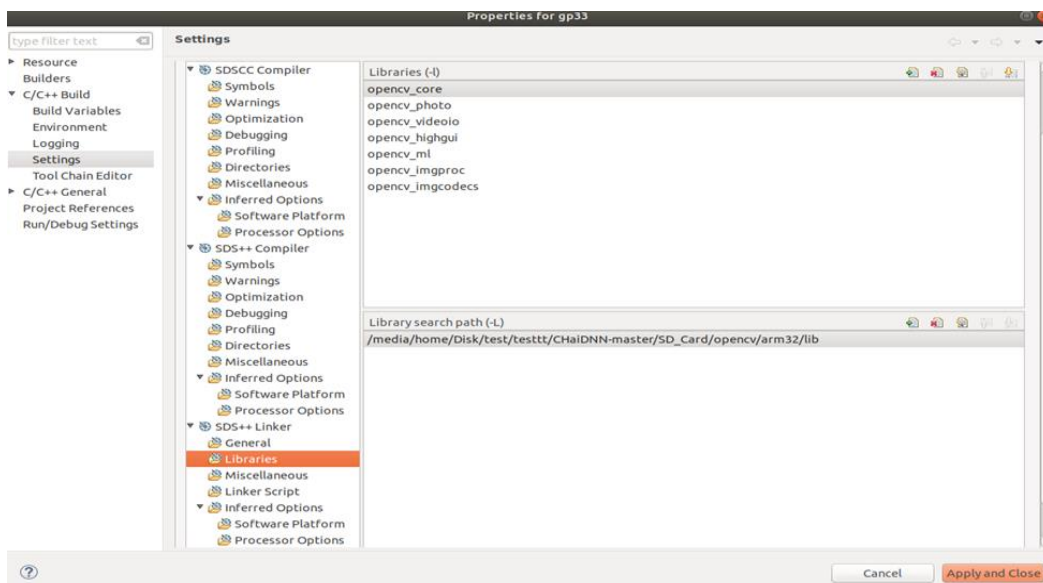


Figure 47: Linking the OpenCV library

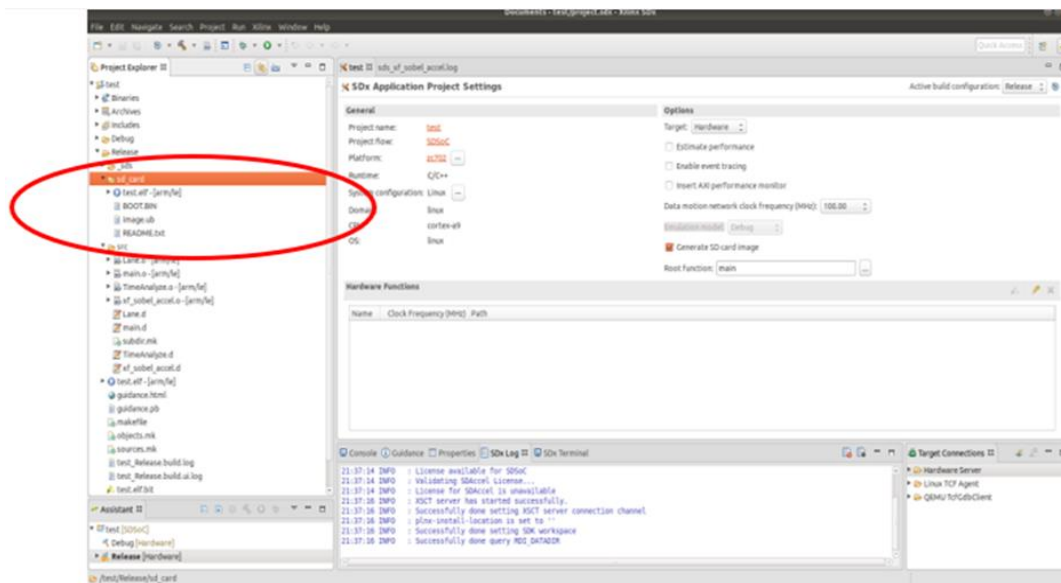


Figure 48: The Executable and Linkable Format (ELF) file.

5.7. Hardware and Software separation

In order to implement the lane detection system faster, a part of the algorithm of the system is thought to be implemented in hardware. The ZC702 development board of the ZYNQ-7000 series of Xilinx company, which is suitable for this system, which will be separated as hardware and software, has been selected. In the ARM Cortex-A9 processor available in the software block ZC702; The hardware block will be implemented in the logic cells section of the ZC702.

5.7.1. Time Analysis

Table 4: Lane keeping system time analysis

	Number	(in a frame)	(in a frame)	(in a frame)
Pre-Processing	1260	81.71 ms	8.21 ms	36.99ms
All Software	1260	153.31ms	25.47ms	77.84ms
Preprocessing/AllSoftware	-	53.3%	32.2%	47.5%

5.7.2. Design Notes

Partition for the code

Problem: this error appeared while trying to mark any function to be accelerated on FPGA.

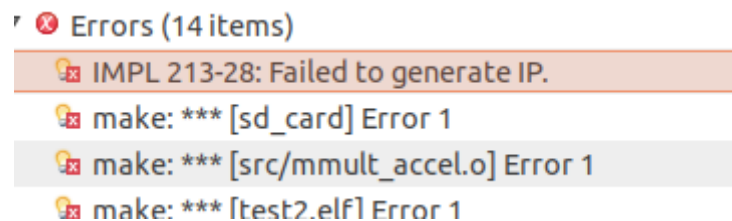


Figure 49: partition errors

Solution: change the Date&Time of the operating system to be before 2020(This is a bug in the tool).sample of the reports after running the project.

```
Timing (ns):
* Summary:
+-----+-----+-----+-----+
| Clock | Target| Estimated| Uncertainty|
+-----+-----+-----+-----+
|ap_clk | 10.00| 7.016| 2.70|
+-----+-----+-----+-----+
```

Figure 50: reports 1 by SDSOC

Resource utilization estimates for Hardware functions			
Resource	Used	Total	% Utilization
DSP	162	1728	9.38
BRAM	64	312	20.51
LUT	29965	230400	13.01
FF	38884	460800	8.44

Figure 51: reports 2 by SDSOC

OpenCV and xf-opencv libraries

Problem: sdx can't find the libraries

Solution: add the path of the libraries (folder of include and lib) as shown below.

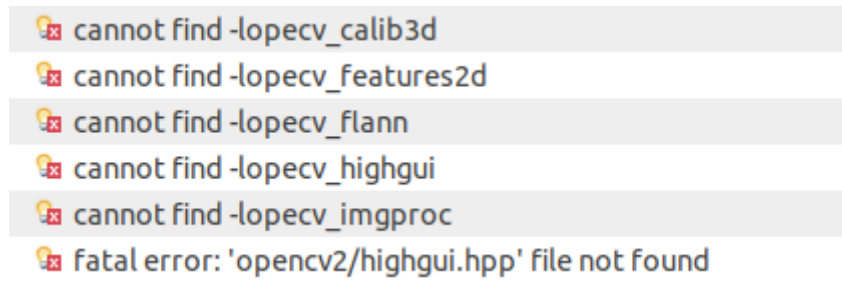


Figure 52: xfopencv libraries

OpenCV libraries


Problem:  SdsCompiler 83-5019: Exiting sds++ : Error when calling 'arm-linux-gnueabi-g++

Figure 53: OpenCV library error 1

Solution: this error appeared as the version of the opencv doesn't suitable with the target platform (you should use version so.3.1).

Problem:

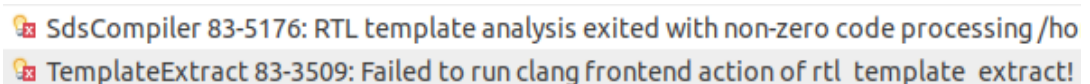


Figure 54: OpenCV library error 2

Solution: remove -hls-target 1 flag from sdscc compiler as shown

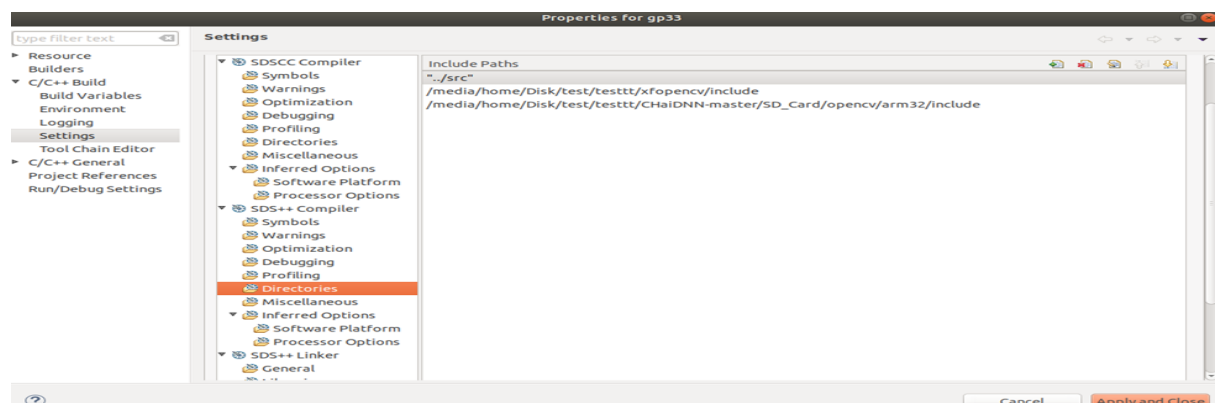


Figure 55: OpenCV library solution

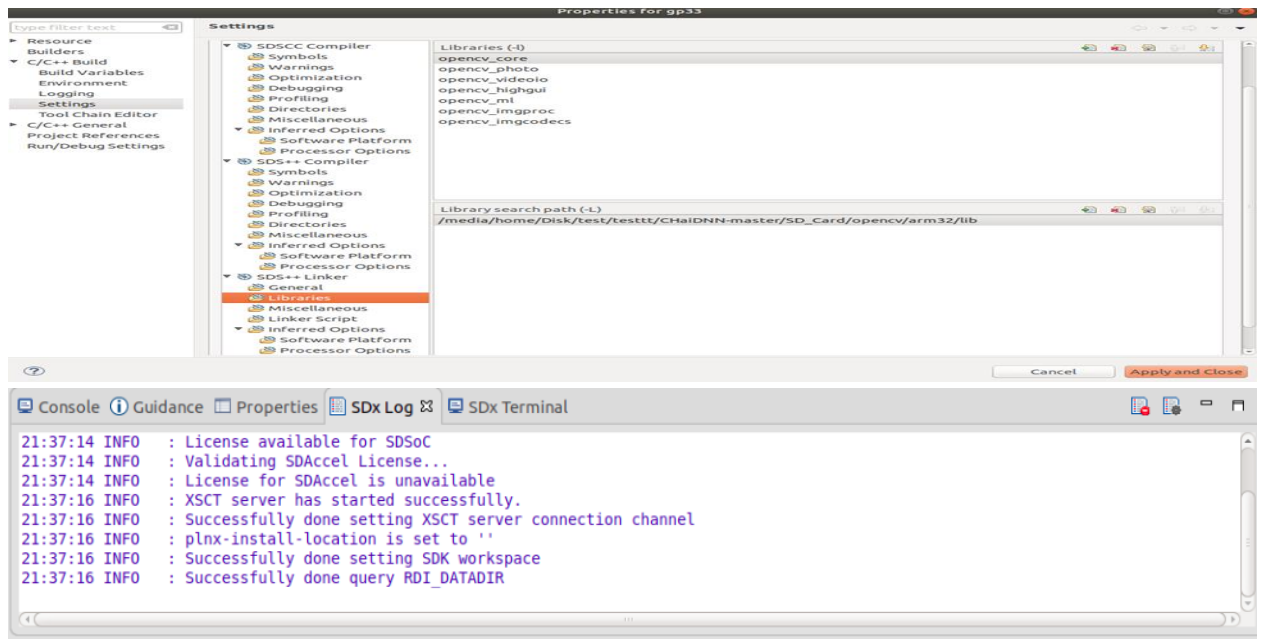


Figure 56: sdx log after running

partition for lane detection algorithm with opencv libraries.

Problem:

```
ERROR: [SdsCompiler 83-5019] Exiting sds++ : Error when calling '/media/home/Disk/test/testtt/Vivado/2018.3/bin/vivado_hls /home/home/Documents/test/Release/
```

Figure 57: Sds compiler error

Solution: Install gcc-multilib package in Ubuntu.

5.8. The Lane Detection Algorithm

Here we use with xf opencv libraries on SDSoC , but after implementing the code on the kit we get an error which need to use static libraries to choose the files which need to implement, to do this task we need to use petalinux environment.

```
Starting tcf-agent: OK
Last login: Tue Nov 20 03:07:50 UTC 2018 on tty1
root@xilinc-zc702-2018_3:~# /mnt/test.elf
/mnt/test.elf: error while loading shared libraries: libopencv_core.so.3.1: cannot open shared object file: No such file or directory
root@xilinc-zc702-2018_3:~#
```

Figure 58: opencv library error

5.8.1. PetaLinux

PetaLinux is a tool used to create your personal system which is compatible with hardware you work on it. It helps you create and deliver a custom Linux distribution. They allow you to work easily with available software which is independently available from the Xilinx GIT or open source communities.

5.8.1.1. Yocto project

The Yocto Project (YP) is an open source collaboration project that helps developers create custom Linux-based systems. Xilinx provides countless meta layers that enable developers to build all the necessary components for running Linux on Xilinx SoCs. PetaLinux Tools are built on top of the YP infrastructure.

Installing PetaLinux in your system

Step 1:

All of the Xilinx tools require 32-bit libraries at some point in time to compile. DocNav requires several 32-bit libraries and PetaLinux needs 32-bit architectures for cross compilation. Therefore the first step is to add the 32-bit architecture to your Ubuntu system. Since there is a 99.999% chance your computer has an Intel based processor, add i386 using the package management system, dpkg

```
~$ sudo dpkg --add-architecture i386
```

Step 2:

Then install all of the required package dependencies for the Xilinx tools
Some of these packages are:

- tofrodos
- iproute
- gawk
- gcc
- git-core
- make
- net-tools

- ncurses-dev
- libncurses5-dev
- tftpd*
- zlib1g-dev
- flex
- bison
- lib32z1
- lib32ncurses5
- lib32bz2-1.0
- ia32gcc1
- lib32stdc++6
- libselinux1

Step 3:

Then install the PetaLinux tool

```
nate@nate-N56JR: ~/Downloads
nate@nate-N56JR:~$ mkdir PetaLinux
nate@nate-N56JR:~$ cd Downloads
nate@nate-N56JR:~/Downloads$ ./petalinux-v2015.4-final-installer-dec.run ../PetaLinux/
```

Figure 59: command for installation

After writing this command the license of petalinux will appear to accept it and start installation

```
Activities Terminal 02:19
ubuntu@ubuntu-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
cept; (iii) by facsimile transmission, upon acknowledgment of receipt of electronic transmission, provided that notice is also provided by one of the other methods herein within five (5) days thereafter; or (iv) by certified or registered mail, return receipt requested, upon verification of receipt. Notice shall be sent to the addresses provided by each party to the other in connection with this Agreement, or to such other address as either party may specify in writing. Notices to Xilinx shall be addressed to the attention of: Xilinx, Inc., Attn: General Counsel, Legal Department, 2100 Logic Drive, San Jose, CA 95124.
(i) Entire Agreement. This Agreement constitutes the entire agreement between the parties with respect to the Software, and supersedes all prior or contemporaneous discussions, understandings or agreements, written or oral, regarding the subject matter hereof. No additional terms or modifications proposed by Licensee shall be binding on Xilinx unless expressly agreed to in writing and signed by Xilinx. All terms and conditions of any purchase order or other document issued by Licensee in connection with this Agreement or the Software shall be void and of no force or effect to add to or modify this Agreement.
(j) Interpretation. By clicking to "accept" or "agree" or entering "03:yes-04" or "03:y-04" to this Agreement, Licensee acknowledges and agrees that it has read and understood this Agreement, has had an opportunity to discuss this Agreement with its legal and other advisors, and agrees to be bound by the terms and conditions of this Agreement. This Agreement shall be interpreted fairly in accordance with its terms and without any strict construction in favor of or against either party.
2015.07.03
```

Figure 60: license of PetaLinux

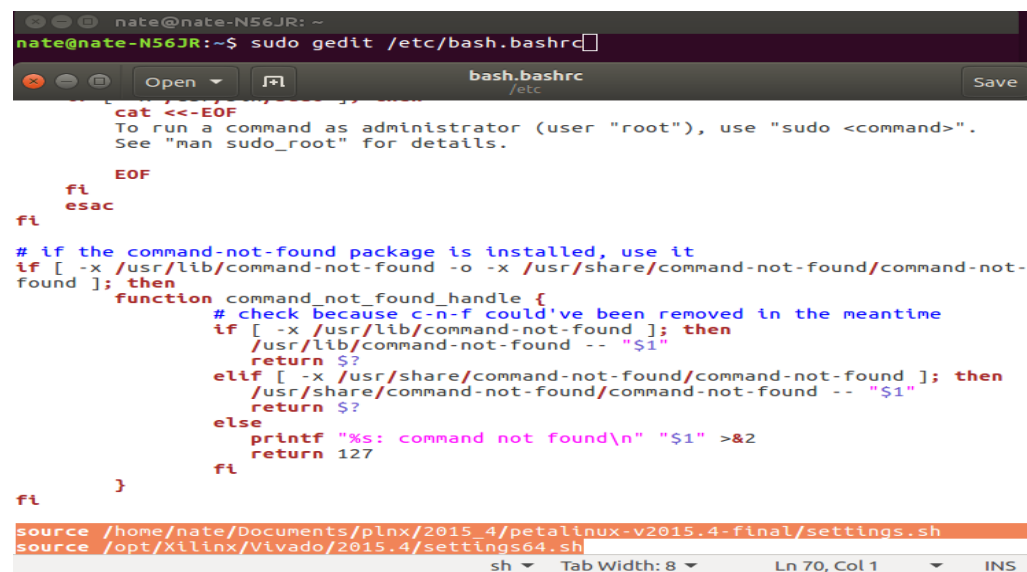
Step 4:

Change Ubuntu's shell from dash to bash as PetaLinux Is only compatible with bash:

```
~$ sudo dpkg-reconfigure dash
```

The PetaLinux tools require you to use 'bash' as your shell rather than 'dash', which is likely your default shell if you're running Ubuntu

The next thing to take care of will be to source the tools for PetaLinux to use within the terminal window. This includes the 'settings64.sh' and 'settings.sh' files in your Vivado and PetaLinux installation directories, respectively. To avoid needing to type the source commands into the shell every time, you can add a couple lines to the .bashrc script. To modify this system wide, use a text editor to open your .bashrc file. For Ubuntu, this will be bash.bashrc located in the /etc directory (see following command and/or first image above).



```
nate@nate-N56JR: ~
nate@nate-N56JR:~$ sudo gedit /etc/bash.bashrc
bash.bashrc
/etc

cat <<-EOF
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

EOF
fi
esac
fi

# if the command-not-found package is installed, use it
if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found/command-not-found ]; then
    function command_not_found_handle {
        # check because c-n-f could've been removed in the meantime
        if [ -x /usr/lib/command-not-found ]; then
            /usr/lib/command-not-found -- "$1"
            return $?
        elif [ -x /usr/share/command-not-found/command-not-found ]; then
            /usr/share/command-not-found/command-not-found -- "$1"
            return $?
        else
            printf "%s: command not found\n" "$1" >&2
            return 127
        fi
    }
fi

source /home/nate/Documents/plnx/2015_4/petalinux-v2015.4-final/settings.sh
source /opt/Xilinx/Vivado/2015.4/settings64.sh

sh Tab Width: 8 Ln 70, Col 1 INS
```

Figure 61: add setting files to bash shell

Step 5: optional

To use PetaLinux, you will need a PetaLinux project directory to work in. This can be done either by creating a totally new project or by using a reference design provided in a board support package (BSP). Creating a fresh project provides you with a basic template from which you can start your development. Just change to a directory you would like to create your project in and enter the following command.

```
nate@nate-N56JR: ~/Documents
nate@nate-N56JR:~/Documents$ petalinux-create --type project --template zynq --name test_01
INFO: Create project: test_01
INFO: New project successfully created in /home/nate/Documents/test_01
```

Figure 62: create a normal project in PetaLinux

The '--type' parameter should remain 'project', the '--template' parameter should be whatever supported architecture you are targeting (either zynq, zynqMP for Ultra scale chips, or microblaze for soft processors implemented in FPGA fabric), and the '--name' parameter can be whatever you want to name your project. Do note that this simply provides a folder structure for PetaLinux to use and requires you to provide pretty much every part of the build, from the first stage boot loader to the file system, and is not suggested for those new to Linux development. New players should instead use a BSP!

Step 6:

Creating a new project from a BSP is the simplest way to get started with PetaLinux, since it provides you with an already functioning and bootable Linux image that you start playing with.

PetaLinux board support packages (BSPs) are reference designs on supported boards for you to start working with and customizing your own projects. In addition, these designs can be used as a basis for creating your own projects on supported boards. PetaLinux BSPs are provided in the form of installable BSP files, and include all necessary design and configuration files, pre-built and tested hardware, and software images ready for downloading on your board or for booting in the QEMU system emulation environment.

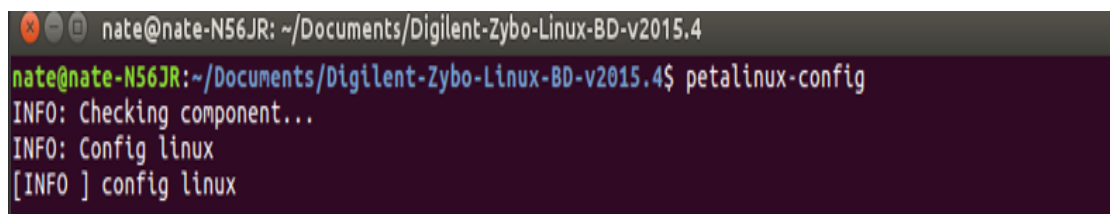
```
ubuntu@ubuntu-VirtualBox:~$ petalinux-create -t project -s /home/ubuntu/Documents/bsp/ZC702.bsp
INFO: Create project:
INFO: Projects:
INFO: * xilinx-zc702-2020.1
INFO: has been successfully installed to /home/ubuntu/
INFO: New project successfully created in /home/ubuntu/
```

Figure 63: Create a bsp project on PetaLinux

We should use the same version of PetaLinux to create this project and the same version of Vivado.

Step 7:

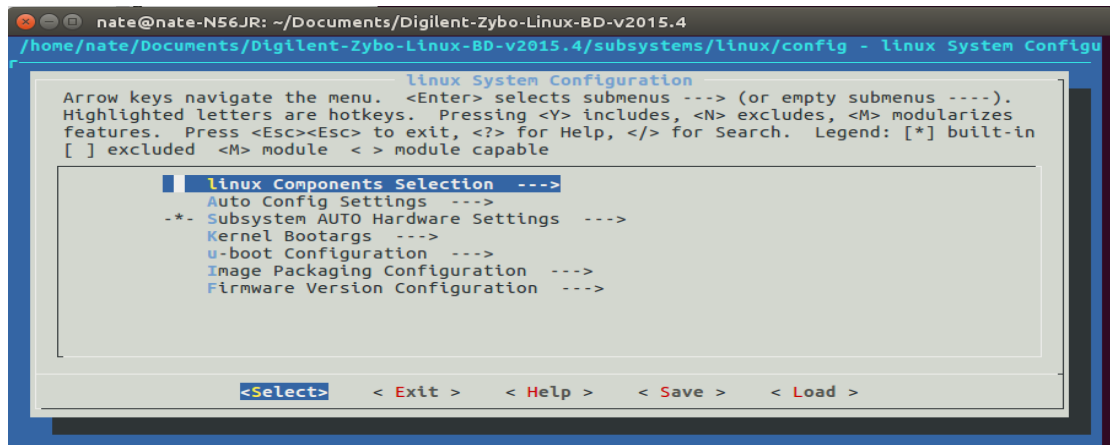
This step is very straight forward from an end user's perspective, but will require you to accept a bit of 'magic' in the background if you are not intimately familiar with the process of compiling a Linux image from scratch. Suffice it to say that by the end of the configure and build process in PetaLinux, you will have a kernel, file system, first stage and second stage boot loaders, and device tree compiled and ready to be deployed to your hardware target. To run configuration on the BSP project you just created, change directory into the directory that was made with the 'petalinux-create' command, and type in the following.



```
nate@nate-N56JR: ~/Documents/Digilent-Zybo-Linux-BD-v2015.4
nate@nate-N56JR:~/Documents/Digilent-Zybo-Linux-BD-v2015.4$ petalinux-config
INFO: Checking component...
INFO: Config linux
[INFO ] config linux
```

Figure 64: command to show configuration

When you enter this command the configuration window will appear



```
nate@nate-N56JR: ~/Documents/Digilent-Zybo-Linux-BD-v2015.4
/home/nate/Documents/Digilent-Zybo-Linux-BD-v2015.4/subsystems/linux/config - linux System Configu

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

linux Components Selection --->
  Auto Config Settings --->
  -* Subsystem AUTO Hardware Settings --->
  Kernel Bootargs --->
  u-boot Configuration --->
  Image Packaging Configuration --->
  Firmware Version Configuration --->

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

Figure 65: configuration window

After that we build the project and try an example to check if it is installed correctly or not

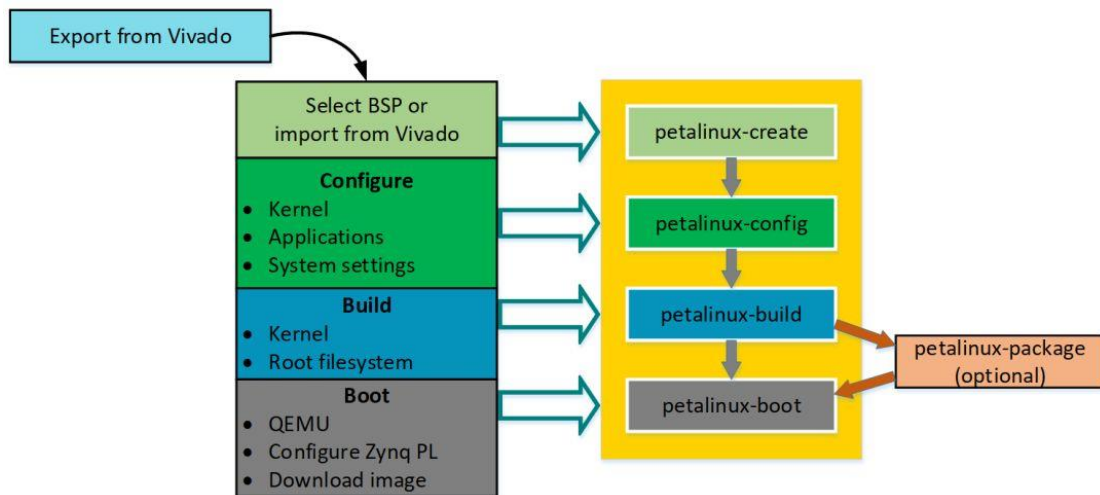


Figure 66: steps on PetaLinux tool

There are some errors when we build a project:

problems:

1. firstly installation permission was denied.

solution: use sudo apt command.

```
home@home-Inspiron-5593:~/Desktop/petalinux$ ./petalinux-v2021.1-final-installer
.run
bash: ./petalinux-v2021.1-final-installer.run: Permission denied
home@home-Inspiron-5593:~/Desktop/petalinux$
```

Figure 67: Petalinux error 1

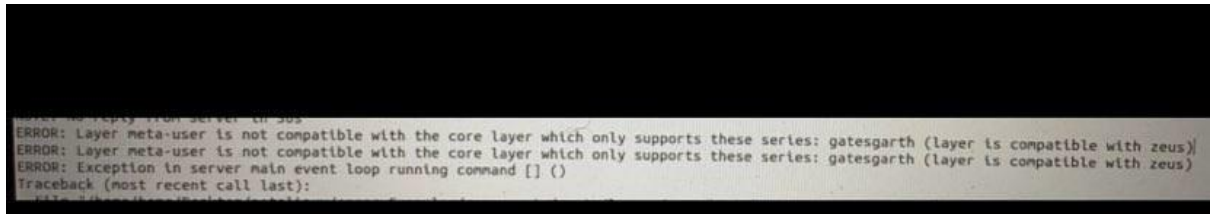
2. Installation failed.

Solution: use another version PetaLinux and install it in another location on the operation system.

```
.....
Please input "y" to continue to install PetaLinux in that directory?[n]y
INFO: Checking PetaLinux installer integrity...
INFO: Installing PetaLinux SDK to "/media/home/Disk/test/testtt/."
y
INFO: Installing buildtools in /media/home/Disk/test/testtt/./components/yocto/buildtools
INFO: Installing buildtools-extended in /media/home/Disk/test/testtt/./components/yocto/buildtools_extended
INFO: PetaLinux SDK has been installed to /media/home/Disk/test/testtt/.
```

3. Build the project on PetaLinux (still under processing and i am trying to solve it)

Solution: download the suite layer(gategarth) to be compatible.



4. Can't read the file.cpp that is created and edit in the file.bb which is created with the project

Solution: we should use an Editor to edit in the file so we use vim Editor

And to save vim file press **Esc** then **:w** and to exit from the editor press **Esc** then **:x**

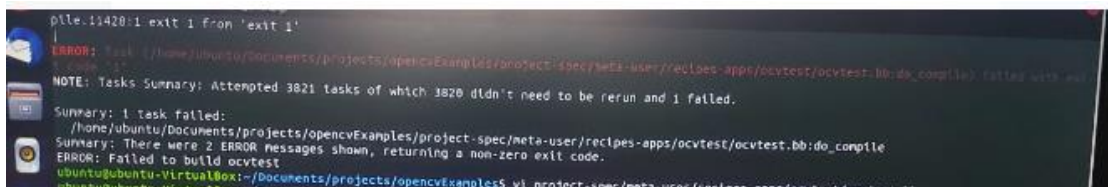
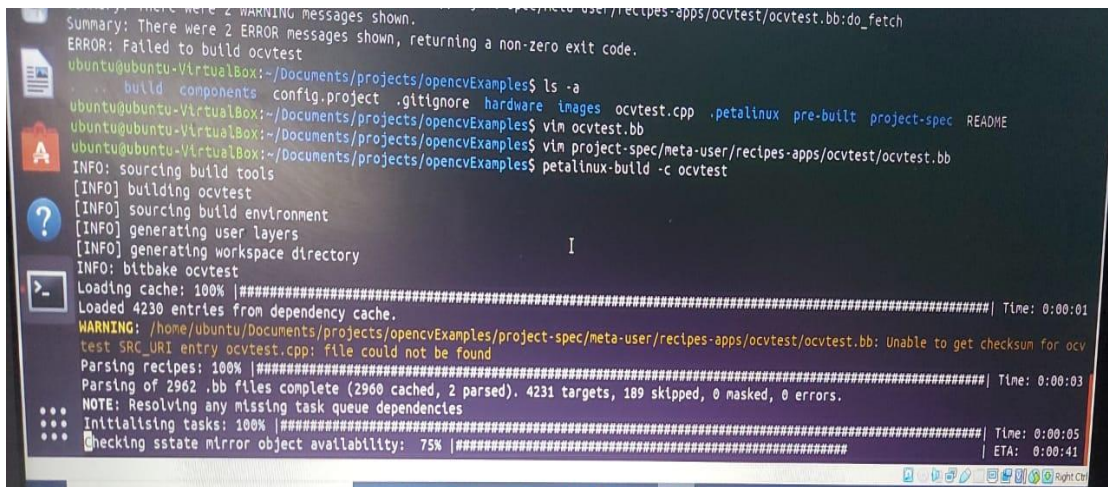


Figure 68: Errors solution

After build the project it will create boot file and image file, we take them and put with the image that chosen on SD Card and put them in the board.

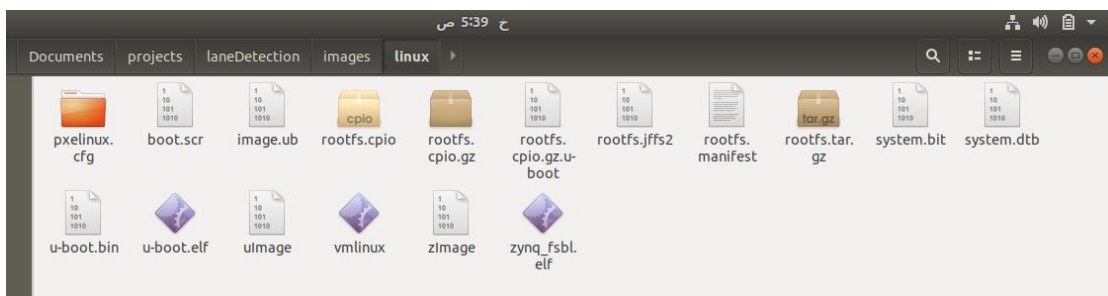


Figure 69: files generated after build the project

5.9. Zynq 7000 board

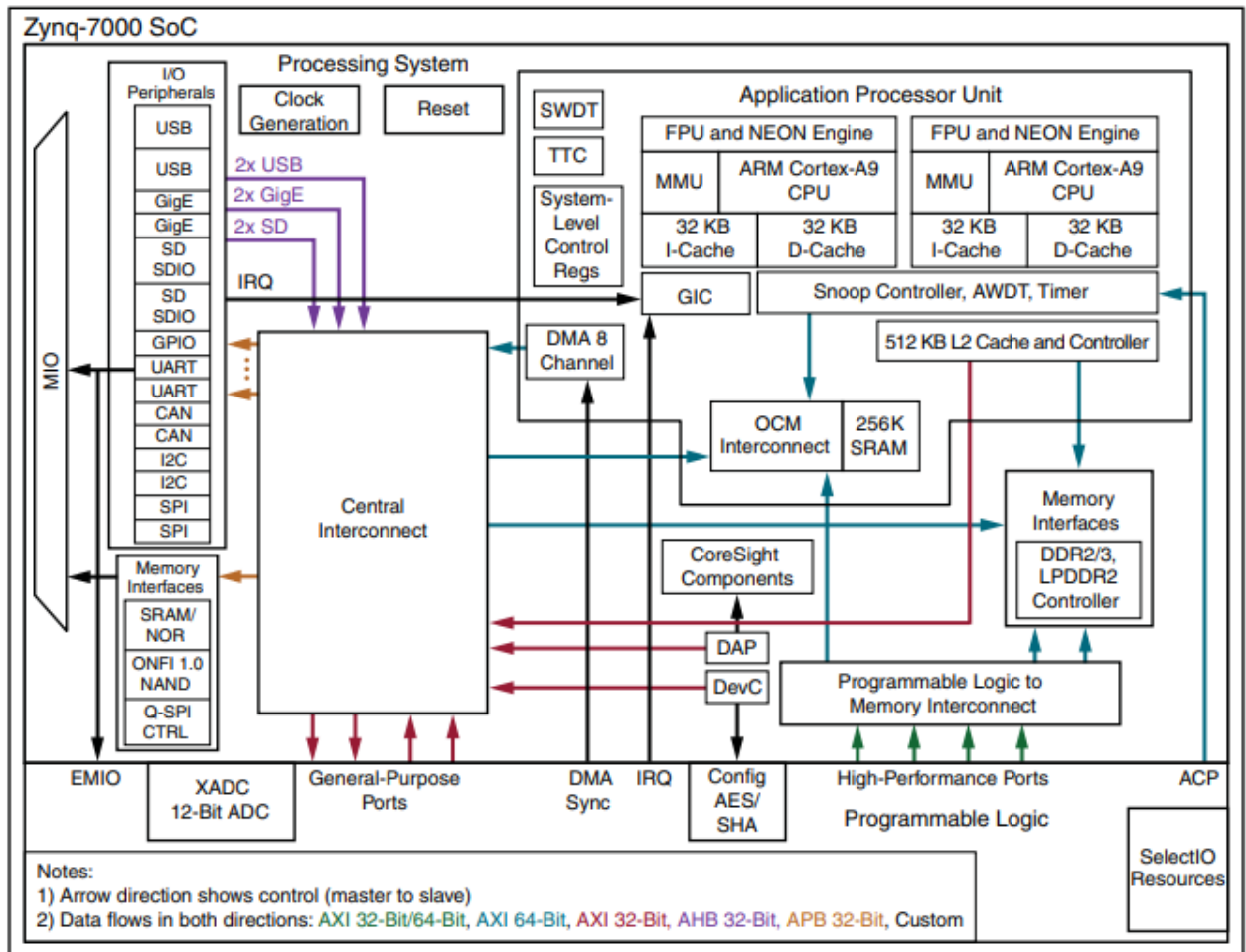


Figure 70: Block Diagram of ZYNQ7000

The Zynq®-7000 SoC family integrates the software programmability of an ARM®-based processor with the hardware programmability of an FPGA, enabling key analytics and hardware acceleration while integrating CPU, DSP, ASSP, and mixed signal functionality on a single device. Consisting of single-core Zynq-7000S and dual-core Zynq-7000 devices, the Zynq-7000 family is the best price to performance-per-watt, fully scalable SoC platform for your unique application requirements.

Naturally, as in any design project, the first stage is to define the desired behaviors of the system, to create an appropriate specification from a set of requirements. This is depicted as the starting point at the top of the diagram, and it forms the basis of the system design that is subsequently developed.

As mentioned earlier in this chapter, the Zynq architecture combines an ARM processor (for software elements of the designed system) with FPGA fabric (predominantly for hardware elements of the system, although additional processors can also be implemented here too, if desired). A key element of the system design stage, which comes next, is therefore to partition the intended functionality appropriately between software and hardware, and to define the interfaces between the two sections. Of course, it is possible that this partitioning will subsequently be adjusted as the designers iterate the system towards completion.

Having partitioned the system, software and hardware development can then progress in parallel, to a large extent. In terms of hardware development, the task is to identify the necessary functional blocks to achieve the design, and to thereafter assemble them through some combination of design reuse and new IP development, and make appropriate connections between the blocks. Similarly, the software aspect of the project can be realized through developing custom code or by reusing pre-existing software. Verification of both software and hardware will be required, and this forms an integral and important part of the process.

Application Processing Unit

Basic Functionality

The application processing unit (APU), located within the PS, contains one processor for single-core devices or two processors for dual-core devices. These are Arm® Cortex™-A9 processors with NEON co-processors connected in an MP configuration sharing a 512 KB L2 cache. Each processor is a high-performance and low-power core that implements two separate 32 KB L1 caches for instruction and data. The Cortex-A9 processor implements the Arm v7-A architecture with full virtual memory support and can execute 32-bit Arm instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java™ byte codes in the Jazelle state. The NEON™ co-processor media and signal processing architecture add instructions that target audio, video, image and speech processing, and 3D graphics. These advanced single instruction multiple data (SIMD) instructions are available in both Arm and Thumb states

The Cortex-A9 processor(s) within the APU are organized in an MP configuration with a snoop control unit (SCU) responsible for maintaining

L1 cache coherency between the two processors and the ACP interface from the PL. To increase performance, there is a shared unified 512 KB level-two (L2) cache for instruction and data. In parallel to the L2 cache, there is a 256 KB on-chip memory (OCM) module that provides a low-latency memory.

Central Processing Unit (CPU)

Each Cortex-A9 CPU can issue two instructions in one cycle and execute them out of order. The CPU implements dynamic branch prediction and with its variable length pipeline can deliver 2.5 DMIPs/MHz The Cortex-A9 processor implements the Armv7-A architecture with full virtual memory support and can execute 32-bit Arm instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java™ byte codes in the Jazelle hardware acceleration state.

Zynq evaluation board has more than one type, we use zc702 evaluation kit.

5.9.1. ZC702 Board

Overview

The ZC702 evaluation board for the XC7Z020 SoC provides a hardware environment for developing and evaluating designs targeting the Zynq® XC7Z020-1CLG484C device. The ZC702 board provides features common to many embedded processing systems, including DDR3 component memory, a tri-mode Ethernet PHY, general purpose I/O, and two UART interfaces.

ZC702 board Features:

- Zynq XC7Z020-1CLG484C device
- 1 GB DDR3 component memory (four 256 Mb x 8 devices)
- 128 Mb Quad SPI flash memory
- USB 2.0 ULPI (UTMI+ low pin interface) transceiver
- Secure Digital (SD) connector
- USB JTAG interface using a Digilent module
- Clock sources:
 - Fixed 200 MHz LVDS oscillator (differential)
 - I2C programmable LVDS oscillator (differential)

- Fixed 33.33 MHz LVCMOS oscillator (single-ended)
- Ethernet PHY RGMII interface with RJ-45 connector
- USB-to-UART bridge
- HDMI codec
- I2C bus
- I2C bus multiplexed to:
 - Si570 user clock
 - ADV7511 HDMI codec
 - M24C08 EEPROM (1 kB)
 - 1-To-16 TCA6416APWR port expander
 - RTC-8564JE real time clock
 - FMC1 LPC connector
 - FMC2 LPC connector
 - PMBUS data/clock
- Status LEDs:
 - Ethernet status
 - Power good
 - FPGA INIT
 - FPGA DONE
- User I/O:
 - Two programmable logic (PL) user pushbuttons
 - PL user DIP switch (2-pole)
 - Eight PL user LEDs
 - Two processing system (PS) pushbuttons shared with PS 2-pole DIP switch
 - Two PS user LEDs
 - Dual row Pmod GPIO header
 - Single row Pmod GPIO header
- SoC PS Reset Pushbuttons:
 - SRST_B PS reset button
 - POR_B PS reset button
- Two VITA 57.1 FMC LPC connectors
- Power on/off slide switch
- Power management with PMBus voltage and current monitoring via TI power controllers
- Dual 12-bit 1 MSPS XADC analog-to-digital front end
- Configuration options:

- Quad SPI flash memory
- USB JTAG configuration port (Digilent module)
- Platform cable header JTAG configuration port
- 20-pin PL PJTAG header
- 20-pin PS JTAG header

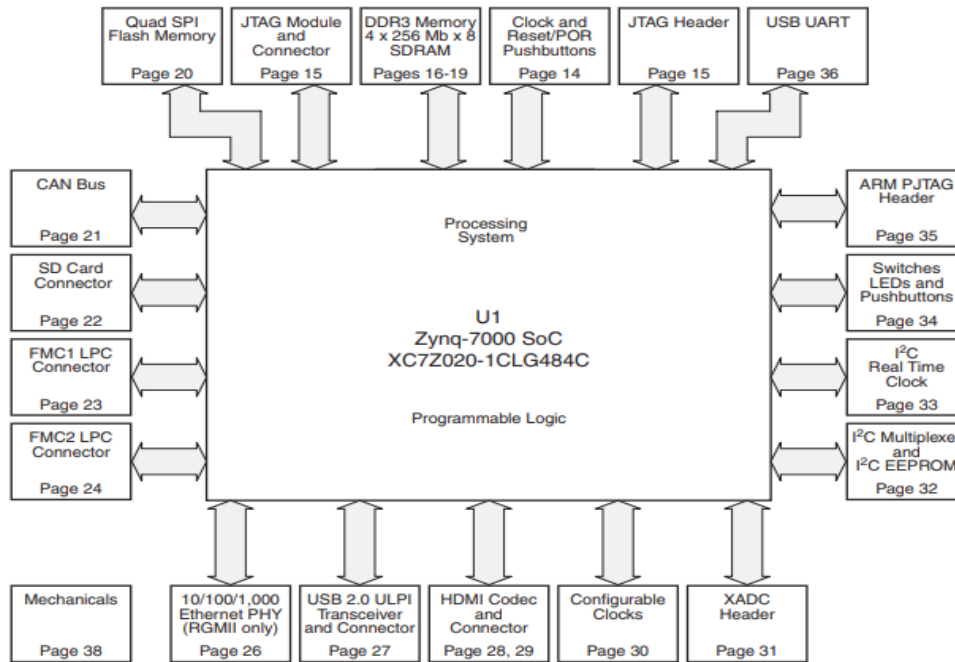


Figure 71: Block Diagram of ZC702

Feature Description

Zynq-7000 XC7Z020 SoC: The ZC702 board is populated with the Zynq-7000 XC7Z020-1CLG484C SoC. The XC7Z020 SoC consists of an SoC-style integrated processing system (PS) and programmable logic (PL) on a single die.

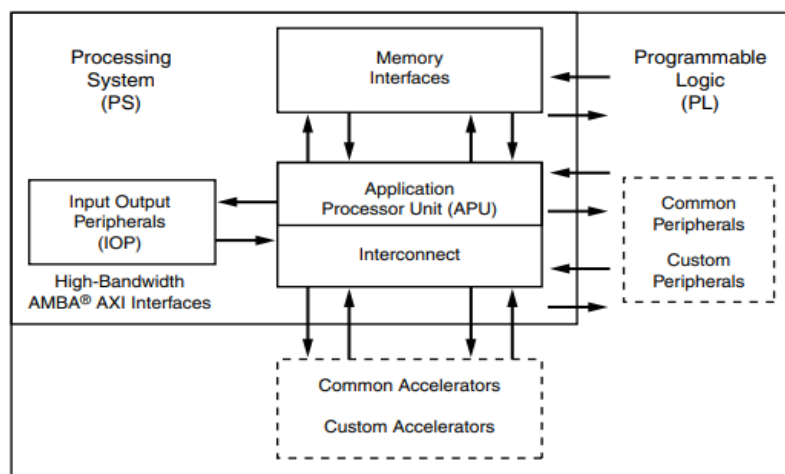


Figure 72: high level Block Diagram

The PS integrates two Arm® Cortex™-A9 MPCore™ application processors, AMBA® interconnect, internal memories, external memory interfaces, and peripherals including USB, Ethernet, SPI, SD/SDIO, I2C, CAN, UART, and GPIO. The PS runs independently of the PL and boots at power-up or reset

Device Configuration

Zynq-7000 XC7Z020 SoC uses a multi-stage boot process that supports both a non-secure and a secure boot. The PS is the master of the boot and configuration process. For a secure boot, the PL must be powered on to enable the use of the security block located within the PL, which provides 256-bit AES and SHA decryption/authentication.

The ZC702 board supports these configuration options:

- ✚ PS Configuration: Quad SPI flash memory
- ✚ PS Configuration: Processor System Boot from SD Card (J64)
- ✚ PL Configuration: USB JTAG configuration port (Digilent module)
- ✚ PL Configuration: Platform cable header J2 and flying lead header J58 JTAG configuration ports

The JTAG configuration option is selected by setting SW16 as shown in Table 1

Table 5: Switch SW16 configuration option setting

Boot Mode	SW16.1	SW16.2	SW16.3	SW16.4	SW16.5
JTAG mode ⁽¹⁾	0	0	0	0	0
Independent JTAG mode	1	0	0	0	0
Quad SPI mode	0	0	0	1	0
SD mode	0	0	1	1	0
MIO configuration pin	MIO2	MIO3	MIO4	MIO5	MIO6

5.9.2. USB 2.0 ULPI Transceiver

Introduction

The USB controller is capable of fulfilling a wide range of applications for USB 2.0 implementations as a host, a device, or On-the-Go. Two identical

controllers are in the Zynq-7000 device. Each controller is configured and controlled independently. The USB controller I/O uses the ULPI protocol to connect external ULPI PHY via the MIO pins. The ULPI interface provides an 8-bit parallel SDR data path from the controller's internal UTMI-like bus to the PHY. The ULPI interface minimizes device pin count and is controlled by a 60 MHz clock output from the PHY.

USB is a cable bus that supports data exchange between a host device and a wide range of computer peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals remain operational.

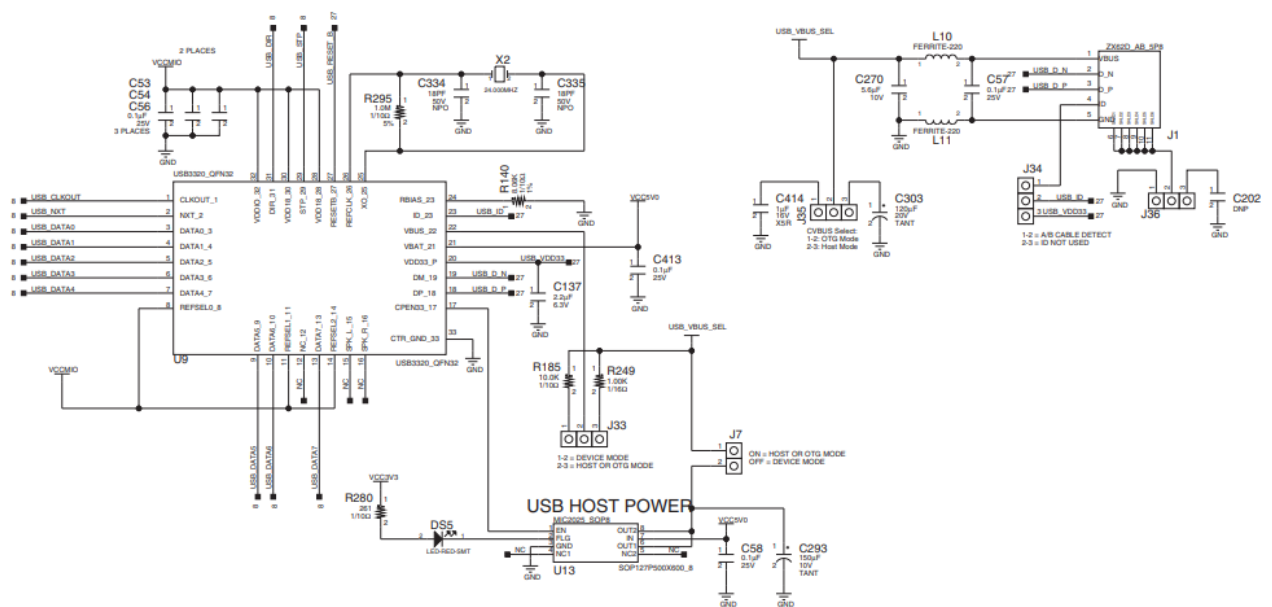


Figure 73: USB 2.0 ULPI Transceiver

The ZC702 board uses a Standard Microsystems Corporation USB3320 USB 2.0 ULPI Transceiver at U9 to support a USB connection to the host computer. A USB cable is supplied in the ZC702 Evaluation Kit (Standard-A connector to host computer, Mini-B connector to ZC702 board connector J1). The USB3320 is a high-speed USB 2.0 PHY supporting the UTMI+ low pin interface (ULPI) interface standard. The ULPI standard defines the interface between the USB controller IP and the PHY device which drives the physical USB bus. Use of the ULPI standard reduces the interface pin count between the USB controller IP and the PHY device.

The interface to the USB3320 transceiver is implemented through the IP in the XC7Z020 SoC Processor System.

The controller interfaces to the PS system memory on one side and an external ULPI PHY device on the USB side. A block diagram is shown in **Figure**

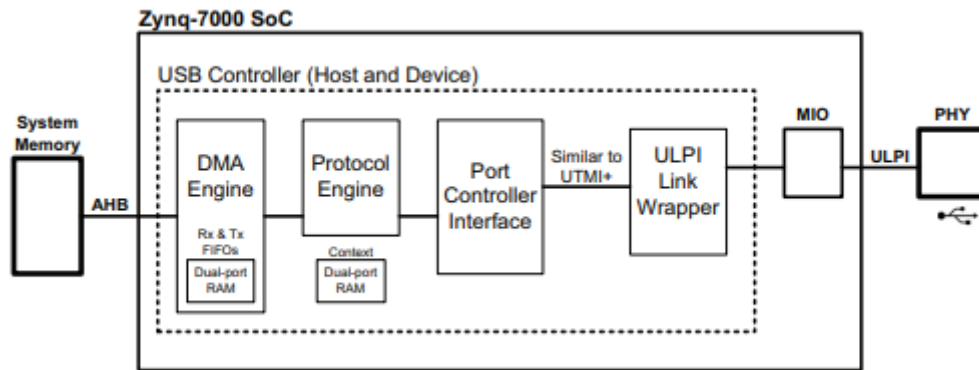


Figure 74: USB Controller Block Diagram

Difference between USB and ULPI:

USB defines the external interface (physical, electrical, various layers of signaling).

The PHY (physical interface circuitry) that presents USB interfaces also has to interface to the host computer. This is done using a UTMI interface.

ULPI is a lower pin-count version of that internal interface. This is beneficial for smaller and lower-cost devices.

Announced on March 1, 2004, the ULPI specification provides a low-pin, low-cost, small form-factor transceiver interface for any USB application.

5.9.3. SD Card Interface

Secure Digital, officially abbreviated as SD, is a proprietary non-volatile memory card format developed by the SD Card Association (SDA) for use in portable devices.

The standard was introduced in August 1999 by joint efforts between SanDisk, Panasonic (Matsushita Electric) and Toshiba as an improvement over Multimedia Cards (MMC), and has become the

industry standard. The three companies formed SD-3C, LLC, a company that licenses and enforces intellectual property rights associated with SD memory cards and SD host and ancillary products.

The companies also formed the SD Association (SDA), a non-profit organization, in January 2000 to promote and create SD Card standards. SDA today has about 1,000 member companies. The SDA uses several trademarked logos owned and licensed by SD-3C to enforce compliance with its specifications and assure users of compatibility.

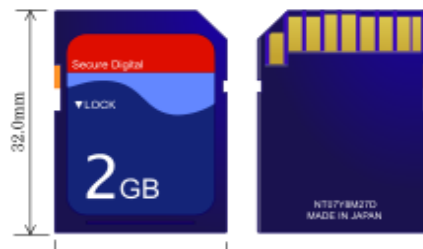


Figure 75: SDI/O

The ZC702 board includes a secure digital input/output (SDIO) interface to provide user-logic access to general purpose nonvolatile SDIO memory cards and peripherals.

The SDIO signals are connected to XC7Z020 SoC PS bank 501 which has its VCCMIO set to 1.8V. A TXB02612 SDIO port expander with voltage-level translation (U61) is used between the XC7Z020 SoC and the SD card connector (J64).

The Figure below shows the connections of the SD card interface on the ZC702 board.

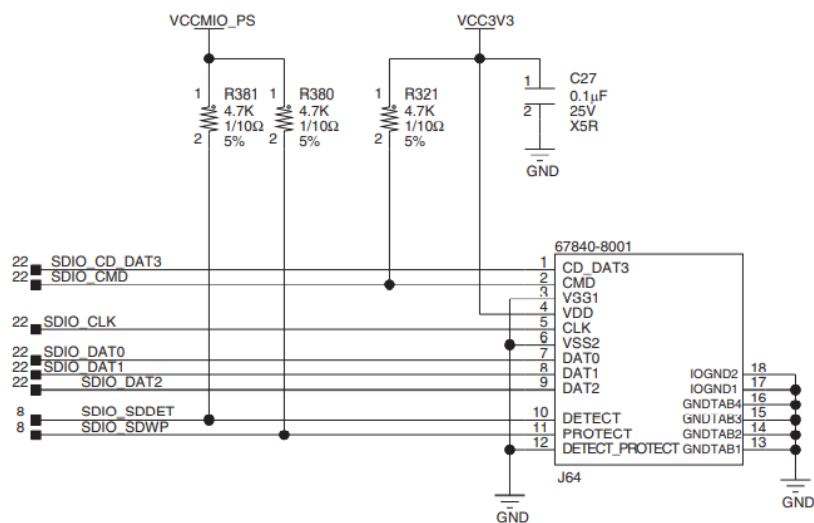


Figure 76: SD card interface

SD card

Problem: the SD port of laptop didn't sense SD card.



Figure 77: SD port

Solution: after search we find that there is another way to connect sd card by using usb reader to connect it to usb Port instead of connection directly by sd port (it is recommended to connect sd card directly by sd card so if you faced this problem you can try to use another sd card or another laptop to ensure that the problem doesn't in the sd port)

Configuring the Board for SD Card Boot

To boot the board from an SD card, you need to physically change either a switch or jumper settings on the board. This section describes settings for a ZC702 board

Problem: when we try to execute the application at the Linux prompt through typing this path: /mnt/ name_of_project.elf. We didn't receive any response.

Solution: after search we find that we should boot up the board using sd mode. Boot Modes: The following table can be used to determine mode switch configuration (sd mode).

Table 6: Configuring the Board for SD Card Boot

Boot Mode	Setting	SW16 (SW16.1-SW16.5)
JTAG	00000	OFF, OFF, OFF, OFF, OFF
Independent JTAG	10000	ON, OFF, OFF, OFF, OFF
QSPI	00010	OFF, OFF, OFF, ON, OFF
SD	00110	OFF, OFF, ON, ON, OFF

- Identify whether you have to change a jumper or a switch.

For Revision D and newer boards:

DIP switch SW16 (light blue/grey color) positions 3 and 4 should be set to 1.

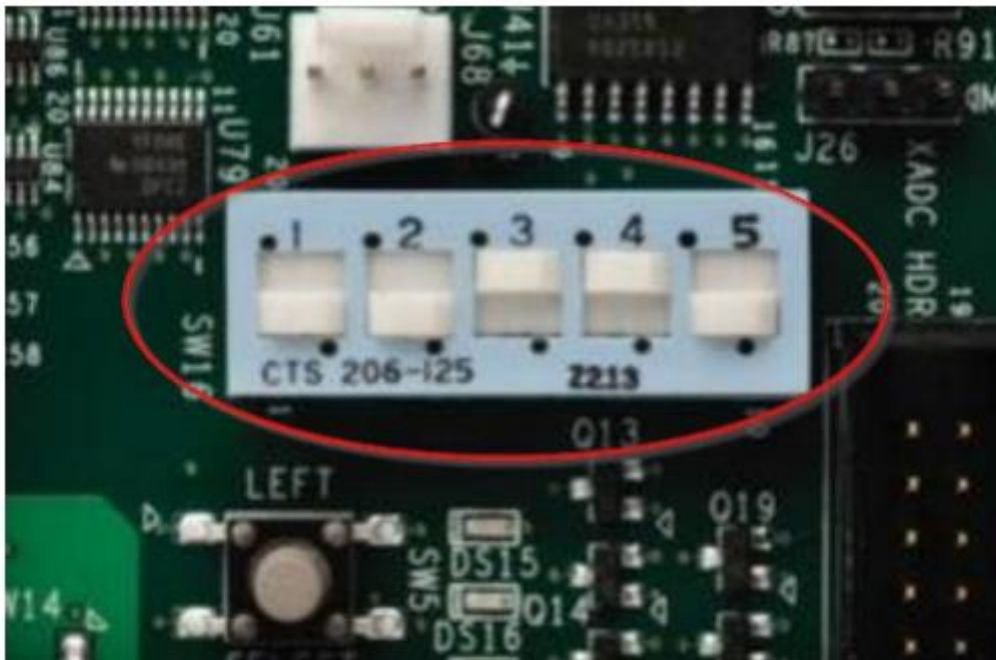


Figure 78: SW 16 in ZC702

Another problem: after determining the sw16 we typed the same path and this one there was response but can't define the command

Solution: you should activate sw1 (press SW1 (POR_B))before type the path to reinitialize board configuration



Figure 79: SW1 in ZC702

5.9.4. USB-to-UART Bridge

The ZC702 board contains a Silicon Labs CP2103GM USB-to-UART bridge device (U36) which allows a connection to a host computer with a USB port. The USB cable is supplied in the ZC702 Evaluation Kit (Standard-A end to host computer, Type Mini-B end to ZC702 board)

connector J17). The CP2103GM is powered by the USB 5V provided by the host PC when the USB cable is plugged into the USB port on the ZC702 board.

Silicon Labs provides royalty-free Virtual COM Port (VCP) drivers for the host computer. These drivers permit the CP2103GM USB-to-UART bridge to appear as a COM port to communications application software (for example, TeraTerm or HyperTerm) that runs on the host computer.

Connecting the Board to a Serial Terminal

To connect a ZC702 board to a serial terminal you need a mini USB cable to connect the UART port on the board to the computer where you run a serial terminal. There is a serial terminal available as part of the SDSoC IDE (tab labeled Terminal 1 at bottom of screen).

1. Connect the mini-USB cable to the UART port.



Figure 80: UART PORT in ZC702

2. Set up the serial terminal (for example, puTTY, minicom, or the SDSoC environment terminal):
 - Set the baud rate to 115200 baud.
 - In Windows, set the serial port to COMn, where n is a number and can be found as follows:
 - Select Start > Computer then right-click Properties.
 - Select Device Manager and open Ports (COM & LPT).

- Use the COM port labeled Silicon Labs CP210x USB to UART Bridge.
 - ✚ If the right COM port does not appear on the Terminal Settings window, make sure the board is connected to the USB port and turned on. Restart the SDSoc environment by selecting File > Restart and the COM port should appear on the list.

Problem: the serial port wasn't defined (we use serial communication).



Solution: download and setup drivers from siliconlabs to define the port. (CP210x USB to UART Bridge VCP Drivers - Silicon Labs).

Click the icon to open the settings. 



Figure 81: serial terminal setting

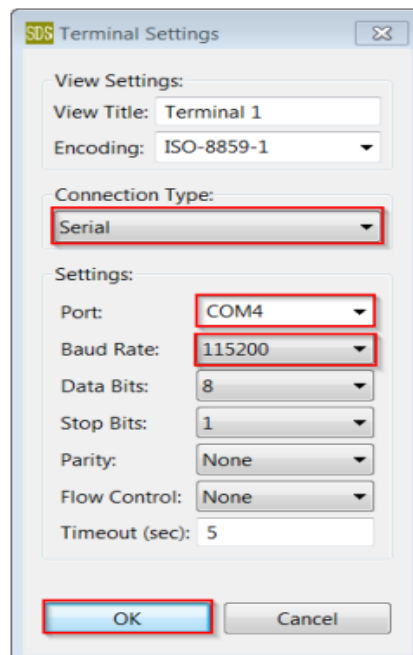


Figure 82: terminal setting in SDx

3. Power on the board

The board needs to be powered on at least once with the mini USB cable connected for Windows to recognize the UART and install the driver. You might need to power cycle the board.

Executing a Pre-built Application

To run the pre-built application on the ZC702 board, follow these steps:

- 1) Copy the contents of the sd-card-prebuilt folder to the root folder of an SD card
- 2) Insert the card into the SD card slot of the ZC702 board.
- 3) Confirm jumpers or switches are set to boot from the SD card as shown above
- 4) Set up a serial terminal.
- 5) With the SD card inserted and cables connected, power up the board and start the serial terminal session.

You should see the Done LED turn green and Linux booting

- 6) At the prompt, type `cd /mnt`

This takes you to the SD card folder containing the application ELF file

- 7) To run the application ELF, type: `./mmult.elf`
- 8) The application displays information about the run and the results of the matrix multiplication.

You see output similar to that shown below

```
root@xilinx-zc702-2018_3:~# /mnt/examplee.elf
Testing 1024 iterations of 32x32 floating point mmultadd...
Average number of CPU cycles running mmultadd in software: 185721
Average number of CPU cycles running mmultadd in hardware: 23101
Speed up: 8.03952
TEST PASSED
root@xilinx-zc702-2018_3:~#
```

Figure 83: results after marking



Chapter 6

Results

6.1 Raspberry pi

The output of the Raspberry pi: In case the detection of straight lane lines
Here we implement the first algorithm on Raspberry pi which is detection of straight lane lines without the detection of the curved ones.

Here we record the percentage of usage of the algorithm while processing on raspberry pi processor. We also record the RAM utilization in order to prove that the processing on digital platforms results in much better performance.

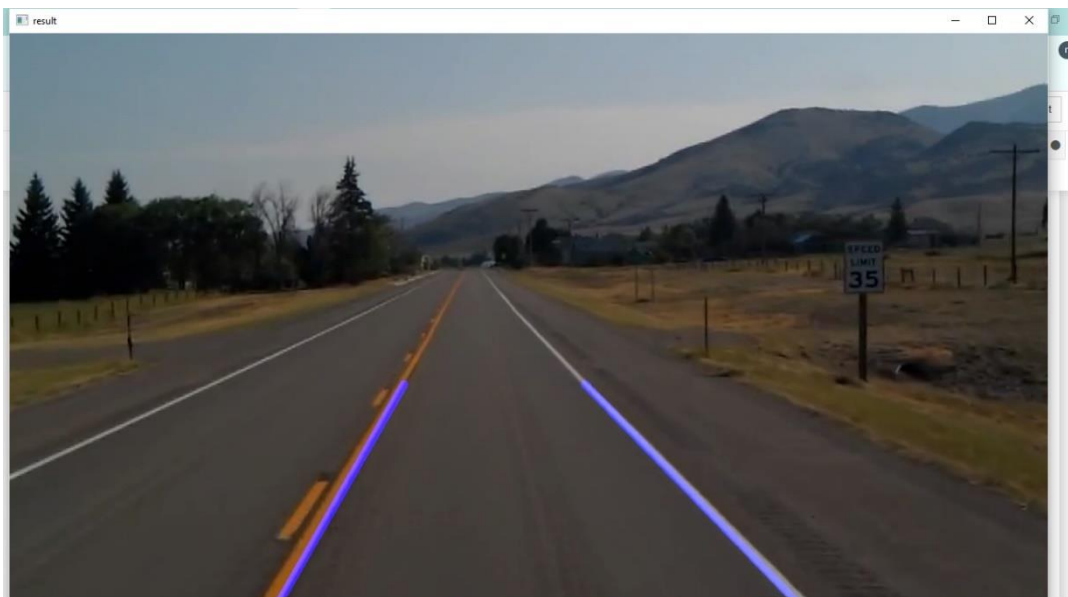


Figure 84: output 1

```
Info: ky 1.11.6 - Linux nancyyy 5.10.1
Uptime: 1h 2m 39s
Frequency (in MHz): 1500
Frequency (in GHz): 1.50
RAM Usage: 513 MiB/1.78 GiB - 28%
Swap Usage: 0 B/100 MiB - 0%
CPU Usage: 70%
Processes: 168 Running: 2

File systems:
/ 4.68 GiB/14.4 GiB
Networking:
Up: 0 B - Down: 0 B

Name      PID   CPU%  MEM%
python3   3279  40.56  8.33
vncserver-x11-c  538  13.01  1.84
Xorg      563   9.69   6.46
lxpanel   802   6.12   3.60
```

Figure 85: performance of raspberry pi in case 1

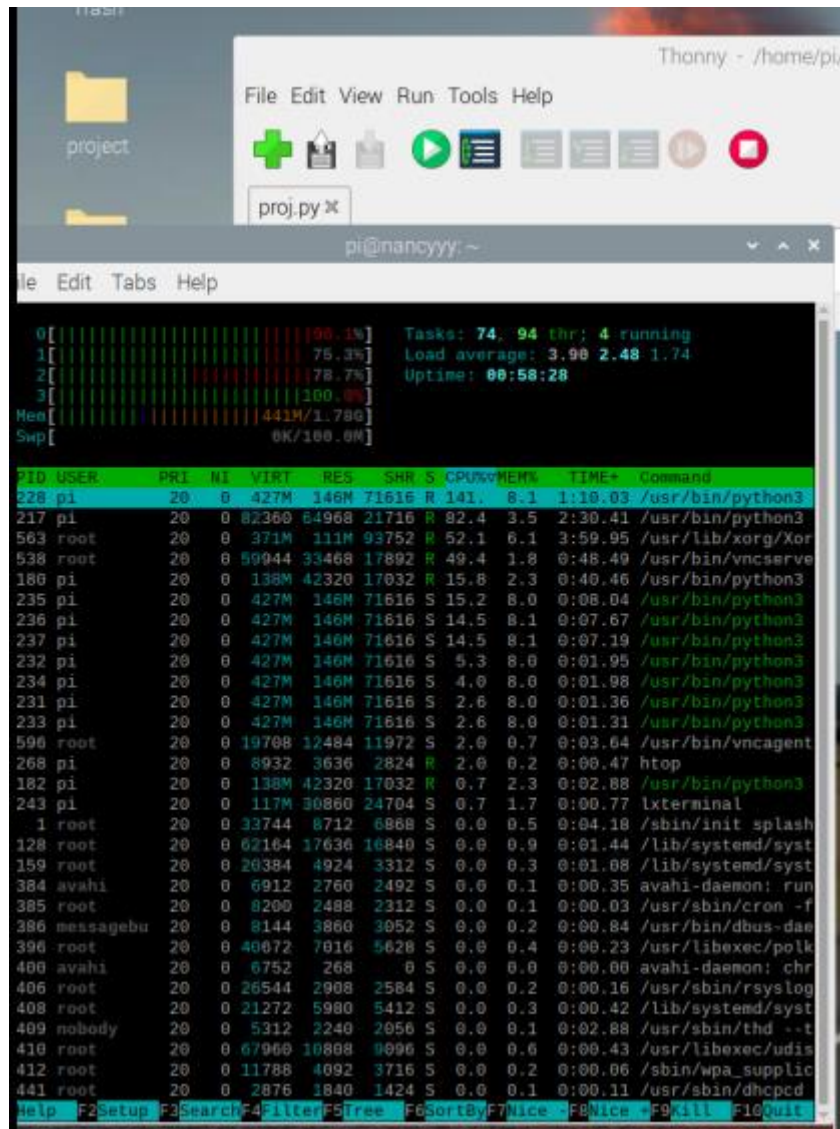


Figure 86: performance of raspberry pi in case 1

The output in case the detection of straight and curved lane lines

Here we implement the second algorithm on Raspberry pi which is detection of straight and curved lane lines.

Here we record the percentage of usage of the algorithm while processing on raspberry pi processor. We also record the RAM utilization in order to prove that the processing on digital platforms results in much better performance.



Figure 87: output 2

```

Info: 1.11.6 - Linux nancyyy 5.10.103-
Uptime: 1h 39m 32s
Frequency (in MHz): 1500
Frequency (in GHz): 1.50
RAM Usage: 952 MiB/1.78 GiB - 52% ██████████
Swap Usage: 0 B/100 MiB - 0% ██████████
CPU Usage: 34% ██████████
Processes: 191 Running: 1

File systems:
/ 4.70 GiB/14.4 GiB ██████████
Networking:
Up: 0 B - Down: 0 B

Name          PID    CPU%  MEM%
python3.9     2964   30.36 10.22
vncserver-x11-c 510    2.04  1.84
ffmpeg       2992   2.04  7.88
Xorg         535    1.53  6.00
  
```

Figure 88: performance of curvature

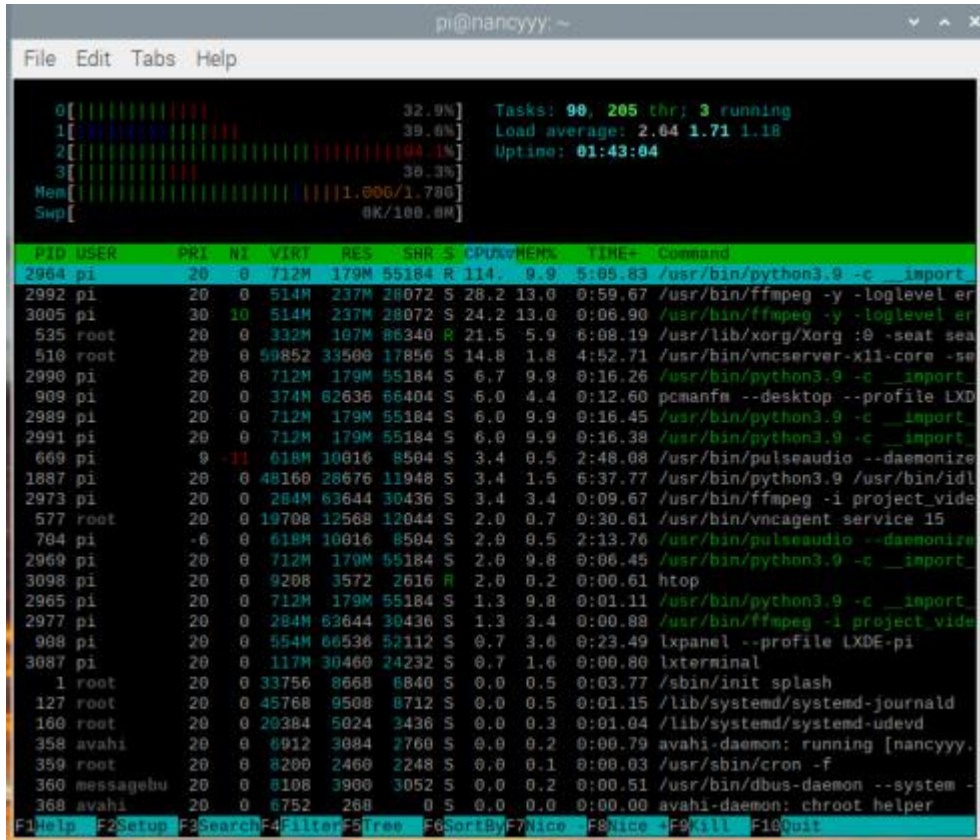


Figure 89: performance 2 of curvature

Here is The output performance in case the detection of straight and curved lines on our pc intel core i7

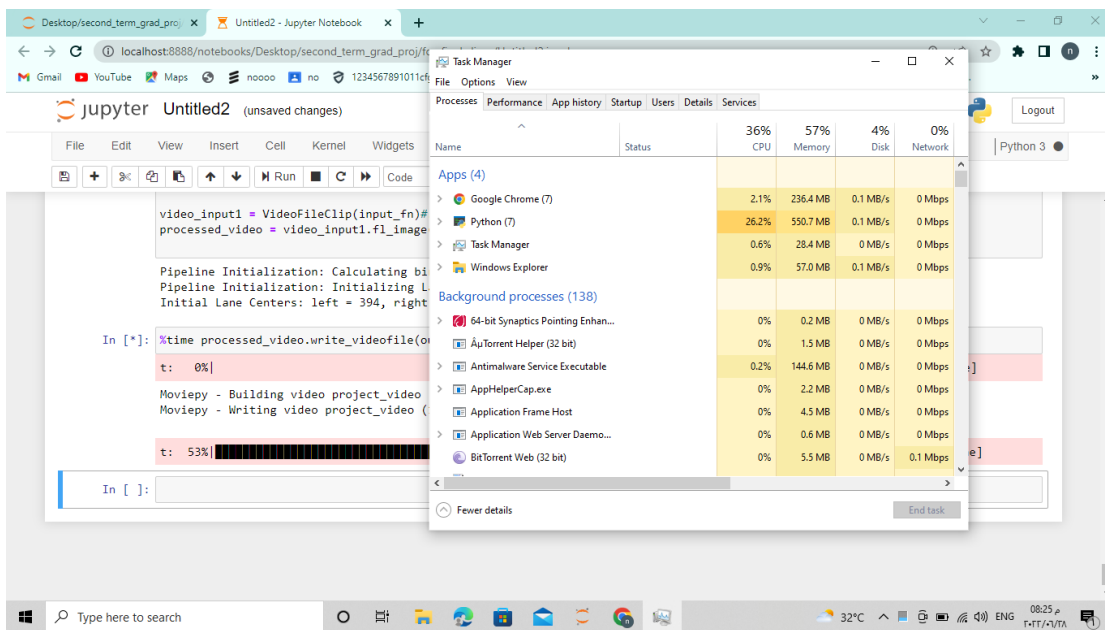


Figure 90: performance on pc

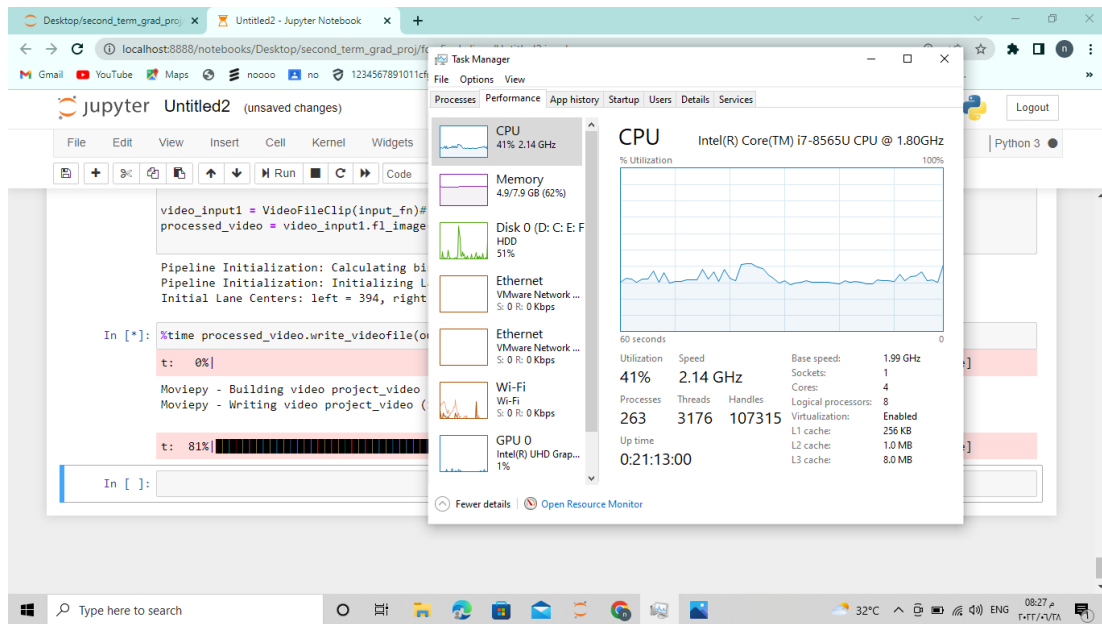


Figure 91: performance 2 on pc

6.2 Vivado HLS

Here is the output of the gray code of the Vivado HLS

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	83
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	15
Register	-	-	10	-
Total	0	1	10	98
Available	280	220	106400	53200
Utilization (%)	0	~0	~0	~0

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.283	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1	1	1	1	none

Figure 92: Reports of grayscale code on Vivado

6.3 SDSOC

Table 7: xf::Sobel resource usage

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4
FIFO	-	-	-	-
Instance	3	-	1295	1975
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	5	-
Total	3	0	1300	2021
Available	280	220	106400	53200
Utilization (%)	1	0	1	3

Table 8: xf::absdiff resource usage

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4
FIFO	-	-	-	-
Instance	0	-	474	1019
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	5	-
Total	0	0	479	1065
Available	280	220	106400	53200
Utilization (%)	0	0	~0	2

Table 9: xf::Threshold resource usage

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4
FIFO	-	-	-	-
Instance	0	-	475	980
Memory	-	-	-	-
Multiplexer	-	-	-	33
Register	-	-	21	-
Total	0	0	496	1017
Available	280	220	106400	53200
Utilization (%)	0	0	~0	1

Table 10: xf::bitwise or resource usage

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4
FIFO	-	-	-	-
Instance	0	-	394	942
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	5	-
Total	0	0	399	988
Available	280	220	106400	53200
Utilization (%)	0	0	~0	1

Table 11: Xf::wrap Transform resource usage

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4
FIFO	-	-	-	-
Instance	676	61	43587	28038
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	5	-
Total	676	61	43592	28084
Available	280	220	106400	53200
Utilization (%)	241	27	40	52

Table 12: SDSoC ZC702 processor + hardware time analysis.

Function	operand Number of Frames	Standby time (Software)	Standby time (Equipment)	Cooldown reduction rate - Performance-
Front Processing (perspective irreversible)	one	157.07 ms	113.76ms	28% reduction
(fps)	one	(6.37fps)	(8.79fps)	(1.38 times the performance gain)

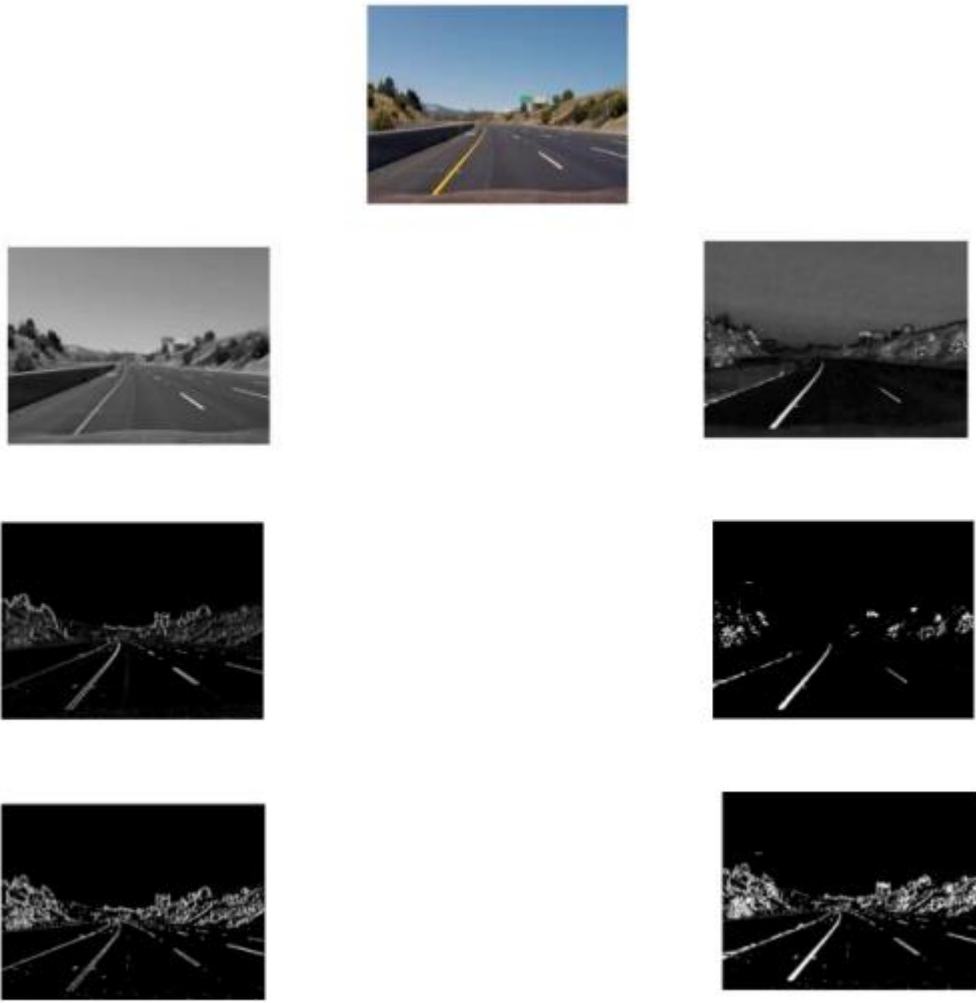


Figure 93: preprocessing stages



Figure 94: perspective transform

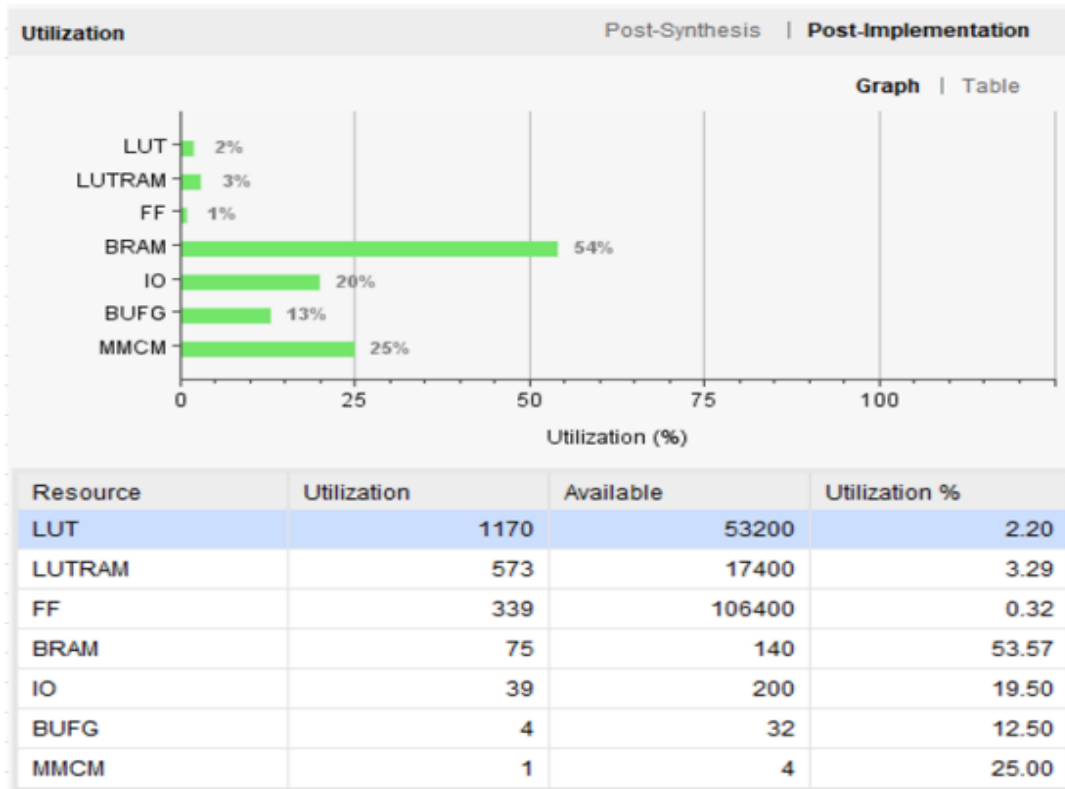


Figure 95: space usage in the system

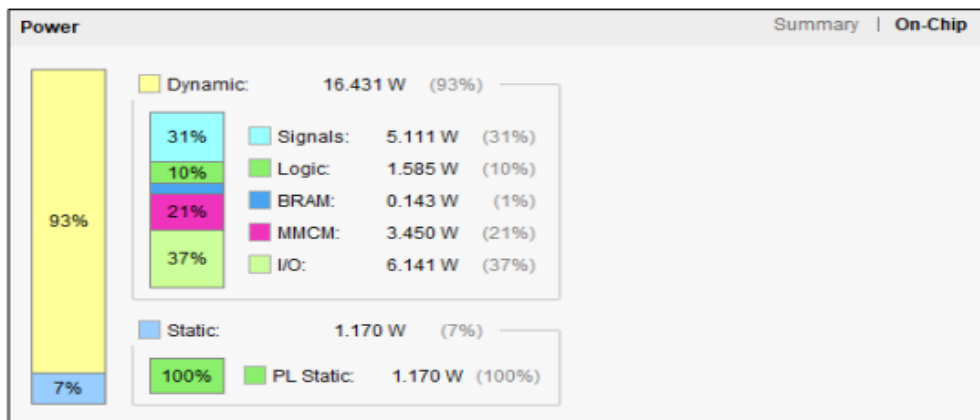


Figure 96: power consumption result in the system

Function	operand Frame Number	Standby time
Pre-Processing	one	157.07 ms
All Software	one	401.14 ms
Preprocessing/AllSoftware	-	39.15%

Figure96:Time analysis of the SW

Chapter 7

Conclusion and Future work

Conclusion

As discussed in the previous chapters we compress the years of the design in few months and overcome the complexity of the design using Verilog by using SDSOC tool and SOC. We use computer vision techniques in order to detect the street lane lines to try decreasing car accidents. We focused on determining the best accuracy in our project. This is the reason why we use software/hardware co-design. We save power and number of cycles which will save more lives due to the faster decisions and save power of millions of cars daily. We also find that the performance of the normal processors has lower efficiency than using digital design.

Future work:

We choose using the xfOpenCV library to decrease the time but it spends more time to solve its problems with SDSOC so, the future work must be try to implement code without using any library and get the results or using Xilinx new tool vitis which support AI, cloud computing and all c++ & python libraries.

We also aim to use deep learning techniques to try to increase the accuracy and robustness and also to be able to detect any lane line in any other street.

Chapter 8

References

- [1] Hassan Mostafa, Implementation of deep neural networks on FPGA-CPU platform using Xilinx SDSOC ,2020
- [2] Hassan Mostafa, Design and Implementation of Authenticated Encryption Co-Processors for Satellite HardwareSecurity ,2020
- [3] Xilinx, SDSOC user guide,2018
- [4] Xilinx, Vivado Design Suite Tutorial,2014
- [5] Xilinx, ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC User Guide, 2018
- [6] Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan and Joseph Walsh. " Deep Learning vs. Traditional Computer Vision",Ireland
- [7] Yang Xing, Chen Lv, Member, IEEE, Long Chen, Huaji Wang, Hong Wang, Dongpu Cao, Member, IEEE,Efstathios Velenis, Fei-Yue Wang, Fellow, IEEE, Advances in Vision-Based Lane Detection.
- [8] Xilinx. ug1233-xilinx-opencv-user-guide,2019
- [9] Xilinx, ug1085-zynq-ultrascale-trm,2019
- [10] Xilinx, ug902-vivado-high-level-synthesis,2017
- [11] Xilinx, ug1028-sdsoc-intro-tutorial 2016
- [12] Adam Ziębiński, Automotive Production Engineering UnifiedPerspective based on Data Mining Methods and Virtual Factory Model. 15 april 2019
- [13] xilinx, ug585-Zynq-7000-TRM,2021
- [14] xilinx, ug821-zynq-7000-swdev.2015
- [15] xilinx. ug850-zc702-eval-bd.2018
- [16] xilinx. ug1028-sdsoc-getting-started.2018
- [17] Xilinx,Zynq-7000 SoC Technical Reference Manual,2021
- [18] Xilinx, Zynq-7000 All Programmable SoC Software Developers Guide,2015
- [19] Hardware Acceleration of Computer Vision Application
- [20] Xilinx, Zynq-7000 All Programmable SoC: Concepts, Tools, and

Techniques (CTT),2012

[21] Xilinx, ug1027-intro-to-sdsoc, 2018

[22] ug1144-petalinux-tools-reference-guide, 2022

[23] Xilinx, ug1282-sdsoc-debugging-guide, 2018

[24] Xilinx, ug1235-sdsoc-optimization-guide, 2018

[25] Basic HLS Tutorial, 2018

[26] Qian Xu, Srenivas Varadarajan IEEE, A Distributed Canny Edge Detector: Algorithm and FPGA Implementation, 2014

[27] Xu Y.Shan X.Chen B.Y.Chi C.Lu Z.F.Wang Y.Q. 1College of Mechatronics and Control Engineering, Shenzhen University, Shenzhen, 518060, China, A Lane Detection Method Combined Fuzzy Control with RANSAC Algorithm,2015

[28] Erke Shang & Xiangjing An, A real-time lane departure warning system based on FPGA,2013

[29] Adaptive Lane Keeping Assistance System design based on driver's behavior

Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision,2018

[30] Jyun-Guo Wang a, Cheng-Jian Lin b,*, Shyi-Ming Chen a, Applying fuzzy method to vision-based lane detection and departure warning system, 2010.

[31] Wang Ze, Weiqiang Ren and Qiang Qiu, "LaneNet: Real-Time Lane Detection Networks for Autonomous Driving,"ArXiv, abs/1807.01726 (2018).

[32] Keerti Chand Bhupathi and Hasan Ferdowsi," Northern Illinois USA", " An Augmented Sliding Window Technique to Improve Detection of Curved Lanes in Autonomous Vehicles", November 2020.

[33] Hardware Acceleration for Machine Learning

[34] HG Zhu, Beijing." An Efficient Lane Line Detection Method Based on Computer Vision",China, 2021.

[35] Ahmed Mahmoud, Loay Ehab, Mohamed Reda, Mostafa Abdelaleem, Hossam Abd El Munim, Maged Ghoneima , M. Saeed Darweesh and Hassan Mostafa," Real-Time Lane Detection-Based Line Segment Detection", 2018.

[36] Qian Xu, Srenivas Varadarajan, Chaitali Chakrabarti, Fellow, IEEE, and Lina J. Karam, Fellow, IEEE, "A Distributed Canny Edge Detector: Algorithm and FPGA Implementation", JULY 2014.

[37] Wei Wang, Hui Lin and Junshu Wang^{3,4}, CNN based lane detection with instance segmentation in edge-cloud computing.

[38] Muthukrishnan.R¹and M.Radha²," Edge detection techniques for image segmentation", Coimbatore, Dec 2011.

[39] Xu Y.Shan X.Chen B.Y.Chi C.Lu Z.F.Wang Y.Q.1," A Lane Detection Method Combined Fuzzy Control with RANSAC Algorithm", China.

[40] Ammu M Kumar and Philomina Simon," Review of Lane Detection and tracking algorithms in advanced assistance driving system", August 2015.