Hardware Acceleration of Real-time Image Processing for Vehicle Control

Yuranan Kitrungrotsakul

School of Information Science
Japan Advanced Institute of Science and Technology
September, 2017

Master's Thesis

Hardware Acceleration of Real-time Image Processing for Vehicle Control

1510204 Yuranan Kitrungrotsakul

Supervisor : Associate Professor Kiyofumi Tanaka
Main Examiner : Associate Professor Kiyofumi Tanaka
Examiners : Professor Mineo Kaneko
Professor Yasushi Inogushi

School of Information Science
Japan Advanced Institute of Science and Technology

August, 2017 (submitted)

Abstract

in this research, we consider the methods for using hardware components in embedded system board to accelerate the high computational intensity program in the embedded system. The purpose of this research is to give the alternative ways to accelerate program in embedded system board. Moreover, the hardware accelerator must be implemented under the resource constraints. We propose the two base types of hardware accelerator. Firstly, the hardware accelerator bases on graphic processing unit (GPU). Secondly, the hardware accelerator bases on programmable logic (PL). The lines tracking program is selected as the high computational intensity program for acceleration. The hardware accelerators are experimented in the real hardware embedded system board. The results from the hardware accelerators are verified to the base program. The results demonstrates that both of the GPU and PL base hardware accelerators significantly reduce the execution time of the high computational intensity program.

Acknowledgements

The author would like to express my special thanks of gratitude to my supervisor, Associate Professor Kiyofumi Tanaka, who guided me through the research. I got the precious opportunity to do the research on the interesting topic, which helps the author to know about many new things. I would like to express my gratitude to my minor research supervisor, Professor Hiroyuki Iida, for gave me the opportunity for the minor research topic, which helped me in doing research.

Lastly, I would like to thank my family and friends who always give me a support when I need.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the computers or laptops that we use in our daily life can handle the most of programs. Even for some applications that involve high computational intensity, our computers can execute program without any problem. However, in the small electronic devices such as smart phones, executing high computational intensity programs is not suitable. The reason behind this difference is the processing power. In computers or laptops that we use, mostly, they already contain the hardware accelerator to handle high computational intensity program while embedded systems don't have.

Embedded systems have been developed in recent decades. The development of embedded systems directly affects our daily life. Nowadays the embedded devices exist everywhere around us. The internet of things is one of the most trendy for embedded technologies. Everything around us is going to be smarter and connected. For example, in smart houses, electronic devices and household appliances are connected together. The demand on the embedded system is increasing every single time. The computation complexity of embedded program is increasing but size of the embedded system is preferred to reduce. The direct effect of reducing size is the processing power. The result of the contrast between high computation complexity and processing power is outstanding in image processing devices, which mostly interact with users.

The image processing algorithms are well known as computationally intensive tasks. Its applications are used in order to solve many problems that involve images. However, the major problem of image processing is the trade-off between accuracy and time consumption. In order to achieve high degree of accuracy, the time consumption increases

significantly. The time consumption of computational program depends on the processing power. It can process images immediately in the high performance computers. On the other hand, it takes more time in the embedded system; as a result, the interactive program cannot respond to the users or the environment instantly.

The autonomous car is a vehicle that can navigate and drive without human interference. The vehicle must have the potential to analyse the environment information from the sensors. The environment information is retrieved in various format such as the video from camera, the data from sensor and so on. The enormous of environment information must be processed as fast as possible in order to respond to the environment. In this research,the road surface marking is chosen as a target program for acceleration. The road surface marking is commonly used in the autonomous car for keeping the vehicle to the lane.

This thesis is divided into 6 chapters. Firstly, chapter 1 consists of the introduction. Then, chapter 2 explains the virtual environment system. Chapter 3 describes about the road surface marking program. Chapter 4 talks about the hardware accelerators. Chapter 5 shows the experimentation and analysis. Lastly, chapter 6 is the conclusion of all works.

1.1 Problem Statement

The self driving have been researched for awhile. Many of them involve the road surface marking program such as [1], [2], and [3]. They purpose the methods and applications for fulfilling the capabilities of self driving vehicle in many aspect. However, they do not address about the processing time, which is an important factor in the practical usage. On the other hand, several researchers tried to attempt the hardware accelerators. For example, [4], [5], and [6], accelerated the system by the hardware accelerators. Their purposed hardware accelerators successfully accelerate their application but only in terms of theoretical.

The most important characteristic of real-time computing is response time of tasks. It must guarantee that the response time should be within some deadline. Thus, more computation resources should be used in order to meet the time constraint. However, accuracy must be acceptable for the given application. In some cases, when reducing

accuracy to minimum, the time consumption still exceeds the time limitation. Thus, the acceleration techniques are used to achieve the timing required while accuracy remains acceptable [7]. However, each acceleration technique has its constraints and drawbacks. Thus, the acceleration techniques must be considered for compatibility and suitability. Personal computers or workstations match many techniques because of their huge computation resources. On the other hand, embedded systems lack computation resources and cannot support many of those techniques. GPU, is a common hardware accelerator that suits many cases [8]. However, there are a lot of systems that dont contain GPU. Thus, the acceleration techniques must be developed for such exceptional cases [9].

From those problems, the aim of this research is to implement the hardware accelerators for the road surface marking program on virtual environment. The hardware accelerator should be able to accelerate the road surface marking program while remain the capable of tracking. The road surface marking program is executed in the virtual environment to simulate the environment information. In order to solve the problem, this research purposes the methods for the hardware accelerators. Two bases of the hardware accelerators are GPU base hardware accelerator and PL base hardware accelerator. we purpose the virtual environment system which is implemented to simulate the real environment.

Chapter 2

Virtual Environment System

The process of developing the embedded system requires the real embedded devices to operate the applications. However, the development under the real circumstance is not appropriate in some applications, especially, for the system that interacts with human. The errors from slight mistake can threaten someone's life.

Virtual prototyping is one of the methods to simulate the real environment. The real environment of the target embedded system is simulated by hardware and software. By doing this, the system can be developed even if the system is not operated under the real circumstance.

2.1 Virtual Environment System Flow

The virtual environment system is constructed by the combination of hardware components and software programs. The purpose of the virtual environment is to simulate the real environment for being used in the experimental. The virtual system imitates the work flow from the real system.

Figure 2.1 shows the virtual environment flow of this research. After the start signal is assigned, The system initializes all hardware components and software programs. Then, the system waits for the initialization step. After the initialization step is done, the virtual environment is generated. Then, the image processing algorithm is applied with the hardware accelerator. After that, the vehicle commands for navigation are generated. After that, the vehicle responds to the virtual system according to the vehicle commands.

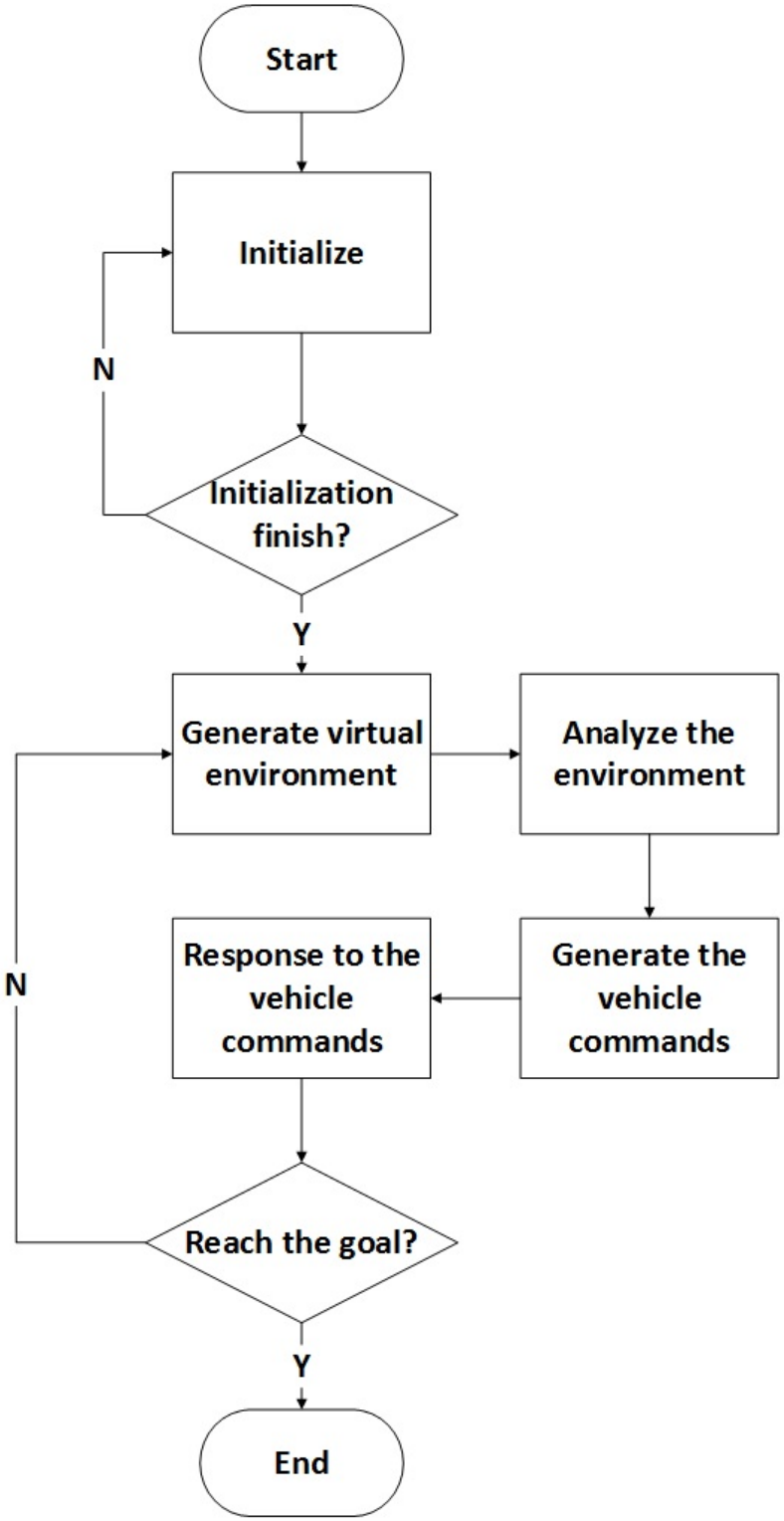

Figure 2.1: A program flow of virtual prototyping environment

Lastly, the system loops the processes from the generating virtual environment to the responding to the vehicle commands until the vehicle reaches the destination.

2.2 Hardware Components of Virtual Environment System

The virtual environment consists of 4 major hardware components that are a windows computer, a hdmi capturing device, an image sense module, and the virtual driving controller. The connections of 4 components are shown in figure 2.2.

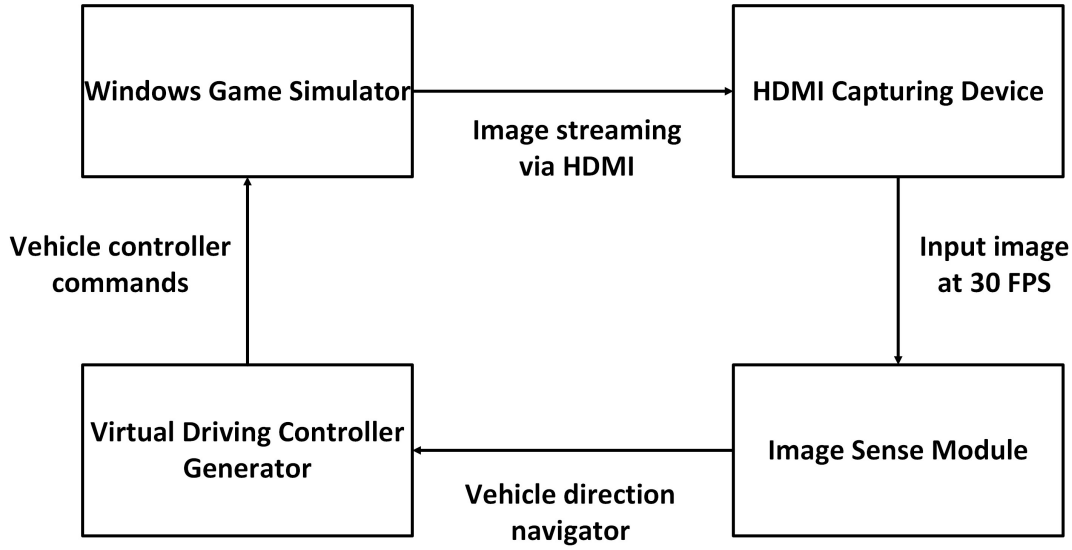

Figure 2.2: Hardware Components in Virtural Environment System

2.2.1 A Windows Game Simulator

The purpose of this module is to generate the virtual input images of the system and respond to the commands from the virtual driving controller generator. The image sense module requires an input image to operate the image processing. Generally, in image processing programs, the source of input can be prepared in any format such as a picture, a recorded video, and a streaming image.

Table 2.1: The comparison of input format types for an image sense module

Format Type	Continuity	Interaction
Image	Lacks of continuity	Cannot respond
Video	Capable of continuity	Cannot respond
Image Streaming	Capable of continuity	Respond to the commands

The realistic of virtual prototyping depends on the input format. Firstly, the image input format can be used for verification of the algorithm of image processing program. However, it is the most unrealistic format type because the program analyses an input one by one. It lacks continuation that occurs in the real environment. Secondly, the recorded video format achieves the continuation that is not achieved by the image input format. The image frames from video can be analysed continuously but the generated commands cannot affect the video because the recorded video is fixed. Lastly, the source of image streaming effects the virtual prototyping. The image streaming from a video camera can be used as an input but it requires a real movement to achieve the responsiveness. On the other hand, the image streaming from the Windows game simulator achieves the responsiveness without the real movement, which is preferred in the virtual prototyping. Table 2.1 shows the summary of the input format type and its properties.

In this research topic, the image sense module must be able to analyse the environment in real time and in the real environment. Thus, the input of this module in real environment is retrieved in a streaming image format. Then, the vehicle must respond to the generated commands immediately.

2.2.2 A HDMI Capturing Device

The data from the Windows game simulator is used in the image sense module. However, The output port of the Windows game simulator is a HDMI port type, which is rarely used in real system. To imitate the real system, the data from HDMI port should be converted to USB port, which is commonly used in the embedded system. Thus, the HDMI port is used as an output port of the Windows game simulator module but it cannot be connected directly to the image sense module. It requires an convertor device to convert the HDMI signal to USB signal to connect two modules together. Febon168 is a HDMI to USB video class grabber. It converts HDMI input signal to USB output

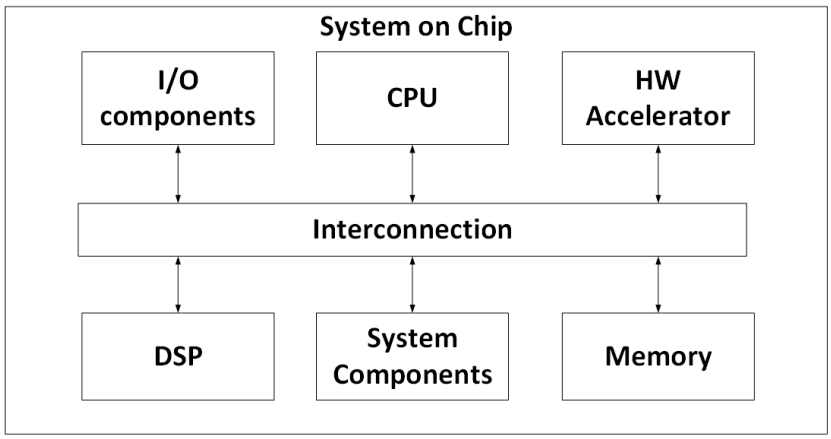

Figure 2.3: An example of system on chip's architecture

signal for video format. It is used as a converter between the Windows game simulator and the image sense module.

2.2.3 An Image Sense Module

The image sense module operates the image processing program to analyse the environment. The input of this module comes from the HDMI capturing device. The output of this module is a direction for vehicle. In this research, there are two major subsystems in the image sense module.

Firstly, the image processing program is the target program for acceleration. The image processing program in this research is a road surface marking. This program is commonly used in the self-driving car. The algorithm of this program is described in Chapter 4.

Another subsystem is the hardware accelerator. In this research, the hardware accelerators were implemented in two based types, which are GPU and PL. Because of the difference architecture, two embedded system boards were used for each based type. Firstly, Jetson Tegra K1 is NVIDIA's embedded Linux development platform system on chip (SoC) that is used in this research for GPU base accelerator. On the other hand, as the PL base accelerator, Zedboard is a complete development kit that uses the Xilinx Zynq-7000 All Programmable SoC architecture.

Both of Jetson and Zedboard are SoC. SoC is an integrated circuit that integrates many

useful computer components together such as CPU, digital signal processing (DSP), and so on. Figure 2.3 shows the example of components in SoC. The performance of system relies on the limitation of processing resources in the image sense module because the image sense module is a bottleneck of system. Generally, the embedded system faces the problem about responsiveness due to its hardware limitation. Jetson and Zedboard face the same problem as the others. In order to solve this problem, the hardware accelerators were implemented in each board to accelerate the image sense module. The hardware accelerators are described in Chapter 5.

2.2.4 A virtual driving controller generator

The purpose of this module is to generate the driving controller commands to the vehicle controller system. The input of this module is the direction information from the image sense module. The output of this module is the vehicle controller commands. This module can be integrated with the image sense module. However, it is separated because it depends on the vehicle system. In this research, the vehicle system is a virtual simulator. Thus, the driving controller commands are generated with the simulator compatible with simulator compatible format.

Chapter 3

Road Surface Marking Program

In order to research the effect of acceleration, we have to choose the target program to be accelerated. The target program should be a computational intensive program because the result from the acceleration is more obvious than the less computational intensive program. In this research, a road surface marking program is chosen to be the target program of acceleration.

3.1 Object Detection

The object detection program is a program that uses the image processing to detect the object. The program deals with the input digital images or videos types, which contain the semantic objects. The interested objects have some specific features that can be distinguished from the others [10]. The features such as shape, color, and so on represent the individuality of the object class. For example, a circle must be round. The red circle must be circle and the color must be red. The specific features help to clarify the object. The complex objects contain many of features. For example, human face must have eyes, mouth, nose, and so on.

In this research, the road surface marking is a variation of an object detection algorithm. Then, the road surface marking was implemented based on the object detection algorithm where the objects have the features of lines.

3.2 Program Flow

The common processing flow of an image processing is pre-image processing, then image processing, and post-image processing at the end of each image. The workloads of pre-image and post image processing are much lighter than the highly intensive workload of the image processing part. The image processing part in the road surface marking consists of three major parts: changing the color space, applying the object extraction, and applying the line detection algorithm.

Fig. 3.1 shows the overall process of the road surface marking program. The program was implemented in the image sense module. The input of program is an image streaming via USB port. The output is sent to the virtual driving controller.

3.2.1 Color Space

The color space is an organization of color storage format. The combination of color in the color space creates the color that its represent. The general color space is Red, Green, and Blue (RGB), which is represented by red, green, and blue. The image is stored into data structure that contains the information of each color channel. A color model is an abstract mathematical model that is used in order to describe the representation of the color as numbers in data structure. In this research, two color spaces are used.

Firstly, the color space that is used in input image is Blue, Green, and Red (BGR). The input for the image sense module is the image streaming via USB port. The input color space is BGR, which is similar to RGB space but the order of color information is different. In this research, the color information is stored to 24-bit unsigned integers. Thus, the first 8 bits represents the information of blue color. Then, the second 8 bits represents green channel. The last 8 bits represents red channel. BGR space is calculated from the RGB color model. The combination of red, green, and blue color creates the new color. The RGB color model is an additive color model. The combination of each color channels are combined together. For example, the combination of pure red and pure green is yellow. This model is used in most of color video displays. In contrast, the subtractive color model is used in Cyan, Magenta, and Yellow (CMY). The combination of information color is an absorption of colors. For example, the combination of pure Cyan and Magenta produces blue. This model is used in natural colourants such as dyes.

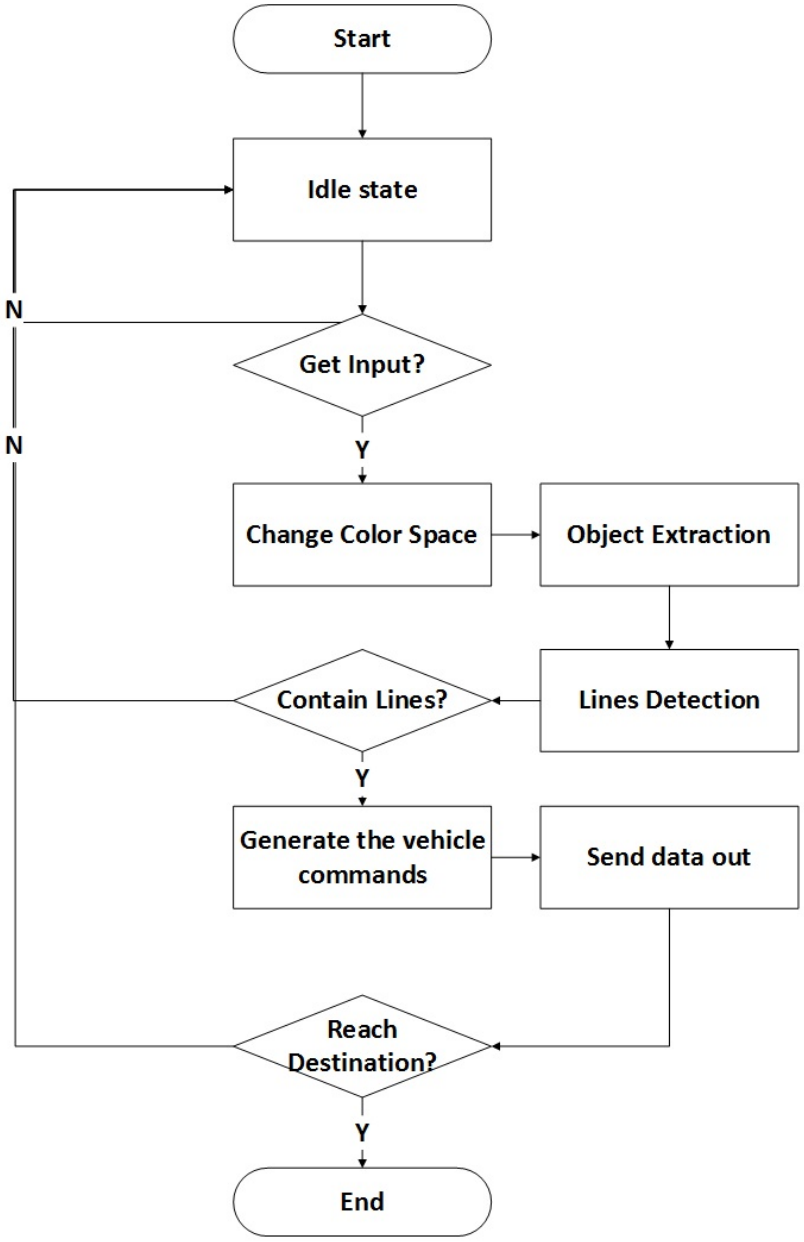

Figure 3.1: The program flow of road surface marking program

The combination of primary color in RGB space, which are red, green and blue, produces the primary color of CMY color space.

Secondly, the color space that is used for image processing in the image sense module is Hue, Saturation, and Value (HSV). HSV is one of the color space that is popular in the image processing. It rearranges the representation of RGB space to another system. In RGB space, the color space is represented by cartesian system. On the other hand, HSV space is represented by cylindrical system for being more intuitive. The first parameter hue is represented by the angle around the central vertical axis. The distance from the central vertical axis represents saturation. Lastly, the distance along the central vertical axis is value. Hue is an attribute of the color that describes the degree of similarity of target color and based colors in rainbow color. Saturation represents the colorfulness of the target color compare to its own brightness. Value is the comparison between brightness and brightness of illuminate white.

According to the Fig. 3.1, the first major step after obtaining the input image is "Change color space". The color space of input image is changed from BGR to HSV space.

$$V = max(R, G, B) \tag{3.1}$$

$$S = \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & otherwise \end{cases} \tag{3.2}$$

$$H = \begin{cases} \frac{60(G-B)}{V - min(R,G,B)} & \text{if } V = R \\ \frac{120(B-R)}{V - min(R,G,B)} & \text{if } V = G \\ \frac{240(R-G)}{V - min(R,G,B)} & \text{if } V = B \end{cases} \tag{3.3}$$

if $H < 0$ then $H = H + 360$.
The output data are in range $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$

The equation 3.1, 3.2, and 3.3, is used in the OpenCV library. The data is normalized. The data must be converted according to the destination data type. In this research, the images are stored in 24-bit unsigned integer. The available data range is 0 to 255. Thus, the data is converted to $V = 255V, S = 255S, H = H/2$.

The problem in BGR color space is the information of color and light intensity. The

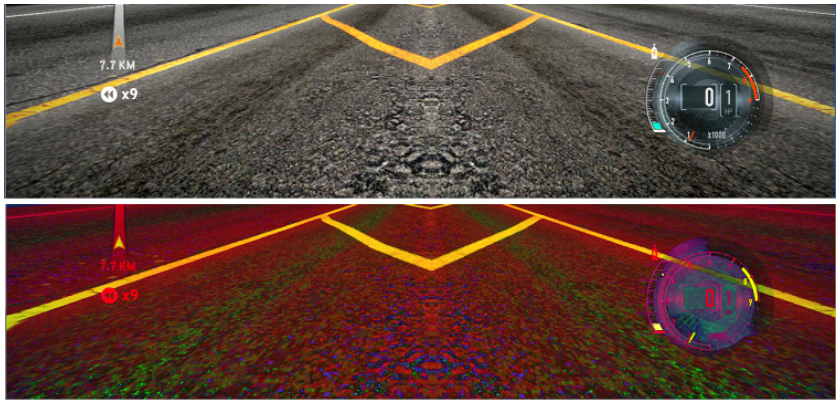

Figure 3.2: An example of color space converting from BGR to HSV.

combination of BGR color space blends the information of light and shadow to the color channel. Thus, the same color in different light source has different data in BGR color space, which affects the object detection based on the color. On the other hand, HSV color space separates the information of color and light intensity. The color has the equal hue and saturation in any light environment. The result from this step is a converted image in HSV color space from the input image which is in BGR color space.

The Fig. 3.2 shows the result of the color space converting. The input image from the Windows game simulator is retrieved in BGR color space. Then, after apply the color space converting, the input image in BGR color space is converted to the output image in HSV color space. The color representation is changed to its color space.

3.2.2 Object Extraction

The object extraction is a process for separating the interested objects from the background. The interested object must have unique features in order to extract from the other objects or background. The feature that we used in this research is color. From the color space, the input image is converted into HSV color space. The color based extraction is used to extract the objects that have the color in specific range.

$$\begin{aligned} dst_i = lowerB_{i,H} \leq src_{i,H} \leq upperB_{i,H} \quad \wedge \\ lowerB_{i,S} \leq src_{i,S} \leq upperB_{i,S} \quad \wedge \\ lowerB_{i,V} \leq src_{i,V} \leq upperB_{i,V} \end{aligned} \tag{3.4}$$

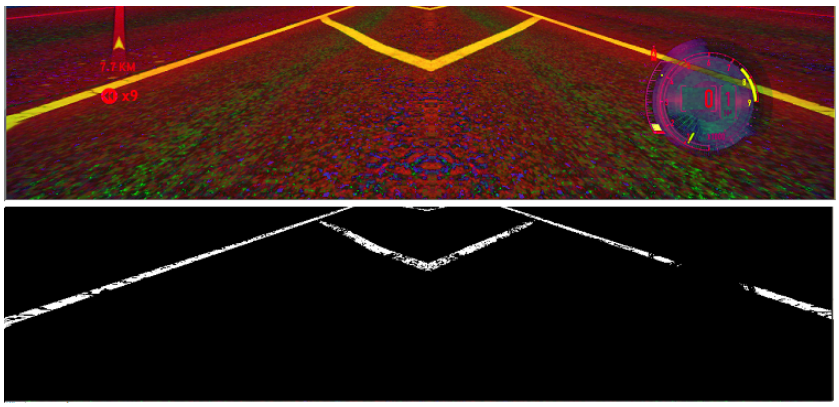

Figure 3.3: An example of an object extraction based on color feature.

From the equation 3.4, the destination image is a single channel image. The equaltion 3.4 is applied into each pixel in the image. All of three data channels, which are hue, saturation, and value, are blended together. In each channel, the upper bound and lower bound is specific to filter the pixel. The target pixel is operated by boolean operators. Thus, the destination pixel can be either 0 or 1, in other word, it is represented by black and white color.

The object extraction based on color requires an specific range of color information before processing, as shown in equation 3.4. In this research, the target color is yellow. In BGR color space, the combination of pure red and pure green produces pure yellow. However, BGR color space is not used for processing. HSV is used in image processing. Thus, from 3.1 and 3.2, the value and saturation of yellow color is 1, which are the maximum number. Then, from 3.3, hue is 30 out of 255. However, in this research, those color information for yellow color are not used in exact number. Concerning to the real environment, the color is hardly to be exactly number as it is calculated. Thus, the numbers are extended in this research to handle the real environment.

Fig. 3.3 shows the result of an object extraction algorithm. The interested objects from the HSV color space are extracted from the others. The result image is a binary color image, which represents the interested objects by white color and the others by black color.

3.2.3 Lines Detection

After the object extraction, the image is converted into black and white image. The white color represents the interested objects. The road surface markings is a target program in this research. Thus, the interested objects are markings on the road, which are road lines. The coordination of markings in the image is required to navigate the program. Moreover, the imperfections of an image data from the transferring and processing cause noise or missing data information in an image. Thus, the lines detection algorithm must tolerate the imperfection data images. For example, the single straight line may be disturbed and separated into two straight lines.

The Hough transform is one of the feature extraction that is used in image processing. The voting methods are used for extracting the interested object from the others. The object is transformed to parameter space. Then, the interested objects are extracted by the local maxima of accumulator space from the algorithm of the Hough transform. Generally, the Hough transform is used for identifying lines, circles, or eclipses. In this research, the Hough transform is used for identifying the straight lines.

Generally, the straight lines are defined as $y = ax + b$. The point (a, b) represents the straight line $y = ax + b$ in parameter space. The parameter a and b are used to define the angulation of the lines. The set of (x, y) represents the position or point in lines. Then, the lines can be represented by another form, which is $r = xcos\theta + ysin\theta$. In this form, r represents the distance between the closest point on the straight line and origin point. θ represents the angle between the x axis and the line from the origin point to the closest point on straight line.

From the equation $r = xcos\theta + ysin\theta$, the straight lines can be represented in r and θ plane, which is referred to as Hough space. In Hough space, the point (r, θ) represents the straight line equation. The (x, y) point in x and y plane contain a set of straight lines that pass through that point. Each straight lines that passes through that (x, y) point has different value of (r, θ). The set of (r, θ) can be mapped to r and θ plane. The sinusoid curve is formed in r and θ plan from the set of straight lines that correspond to point (x, y) in x and y plan. Then, the intersection points of sine waves in r and θ plan represent the straight lines that pass through points (x, y) which correspond to each sine wave. Thus, the straight lines in the processed image can be identified by applying the Hough transform algorithm. The number of sine waves that intersect in the same (r, θ)

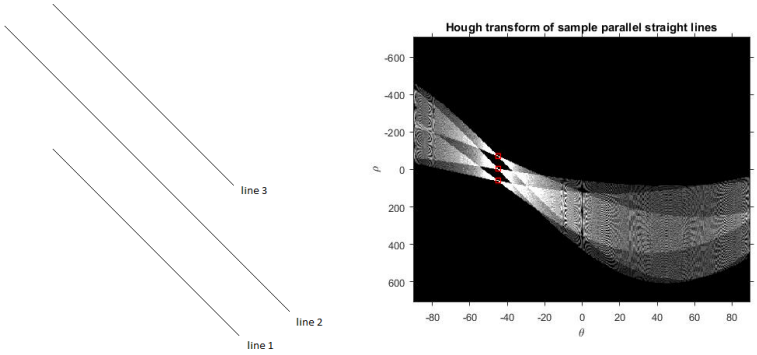

Figure 3.4: An example of applying the Hough transform algorithm to detect the straight lines.

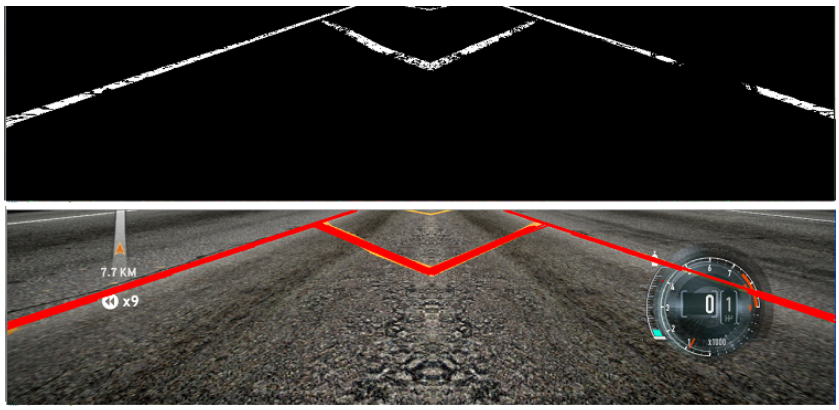

Figure 3.5: An example of applying the Hough transform algorithm to detect the lines.

point is used as a threshold for classified the straight lines.

Fig. 3.4 shows an example of Hough transform with the three straight lines. All of three straight lines are parallel to each other. Thus, the value of θ are equal in Hough space. The different distances from the origin of three lines are represented by three different ρ value. The top three highest intersection points of sine waves in Hough space represent the straight lines.

After applying the Hough transform to the processed image from object extraction method, the straight lines in the image are detected. The set of straight lines are obtained. Then, the navigation system can use these information to determine the direction of vehicle. The vehicle control commands are generated, which depends on the position of vehicle and detected lines.

Fig. 3.5 shows the result of applying the Hough transform algorithm to the extracted image. The detected lines are shown in red lines, which are reapplied to the input image in BGR color space. The Hough transform can identify the imperfect line in the image and reconstruct it.

Chapter 4

Hardware Accelerators

The computation resources such as central processing unit (CPU), memory, input-output ports, and so on, in the embedded systems are dramatically less than workstations and desktop PCs. Therefore, designing the high intensive computational programs in the embedded systems is different from that for workstations and desktop PCs. Moreover, in general, the embedded systems are used in an interaction environment. The responsiveness is one of the major concerns in the embedded system. In this research, the interested embedded system is the vehicle controller system. The navigation program, which is a road surface markings program, should respond to environment fast enough. Even though the road surface markings program is not highly compute-intensive program, it's response in embedded system is not fast enough to be used in the real-time environment. The CPUs processing in the embedded device is much less processing power than CPUs processing in the workstation. Thus, in order to achieve the real-time response, the system requires an accelerator. The software accelerators require no add-on components but they have the limitation of acceleration. On the other hand, the hardware accelerators provide much more acceleration factors than the software accelerator. However, the acceleration of the hardware accelerator depends on the hardware components that are used to accelerate. In this research, the graphics processing unit and programmable logic are used as the hardware accelerator.

The major difference between CPUs and hardware accelerators is the number of processing units. The hardware accelerators consist of a large number of low processing power units [5]. Those units process concurrently the same task with different data. In

the same task of computation, whether the hardware accelerator can be better or worse depends on its design. The important characteristic for using the hardware accelerator is the workload distribution. The hardware accelerator units are much more slower than CPU processing units. In contrast, the amount of hardware accelerator is massive. The workload distribution comes to affect the system performance. The amount of workload in each hardware accelerator unit should be nearly equality. The system is blocked by the slowest processing units. Because of this, hardware accelerators are not versatile for every algorithm. It can be a drawback with an inappropriate design. Moreover, the operation units are located outside the CPU. The workload of the target program must be moved from CPUs to hardware accelerators. The selected parts of the program, which are the compute-intense parts, are operated by hardware accelerators. Thus, the program flow of the system is changed. When the operation done in CPU parts reaches the compute-intense parts, the operation units of the program are changed from CPUs to hardware accelerators.The program flow is changed in this point. The CPUs and hardware accelerators can operate at the same time but the involved data and processes of CPUs and hardware accelerators must be completely separable. The hardware accelerator units operate with the separated memory storages. The involved data must be transferred from CPU memory to hardware accelerator memory. Thus, a run time overhead is incurred because of the data transferring.

As describe in Chapter 3, the road surface marking algorithm is implemented based on the object detection algorithm where the objects have the features of lines. The common processing flow of an image processing is pre-image processing, then image processing, and post-image processing at the end of each image. The workloads of pre-image and post image processing are much lighter than the highly intensive workload of the image processing part. The image processing part in the road surface marking algorithm consists of three major parts: cleaning the input image, applying the line detection algorithm, and analysing the detected lines. These tree majors parts are designed to be processed by the hardware accelerator units. The pre and post image processing are executed by CPU.

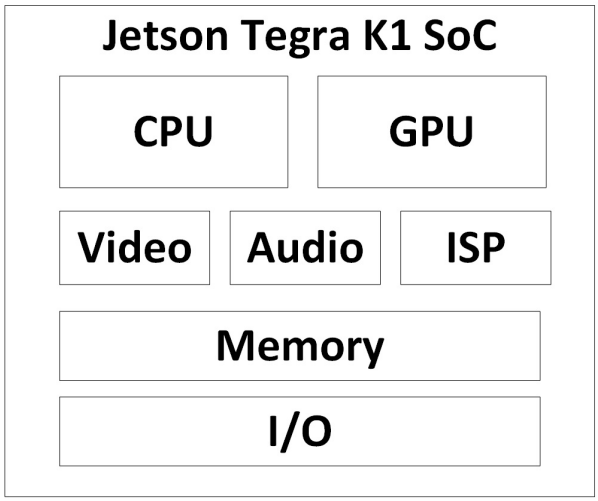

Figure 4.1: The hardware components of Jetson Tegra K1

4.1 Graphics Processing Unit

In computer graphics and computer vision, GPUs are widely used for accelerating program performance. GPUs are the hardware component that is designed for special purpose. It provides a massive number of programmable cores with a high bandwidth memory interface. The structure of GPUs is different from CPUs. GPUs are designed with highly parallel structure. Thus, the parallelism of GPUs is higher than CPUs. In image processing in which a massive number of data are processed with the same algorithm, GPUs are more efficient than CPUs which are designed for general purpose [11].

Due to the structure of GPUs, it is not appropriate to implement all of the road surface marking in GPUs. The suitable parts are the highly parallel processing parts. In general, the image processing algorithms suit for GPUs because of its parallelism. However, GPUs, which are hardware accelerators, have their own memory. It requires data transferring from CPUs to GPUs before processing the algorithms. Thus, the data transferring overhead may dominate the processing time. As a result, the processing in GPUs together with data transferring may take more time than the processing in CPUs where the involved data are already in the CPU-side storage.

The SoC for GPU programming that is used in this research is JetSon Tegra K1. The CPU is ARM Cortex-A15-based quad core CPU. It is designed for supporting every

Table 4.1: The frequency parameters for each mode in Jetson Tegra K1

Setting Mode	**#CPU cores**	**CPU clock freq. (GHz)**	**GPU clock freq. (MHz)**
Low power	1	0.51	72
Adaptive	4	2.32	72
High power	4	2.32	852

tasks. All of four CPU cores are performed in performance intensive application. The battery saver core is used in low performance application and background tasks in idle state. The CPU clock frequencies vary between 51 MHz to 2.3 GHz. GK20a GPU core is included in the board for supporting high performance. It supports at most 1024 threads per block with the maximum 32 warp size. The common usage I/O ports are integrated in the board, which is shown in Fig. 4.1.

In the road surface marking program, the major parts, which are compute-intensive parts, were implemented to operate in GPUs. The implementation of the image processing algorithm can be done by using open source computer vision (OpenCV). OpenCV is an open source library function, which focuses on the image processing algorithm. OpenCV provides both of CPUs and GPUs implementation. The compute-intense algorithms such as the object detection have GPUs implementation but some do not because the data transferring dominates the processing time. By using OpenCV, the data transferring occurs whenever the GPUs library function is called. The performance of the program is improved by the advantage of using GPUs. However, it can be much faster by reducing the number of data transferring to minimum.

Those algorithms which do not have OpenCV implementation for GPUs are implemented by compute unified device architecture (CUDA). CUDA implementation was applied if and only if the involved data are stored in GPUs memory. By applying CUDA implementation, the advantage of using GPUs is obtained while reducing the data transferring to the minimum number. In this research, the data transferring from CPU to GPU occurs only once and the data transferring from GPU to CPU occurs only once. The number of data transferring is reduced to the minimum number.

4.2 Overclocking

The overclocking is a function to make the hardware components run at a faster clock frequency than the default setting [12]. The performance of hardware components such as CPUs and GPUs can be increased or decreased by adjusting the clock frequency. The faster the clock frequency is the higher the computing performance is. Moreover, the default setting of most manufactured embedded systems is configured as a low power mode because of a power consumption factor. The difference between low power mode and high power mode has a great impact on the high performance system. The overclocking can be used in both of CPUs and GPUs. Thus, the system with GPUs parts obtains higher performance from increasing the clock frequencies of both CPUs and GPUs.

In this research, the clock frequencies of CPUs and GPUs are increased to the maximum to achieve the highest performance, which affect the respond time of system. Jetson Tegra K1 is designed for mobile applications. Thus, it is designed to control a wide range of clock frequency and a power usage of the system. The default settings are adaptive settings which suit for most use cases. The clock frequency of Jetson Tegra K1 performs high speed frequency when it operates intense programs. On the other hand, it performs with save power for light programs.

The parameters of overclocking in this research include the number of CPU cores, the CPU clock frequency, and the GPU clock frequency. The lowest number of CPU cores is 1 and the highest number of CPU cores is 4, as shown in Table 4.1. The range of CPU clock frequency is between 0.51 to 2.32 GHz. The range of GPU clock frequency is between 72 to 852 MHz. For the low power mode, all of setting parameters are set to the lowest number, which are a single active CPU core with 0.51 GHz clock frequency and 72 MHz for GPU clock frequency. On the other hand, the high power mode sets every setting parameter to the highest number, which are 4 active CPU cores with 2.32 GHz clock frequency and 852 MHz for GPU clock frequency. For the adaptive mode, all of setting parameters depend on run time environment. In the adaptive mode, usage of the processing resources changes while executing the lines tracking program. Thus, the result for the adaptive setting in Table 4.1 shows the maximum values during the execution. In the adaptive mode, the CPUs performance is the highest with four cores and 2.32 GHz, while the GPU capability is the lowest with 72 MHz.

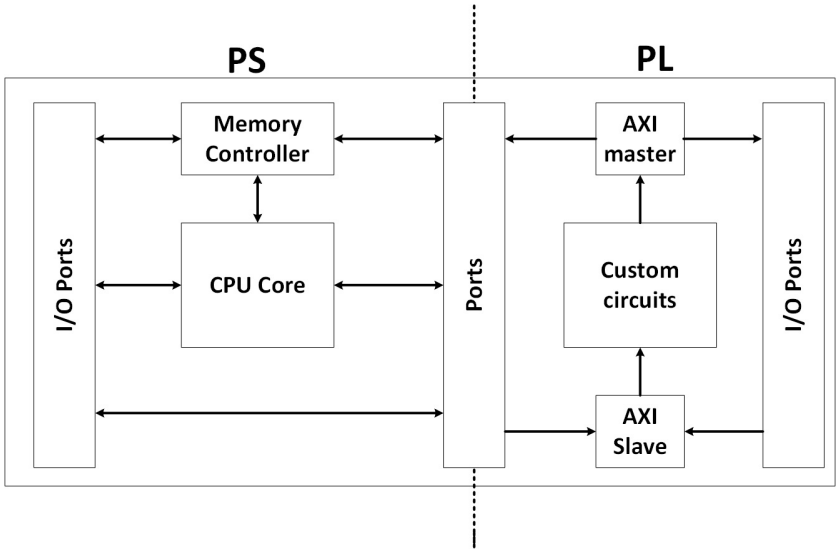

Figure 4.2: The example hardware architecture of programmable logic SoC

4.3 Programmable Logic

In modern embedded systems, SoC technology is commonly used because of the complexity of applications on the embedded systems. The application on the embedded system is not only ordinary computation but it may use an optical information such as streaming images from the camera, the special devices, and so on. Thus, the common components of a computer and some electronic systems are integrated together and provided by SoCs. Moreover, SoCs provide low power consumption. On the other word, the processing resources are limited to provided low power mode. Normally, SoC architectures increasingly feature hardware accelerators to achieve energy-efficient high performance [13].

Programmable SoC (PSoC) is one of the separated categories in SoC. The system of PSoC consists of 2 major systems, which are processing system (PS) and programmable logic (PL). The components in PS are permanently defined and unchangeable such as CPU. In contrast, PL side is not permanently defined but programmable. PL side can be programmed for any purpose which depends on the user design. In this research, PL is designed and programmed to be used as a hardware accelerator for cooperating with the PS.

The SoC for PL programming is not Jetson Tegra K1 because it does not contain

Table 4.2: Resource usage in Zedboard for hardware accelerator module

Name	Usage	Available	Utilization (%)
BRAM_18K	132	280	47
DSP48E	87	220	39
FF	26625	106400	25
LUT	39901	53200	75

the programmable logic part in system. Zedboard is used instead of Jetson Tegra K1. Zedboard is a SoC in the Zynq-7000 all programmable SoC (AP SoC) family. It integrates both of PS and PL. The hardware architecture of programmable logic SoC is shown in Fig. 4.2. The dual ARM Cortex-A9 MPCore is used as an CPU processor core. The CPU frequency operates up to 1 GHz, while the maximum frequency of PL is 667 MHz.

Programming PL is conventionally different from high-level programming language. In PL programming, a hardware description language (HDL) is used for describing the structure and behaviour of circuits. HDLs are interpreted into to a netlist, which is a specification of every physical electronic component and its connection. Then, the netlist is placed and routed on PL to create programmed circuits.

In this research, high-level synthesis (HLS) was used instead of programming with HDL directly. HLS interprets high-level language to HDLs automatically. The notion of time is not explicit in high-level language. However, notion of time is one of the most important in HDLs. Thus, the HLS tools must transform untimed functional code in high-level language into fully timed in HDLs.

Due to the programming model, PL has highly parallelism if it is programmed appropriately. The suitable parts are the highly parallel processing parts as same as GPUs programming. The data from the Windows game simulator are retrieved from PS part. At the end, the result data from the image sense module are sent by PS part. Thus, the data transferring occurs from PS to PL at the beginning and PL to PS at the end of acceleration. In SoCs, there are many peripherals in a system. The data transferring for interconnection has the standard protocol to standardize every transferring of peripheral. The advanced extensible interface (AXI), which is a common open-standard for SoC, is used in this research.

In case of PL, the resource usage of hardware accelerator is to be considered because it is programmable resources. The performance depends on the processing resource.

The more processing resource usage, the higher performance is improved. The resource usage in the hardware accelerator is shown in Table 4.2.

4.4 Amdahl's law

Amdahl's law is a formula for estimating the theoretical speedup of a program when some parts are improved. The overall speedup of the program depends on the ratio of improved parts. Thus, the theoretical speedup of the program is limited by the serial parts as follows [14].

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.1}$$

According to the equation 4.1, the overall speedup S is calculated from the proportion of parts benefiting from improvement, P, and the speedup factor, N.

In this research, the hardware accelerator improves the system performance from improving the parallel parts of the program. According to the equation 4.1, the improvement of using the hardware accelerator has a limitation from serial parts, which are not improved from the hardware accelerator. In GPU base accelerator, the overclocking increases the clock frequency of GPU. The CPU operation speed directly affects both of the serial part and parallel part. Thus, the overclocking improves the performance of the whole program.

Amdahl's law can be used for forecasting the improvement of the hardware accelerator, especially the upper bound limitation of an improvement. The high proportion parts are worth to improve even if the improvement is not high but the impact to the program is greater than the small proportion parts. Moreover, the data transferring is taken into account in order to use the hardware accelerator. From the beginning of the data transferring from the CPU to the hardware accelerator unit as the input data to the ending of the data transferring from the hardware accelerator unit to the CPU as the output data, the data transferring that occurs in the beginning and the end is operated in serial operation. The amount time of data transferring depends on the amount of data. In this research, the input image for the hardware accelerator and the output commands from the hardware accelerator are fixed. In other word, the execution time of data transferring is fixed. Then, if the improved part requires data transferring, the execution time of

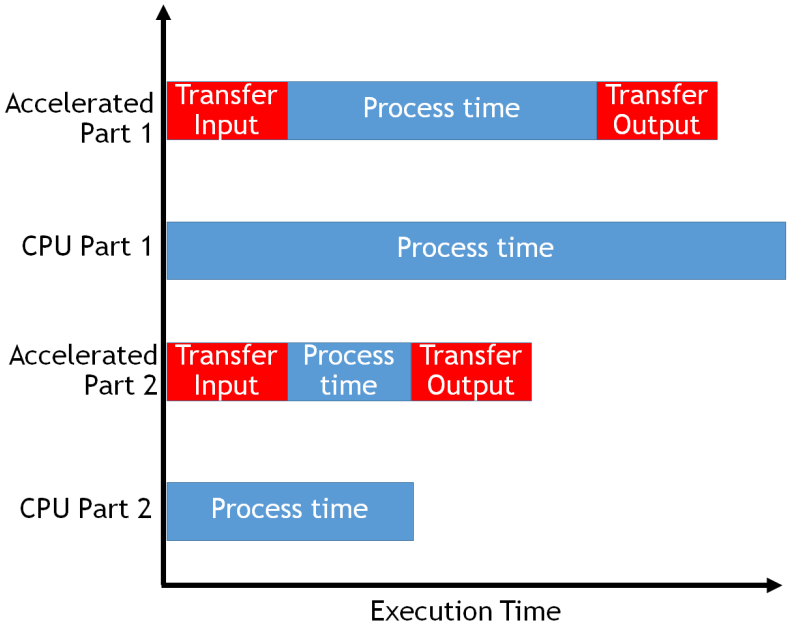

Figure 4.3: An example of the impact from the proportion of execution time

improved part must reach at least some curtain point to improve the whole program.

The target part for an acceleration must be chosen based on the proportion of that part according to the Amdahl's law to achieve the highest benefit for using the hardware accelerator. Moreover, in some part, applying the hardware accelerator may reduce the performance because of the data transferring. Fig. 4.3 shows an example of two parts of program. Part 1 has longer execution time than the part 2. Both of them are improved 2 times faster than the original version. However, in term of the impact from the improvement to the whole program, part 1 has a higher impact than part 2 because it has higher proportion. Moreover, the data transferring is taken into account because it is accelerated by the hardware accelerator. In part 1, the execution time of accelerated is better than the original. In contrast, the accelerated part 2 is slower than the original part 2 because of the data transferring.

Chapter 5

Experimentation and Analysis

The purpose of this research is to create the hardware accelerator for an image processing in the embedded system. Then, the concerned factor is the responsiveness or the performance that is improved from the hardware accelerator. In terms of responsiveness, the amount of time in each frame must be as fast as possible to achieve the better response time. Thus, the performance of this research is evaluated in terms of the execution time per frame. Generally, the real-time environment is achieved if and only if the processing can compute at least 30 frames per second (FPS), which is a common number for smoothness. In other words, the execution time for each frame is around 33.33 ms.

In this research, the hardware accelerators are implemented based on two types, which are GPU and PL. The evaluation of hardware accelerators are completely separated because of the hardware difference between Jetson Tegra K1 for GPU based and Zedboard for PL based implementation.

5.1 Experiment Setting

In the experiment, the image information is retrieved from the Windows game simulator. The image is cropped from the full 1280 width and 720 height to 1280 width and 300 height because the road surface marks are located in lower side of image. Then, the hardware accelerators are experimented by processing the cropped images. The performance is evaluated in terms of execution time per frame. The execution time per frame is retrieved from the average execution time.

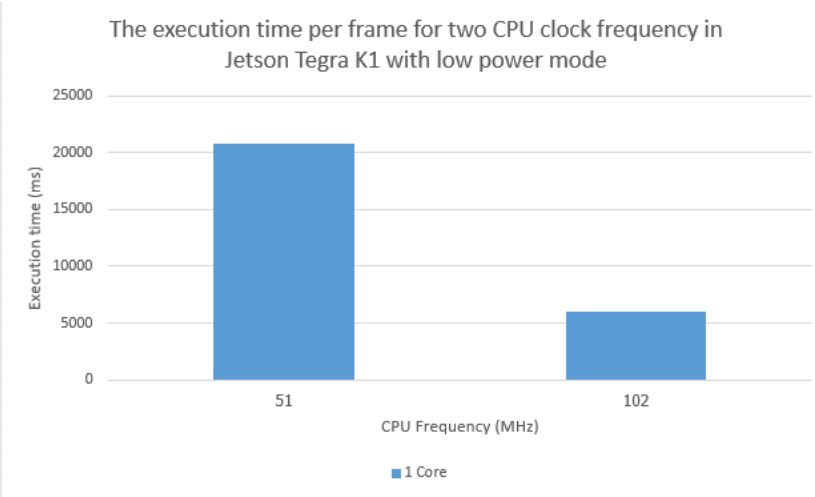

Figure 5.1: The execution time per frame for two CPU clock frequency in Jetson Tegra K1 with low power mode.

5.2 Experiment from GPU Based Accelerator

In case of GPU based accelerator, the performance of the GPU based accelerator and overclocking was compared with CPU-only processing in Jetson Tegra K1. The experimental results are shown in figure 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, and 5.7. The frequencies and number of CPU cores that are used in this experiment are supported by the hardware components and the manufacturer.

The performance of the standard based system, which is CPU-only processing, is evaluated with the various frequencies and number of usage cores. The x axis is the clock frequency in MHz. The y axis is the execution time per frame in ms. Each lines represents the number of CPU core that is used.

In case of Jetson Tegra K1, the lowest two frequencies are operated if and only if the Jetson Tegra K1 is operated in low power mode. This mode forces the Jetson Tegra K1 to operate with single core processor with the low frequency. The performance of low power mode is incomparable to the others. Thus, the performance of low power mode is separated from the other and shown in figure 5.1.

The other frequencies vary from 204 to 2320.5 MHz. From the figure 5.2, the trend of system performance is better when the frequency is increased. In terms of number of processing cores, the difference in performance is not clear in higher frequencies.

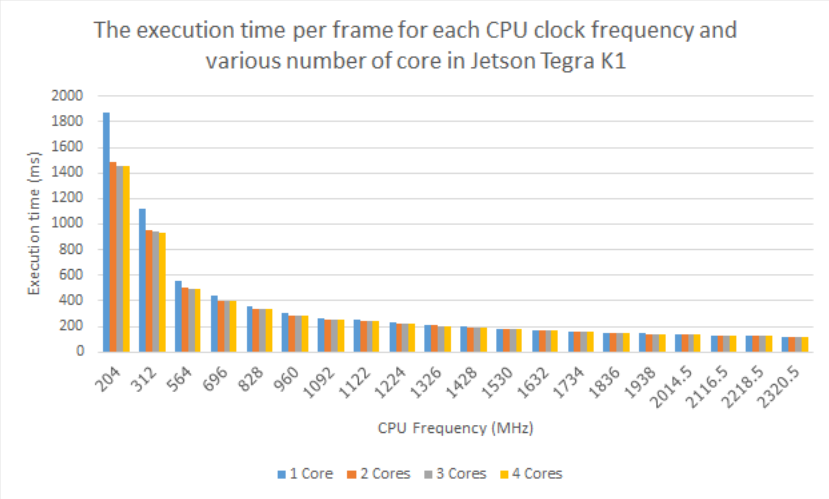

Figure 5.2: The execution time per frame for each CPU clock frequency and various number of core in Jetson Tegra K1.

However, at the lower frequencies, the performance of a single core processor is different from the 2, 3, and 4 cores. At 312 MHz, the execution time per frame is increased from the 564 MHz with an outstanding result. The workload from the road surface marking is not intense enough for the higher frequencies but when the frequency is decreased to below the edge, which is 564 MHz, the workload impacts the system with high computation time.

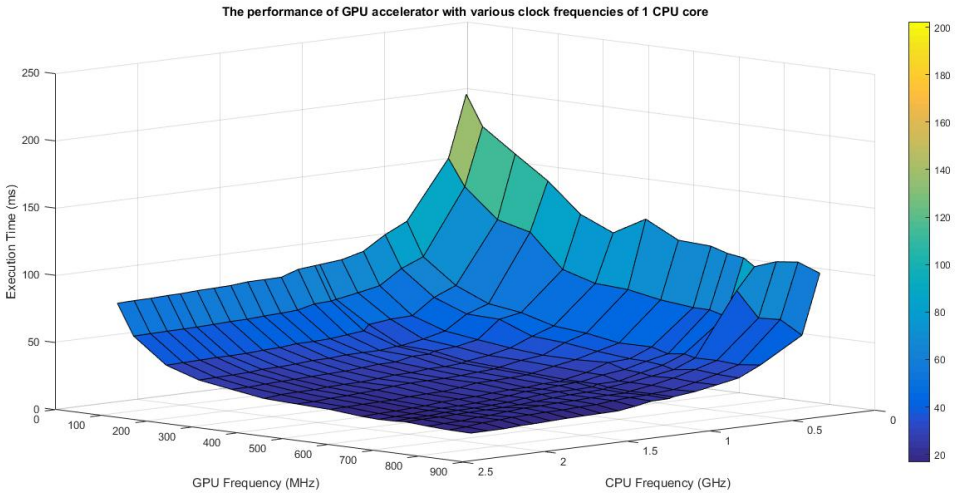

Figure 5.3: The performance of GPU accelerator with various clock frequencies of 1 CPU core

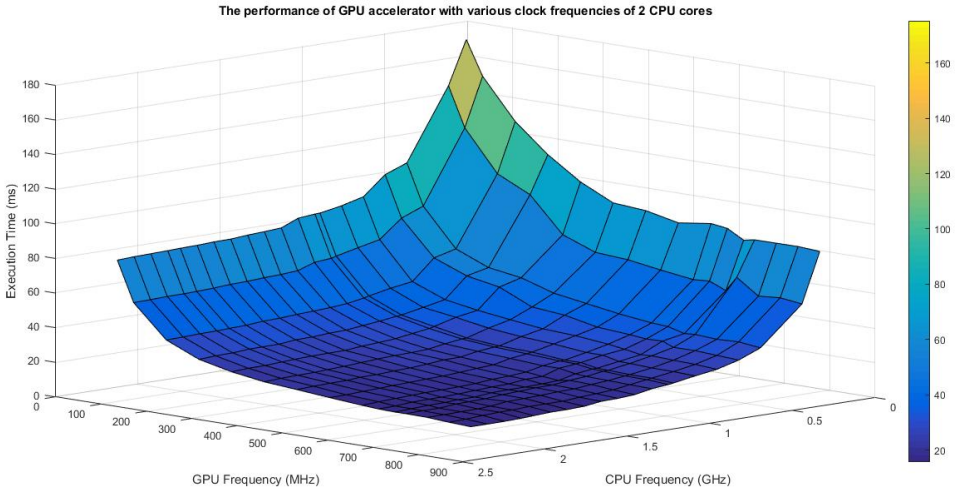

Figure 5.4: The performance of GPU accelerator with various clock frequencies of 2 CPU cores

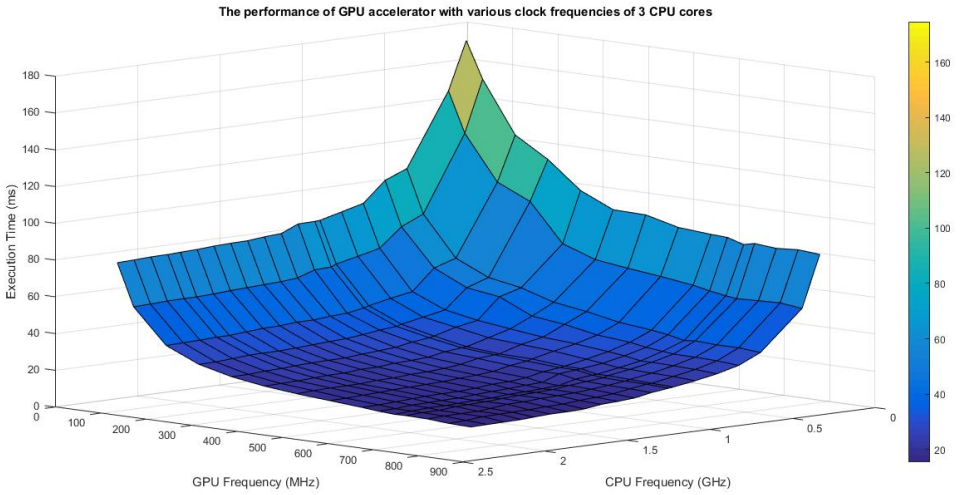

Figure 5.5: The performance of GPU accelerator with various clock frequencies of 3 CPU cores

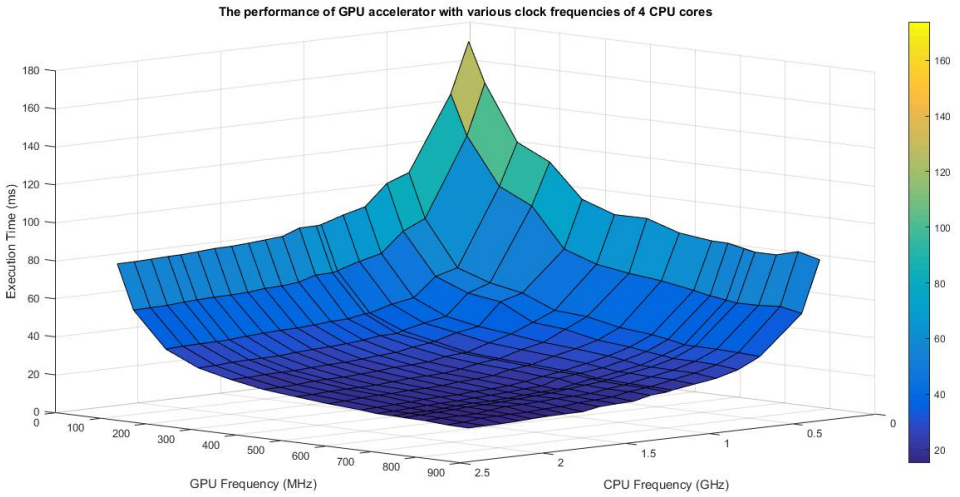

Figure 5.6: The performance of GPU accelerator with various clock frequencies of 4 CPU cores

In case of GPU performance, the evaluation is done with 3 parameters, which are GPU frequency, CPU frequency, and number of CPU cores. The combination from 3 parameters are evaluated in terms of execution time per frame. The results are shown in figure 5.3, 5.4, 5.5, and 5.6.

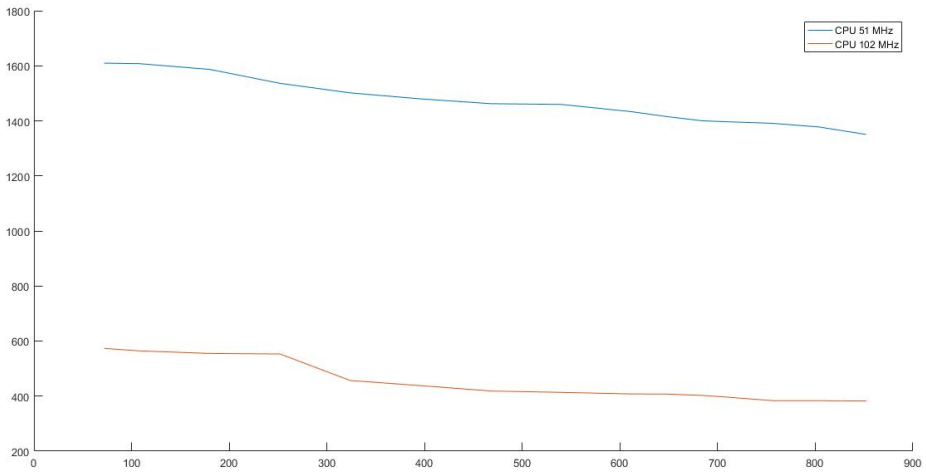

Figure 5.7: The performance of GPU accelerator with low power mode

The range of CPU frequencies is from 204 to 2320.5 MHz. The range of GPU frequencies is 72 to 852 MHz. The combination of two parameters are shown in 4 graphs, which combine with CPU core parameter from 1 to 4 cores. The performance gap of GPU accelerator between 2 to 4 CPU cores is not huge. However, the performance of single core CPU with GPU accelerator is slower than the others, especially when the frequencies of CPU or GPU are lower. The result from 4 graphs represents the same trend of performance. The performance of GPU accelerator is better when clock frequencies of GPU and CPU are increased. Thus, the highest clock frequencies of GPU and CPU, which are 852 MHz for GPU and 2.3205 GHz for CPU, with 4 cores have the best performance in this research. In contrast, the lowest clock frequencies of GPU and CPU, which are 72 MHz for GPU and 51 MHz for CPU, with a single core have the worst performance. The lowest improvement of GPU accelerator is operated in low power mode because CPU is operated at 51 MHz. The result from low power mode is shown in figure 5.7. As low power mode, an only single core CPU can be operated at 51 and 102 MHz.

According to figure 5.2 and 5.6, the performance of the standard based system, which is CPU-only processing, is far away from the real-time environment in all possible frequencies. In case of the best improvement, CPU is operated at the highest capability,which is 2.305 GHz and 4 cores. The execution time of this case is 117.519 ms. Then,

Table 5.1: The execution time per frame for each method in Zedboard

Method	Execution time (ms)	Speedup
CPU-only processing	303.556	-
Hw/Sw Co-design	16.332	18.59

the performance of the GPU based accelerator with overclocking is 15.197 ms that is 7.73 times faster than the standard based system.

5.3 Experiment from PL Based Accelerator

In case of PL based accelerator, the performance of the PL based accelerator was compared with CPU-only processing in Zedboard. The experimental results are shown in Table 5.1.

According to Table 5.1, the performance of the standard base system, which is CPU-only processing, is far away from the real-time environment. Then, the performance of the PL base accelerator is 18.59 times faster than the standard base system. The PL base accelerator achieves the real-time environment with the execution time of 16.332 ms.

Chapter 6

Conclusion

The embedded systems have been developed and used for decades. The embedded systems are used in many fields, especially an interaction system with the users. This kind of applications does not have any problem in the personal computer or laptop. However, when it is implemented into the embedded system, the problem occurs because of the low processing resource. The main problem from the lack of processing resource is the slow response time. On the other word, it lacks of the characteristic of the real-time system because it cannot be used in real-time environment. This kind of system is still in demand. Thus, it needs an accelerator methods to accelerate the system in order to reach the real-time response. In this research, the vehicle controller, which is the embedded system in a vehicle, is used as a model for prototyping. The target application of the vehicle control is the navigation system that is used in the self-driving car. The road surface marking program is chosen as the target program to be implemented in the vehicle controller. The virtual prototype imitates the real system by using hardware components. The input of vehicle controller is generated by the Windows game simulator and the HDMI capturing device. The vehicle controller is simulated by the image sense module with GPU or PL to accelerate the program execution. The control signal transmitter is simulated by the virtual driving controller generator. All of these components were integrated together to create the virtual prototype of this system.

The hardware accelerators are effective in achieving a higher performance. In this research, the system is accelerated by using two based type of the hardware accelerators. Firstly, GPU based accelerator is implemented in Jetson Tegra K1 with the overclocking

method. Lastly, PL based accelerator was implemented in Zedboard.

The result from the GPU based accelerator and overclocking shows that execution time per frame is 15.197 ms. The clock frequencies of CPU and GPU are used at the highest possible number with 4 CPU cores.The performance is around 7.73 times higher than the standard based system, which has the execution time is 117.519 ms.

The result from the PL base accelerator shows that execution time per frame is 16.332 ms, which is fast enough for the real-time requirement. The performance is around 18.59 times higher than that of the standard base system, the execution time of which is 303.556 ms. The resource usage of PL for hardware accelerator is controlled. The performance can be better if the resource usage is increased.

In conclusion, the virtual prototype of the vehicle controller system, which implements the road surface marking, can simulate real-time environment, in term of the responsiveness. The hardware accelerator is able to accelerate the road surface marking while remain the capable of tracking.

Bibliography

[1] Shahroz Tariq; Hyunsoo Choi; C. M. Wasiq; Heemin Park. Controlled parking for self-driving cars. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1861 – 1865, 2016.

[2] Seongrae Kim; Junhee Lee; Youngmin Kim. Speed-adaptive ratio-based lane detection algorithm for self-driving vehicles. In *2016 International SoC Design Conference (ISOCC)*, pages 269 – 270, 2016.

[3] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. Lost and found: detecting small road hazards for self-driving vehicles. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, 2016.

[4] Jeremy W. Sheaffer Kevin Skadron John Lach Shuai Che, Jie Li. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101 – 107, 2008.

[5] Andrew Putnam; Adrian M. Caulfield; Eric S. Chung; Derek Chiou; Kypros Constantinides; John Demme; Hadi Esmaeilzadeh; Jeremy Fowers; Gopi Prashanth Gopal; Jan Gray; Michael Haselman; Scott Hauck; Stephen Heil; Amir Hormati; Joo-Young Kim; Sitaram Lanka; James Larus; Eric Peterson; Simon Pope; Aaron Smith; Jason Thong; Phillip Yi Xiao; Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *IEEE Micro*, pages 10 – 22, 2015.

[6] Shuichi Asano; Tsutomu Maruyama; Yoshiki Yamaguchi. Performance comparison of fpga, gpu and cpu in image processing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009.

[7] Takaaki Miyajima; David Thomas ; Hideharu Amano. A domain specific language and toolchain for opencv runtime binary acceleration using gpu. In *Networking and Computing (ICNC), 2012 Third International Conference*, 2012.

[8] Jason M. Ready; Clark N. Taylor. Gpu acceleration of real-time feature based algorithms. In *Motion and Video Computing, 2007. WMVC '07. IEEE Workshop*, 2007.

[9] Santiago Snchez-Solano Javier Cerezuela-Mora, Elisa Calvo-Gallego. Hardware/software co-design of video processing applications on a reconfigurable platform. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 1694 – 1699, 2015.

[10] Nicholas J. Butko; Javier R. Movellan. Optimal scanning for faster object detection. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[11] J. Kruger; R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Visualization, 2003. VIS 2003. IEEE*, pages 287 – 292, 2003.

[12] Brian Greskamp; Josep Torrellas. Paceline: Improving single-thread performance in nanoscale cmps through core overclocking. In *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*, 2007.

[13] Christian Pilato; Paolo Mantovani; Giuseppe Di Guglielmo; Luca P. Carloni. System-level optimization of accelerator local memory for heterogeneous systems-on-chip. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[14] R. Kumar; D. M. Tullsen; N. P. Jouppi; P. Ranganathan. Heterogeneous chip multiprocessors. In *Computer*, 2005.