Lane Exit Identification and Alert (LEIA)

Real-time Lane Detection on an FPGA

Samantha Ball and Jason Pilbrough
Department of Electrical Engineering
University of Cape Town
South Africa

12 June 2020

***Abstract*—This paper aims to develop an embedded vision prototype to detect road lanes and markings in real-time. The computational complexity of lane detection is a significant barrier for real-time detection on traditional CPUs. A FPGA based architecture is presented to address this bottleneck by accelerating the image processing pipeline. Such a system has application in the field of autonomous vehicles and could form part of a Lane Keep Assist System (LKAS) or a larger Advanced Driver Assistance System (ADAS). The proposed design utilised the Hough Transform for straight line detection and was successfully implemented on the FPGA. A working hardware prototype was developed and achieved a processing time of less than 23.1 ms for each frame, sufficient for real-time application at 43.3 fps. This resulted in a speedup of 10.1 over an equivalent software implementation. The prototype was also able to correctly predict lane departures by analysing the slope of the detected lane markings in the image to identify when the vehicle was about to drift into a neighbouring lane. The robustness of the hardware design was demonstrated by testing the prototype at night in difficult conditions.**

I. Introduction

Autonomous vehicles are poised to radically disrupt the automotive industry. The present system of automotive transportation is prone to collisions and casualties, causing millions of people to lose their lives in avoidable accidents every year. Advances in computing and sensor technologies have paved the way for intelligent vehicles to solve this problem. Innovation is the field is currently begin driven by tech-giants like Tesla, Google, and Uber who are investing substantial resources into this technology.

This paper focuses on a particular form of driver assistance known as lane keep assistance and lane departure warning. In this contribution, we present a FPGA prototype that could be used to alert the driver when their vehicle is about to drift into a neighbouring lane. The idea is to use a forward facing camera to provide monocular vision of the road in front of the vehicle. The proposed system will use this vision to detect lane markings on the road using the Hough Transform for straight line detection. This type of driver assistance relies on rapid processing speeds and low latency to provide timely feedback to the driver. General CPUs may lack the required processing resources to meet this performance demand. On the other hand, FPGAs are well suited to the task of high-speed video/image processing and could meet the strict demands on performance. The inherent parallelism of the proposed image processing pipeline is also well suited to benefit from the hardware acceleration provided by an FPGA. In addition, the lower power consumption and smaller size make an FPGA suitable to be embedded in a larger Advanced Driver Assistance System.

A few different approaches have been proposed for lane detection in related work. McCall et al. [1] used steerable filters to detect road markings using the VioLET system. Risack et al. [2] proposed a lane state model to provide lane detection and to monitor estimation performance. Hajjouji et al. [3] used an adapted version of the Hough Transform to detect lane markings with less memory overhead.

This paper is organised as follows: Chapter 2 develops the theory behind image and video processing. Chapter 3 presents an overview of our approach to the problem including the proposed image processing pipeline and testing and validation strategies. Chapter 4 provides the detailed design of the system. Chapters 5 and 6 contain a proposed development and experimentation strategy. The performance of the prototype is detailed in Chapter 7 before conclusions are made in Chapter 8.

II. Background and Development of Theory

A. Digital Image Basics

An image can be modelled by a function of two variables $f(x, y)$, where x and y are spatial coordinates in the image plane as shown in Fig. 1 below:

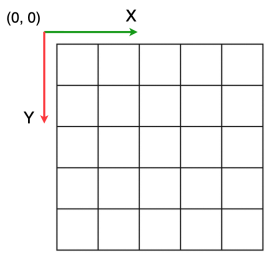

Fig. 1: Image coordinate system

The magnitude of $f(x, y)$ describes the intensity of the image at a given location in the image plane. A digital image is formed by quantising the x and y variables to form a finite number of units known as pixels. The intensity of the image is constant over the area of each pixel. The intensity is also a discrete quantity often stored as an 8-bit integer. Storing multiple intensity values for each pixel makes it possible to encode information about the color of the pixel. The most widely used color system is RGB ("Red Green Blue") which stores three values for each pixel to represent the "amount" of red, green or blue contained in the pixel [4]. The combination of these three values produces the overall color of the pixel. These separate values of intensity to describe each pixel are called color channels. When only a single channel is used, each pixel in the resulting image will be a different shade of gray. This is called a grayscale image.

B. Edge Detection

Edge detection is the process of emphasising regions in an image where the intensity of the pixels is changing rapidly. These regions correspond to edges in the image. A Sobel filter performs edge detection by computing the approximate gradient of the image intensity function $f(x, y)$ for each pixel in the image [5]. Pixels with a large gradient are likely to correspond to edges in the image. A Sobel filter is applied by convolving two separate kernels G_x and G_y with an image to emphasize vertical and horizontal edges respectively. These two Sobel kernels are defined in (1).

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & 2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

The output of each kernel at each point is then combined together to form a single value using equation (2).

$$G = \sqrt{(G_x)^2 + (G_y)^2} \tag{2}$$

The effect of applying a Sobel filter to an image is illustrated below. This is often called a contour image.

Fig. 2: Image before and after sobel filtering

C. Image Binarisation

A binary image is an image containing only two pixel intensities, typically black with a pixel intensity of 0, or white with a pixel intensity of 255. The image binarization algorithm thus introduces a threshold which determines whether a certain grayscale pixel is classified as black or white. If a pixel is greater than the threshold, it is set to 255, otherwise it is set to a value of 0. Binarization may take a simple or complex form depending on the application. In the simplest case, a fixed threshold such as the midpoint can be used. However, for more complex applications, an adaptive binarization algorithm can be implemented. In the context of lane detection, adaptive binarization would play an important role if the brightness of the image were to differ significantly such as in day and night conditions [5]. The binarization step is crucial in the processing pipeline as the input to the Hough Transform, which performs the line detection, must be a binary image. The effects of binarization can be observed in the image below. The most noticeable effects occur in the grey area of rock and trees on the left which becomes more starkly black and white after binarization.

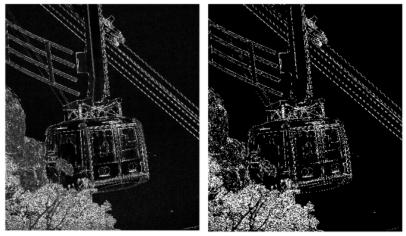

Fig. 3: Image before and after threshold binarization

D. Image Dilation

Image dilation has the effect of expanding the image pixels and is typically applied to binary images. Hence it follows the binarization stage in the pipeline. Image dilation is one of two fundamental morphological operations, with the other being erosion. Image dilation minimizes any holes in the boundaries found in the image and is thus an important pre-processing step before the Hough Transform. Image dilation effectively adds pixels to the boundaries in an image and thus makes these boundaries clearer before being input to the Hough Transform module [6]. Image dilation is performed by applying an NxN kernel to the original image which detects whether the centre pixel of the kernel is part of a boundary (i.e. if the centre pixel has a value of 255) and then sets the surrounding window of neighbouring pixels to 255 as well. In this way, the boundaries in the image are enhanced. The larger the size of the kernel, the more notable the effects of the dilation. An important step in designing the image dilation algorithm is determining the optimal method to handle boundary issues. Several common options include avoiding processing the boundaries, filling in the missing window entries from elsewhere in the image or shrinking the window near the boundaries so that each window is filled.

Fig. 4: Image before and after image dilation

E. The Hough Transform for Straight Line Detection

The Hough Transform (HT) is a robust method used to detect straight lines in an image with the presence of noise. The HT is appropriate for this application as most lanes will appear approximately straight in an image. It works by transforming points in the image plane (x, y) to Hough Space (HS) which is parameterized by (ρ, θ) The parameter ρ is the magnitude of the normal vector drawn from the origin to the straight line, and θ is the angle between this vector and the horizontal axis of the image [5]. This relationship is given by equation (3).

$$\rho = xcos(\theta) + ysin(\theta) \tag{3}$$

Given a point in the image plane, the set of all possible straight lines that pass through this point appear as a sinusoid in HS. A set of two or more points that lie on the same line correspond to the intersection of these sinusoids. The more points that exist on the same line in the image plane, the more sinusoids that will intersect at the same point in HS. The following figure shows how two lines in the image plane would map to HS:

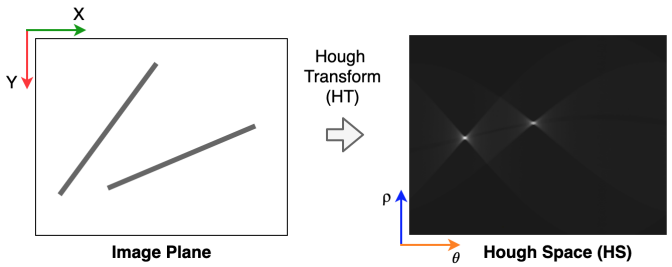

Fig. 5: Illustration of the mapping between the image plane and Hough Space (HS)

Fig. 5 illustrates how points on the same line in the image plane correspond to a point of intersection in HS. These intersection points are shown by the two "bright" spots in HS. The more pixels that are on the line, the more sinusoids that will intersect at the same point in HS, the "brighter" the point of intersection becomes in HS.

The Inverse Hough Transform (IHT) is the process of mapping points in HS, parameterised by (ρ, θ) back to straight lines in the image plane given as $y = mx+b$. The points in HS with the largest magnitude, correspond to lines in the image plane that are most apparent. The parameters of the line in the image plane are calculated using equation (4) and (5).

$$m = cot(\theta) \tag{4}$$

$$b = \rho/sin(\theta) \tag{5}$$

F. Video Streaming and VGA Output

Video Graphics Array (VGA) is a graphics standard which describes an analogue interface between a device and a monitor. A VGA connector is a three-row 15-pin DE-15 connector. VGA cables carry analogue RGBHV video signals. These signals include the RGB values for each pixel as well as the *hsync* and *vsync* signals which ensure synchronization. The horizontal sync (*hsync*) specifies the time taken to scan through a single row of pixels and vertical sync (*vsync*) signal gives the time taken to scan an entire screen of pixels [7]. Thus the horizontal sync demarcates a line while the vertical sync demarcates a frame. VGA provides a resolution of 640x480 pixels with a refresh rate of 60Hz [8]. The Nexys A7 board uses 4-bits per colour and uses resistor divider circuits to produce 16 signal levels for each colour signal [9]. VGA signals are partitioned into two phases: display time (drawing pixels) and blanking intervals [8]. To achieve a 60Hz refresh rate, the VGA circuit should be clocked at 25MHz where each pixel scan will take 40ns [7]. The timing of the VGA circuit is summarised in the diagram below which shows the relationship between the display time and blanking time.

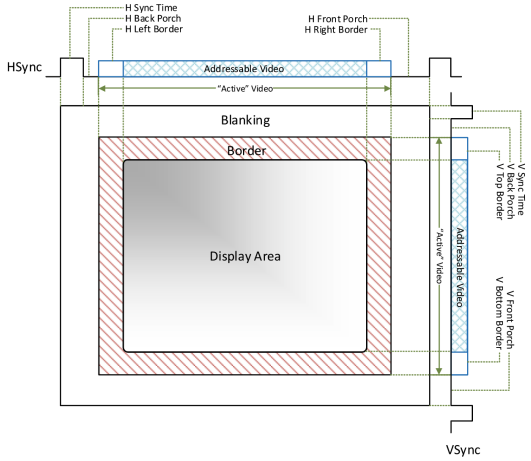

Fig. 6: Diagram showing synchronization and timing of VGA signals

In Fig. 6 it can be observed that the sync signals occur during the blanking intervals. Thus the display time makes up only a portion of the total time per line, with blanking taking up the residual time. This must be accurately mapped to the number of clock cycles on the FPGA in order to ensure correct functioning.

III. Methodology

This chapter describes the methodology adopted for this project.

A. Overview of Approach

The proposed system will be prototyped on a Nexys-A7 development board. The FPGA will be used to accelerate the image processing pipeline required for lane detection. The prototype will be capable of processing a single frame of video to extract useful information about the road markings. This will be sufficient to demonstrate a proof-of-concept and the system could be extended to process real time video in practice. A single 520x400 px frame will be taken from a frontward facing camera mounted on the vehicle and will be loaded into BRAM on the FPGA. The frame will be stored in greyscale to save memory.

B. "Plan A" and "Plan B" Approach

The project imposed significant time constraints and thus a dual-approach was taken to allow for adjustment of the scope of the problem during development. The pipelined nature of the project allowed for the scope to be reduced or extended with ease and thus the complete specification of the system, or "Plan A", could have been reduced if necessary. The 'Plan B' design constituted a revised version of the prototype to implement a simpler system with a subset of the original features.

Plan A: A complete prototype would implement the entire image processing pipeline required for lane detection. This pipeline would begin with first applying an edge detector to the input image. The result of which would undergo binarization and image dilation. This binary image would then be passed to the Hough Transform (HT) module for straight line detection. The HT module would then determine the position and orientation of the traffic lanes in the image and output this information to the line drawing module. This module would show the detected lanes on the original image before the final image is outputted to an external VGA display. The orientation of the detected lanes is also used to determine if the vehicle is about to drift into neighbouring lanes of traffic, triggering a warning LED.

Plan B: In the case of insufficient time to meet all the specifications of "Plan A", a simpler "Plan B" system would have been developed instead. The revised design focused on the fundamental stages in the image processing pipeline including the edge detection and threshold binarization. The resulting image would be output to an external display using VGA as per "Plan A". The pipelined nature of the project allowed for the scope to be reduced or extended with ease in accordance with time constraints.

C. Tools and Hardware

The hardware prototype was developed on a **Nexys A7** development board with **Xilinx Artix-7 100-T FPGA**. The FPGA has the following specs [9]:

- 15,850 logic slices, each with four 6-input LUTs and 8 flip-flops
- 4,860 Kbits of fast block RAM
- 240 DSP slices

A list of other required hardware is included below.

- Monitor with VGA input
- VGA Cable
- USB Cable (for programming the FPGA and debugging)

The prototype was implemented in Verilog HDL. Xilinx Vivado Design Suite was used for synthesis, translation, and route and place.

D. Basic Development Workflow

Each module in the image processing pipeline was developed independently. The first step involved prototyping each module in software until it produced acceptable results. Then an equivalent hardware design was drafted and simulated using the Vivado simulation environment. This process was iterative with modifications being made to each design as necessary. Once complete, each module was synthesised and programmed onto the FPGA for further testing. Finally, each module was integrated into the existing pipeline.

E. Expected Results

The proposed system is expected to detect lane markings from a frontward facing camera on a vehicle. The detected lanes should be drawn onto the original image and output to an external display. This behaviour is not strictly required for the system in practice, however it is important for development to validate the output. In addition, the proposed system is expected to correctly identify when a vehicle is about to exit its current lane from the forward facing image. This should trigger a visible lane departure warning. The system is also expected to perform both of these operations in real-time.

F. Validation and Testing Strategy

1) Golden Measure: The hardware prototype was benchmarked against a golden measure that is known to produce correct results. This golden measure was developed in software using a well tested library. The hardware design was validated to ensure that it produced the correct results before testing the processing speed and performance of the design.

2) Validation Strategy: The golden measure was used to validate the results produced by hardware design. Each stage in the image processing pipeline was validated separately using one of the three validation strategies outlined below:

Cross-correlation:
The output image of the hardware module is compared to the golden measure using a 2D cross-correlation function. The process of cross-correlation produces a value r between -1 and 1, where 1 is perfect correlation, 0 is no correlation, and -1 is perfect negative correlation. The goal would be to achieve a high correlation (value close to 1) indicating that the module produces an output that is very similar to the golden measure. A standard library implementation of a cross-correlation function was used for validation.

Pixel Error:
Another strategy to validate module output is to count the total number of pixels in the output image that have been misclassified. This is known as the pixel error and is typically used when evaluating a binary output image [10]. In order to provide a more intuitive understanding proportional to the size of the input image, the pixel error can be written as the pixel error rate (PERR) defined in equation (6).

$$PERR = \frac{pixel\ error}{total\ pixels} \tag{6}$$

It should be noted that the pixel error rate can be converted to a more traditional image quality descriptor, mean squared error, using equation (7) since the magnitude of the error will be 255 if a pixel is misclassified.

$$MSE = PERR.(255)^2 \tag{7}$$

The basis for determining whether a pixel is misclassified will be a pixel-wise comparison with the golden measure.

Mean-squared-error (MSE):
A final approach that is used to validate certain modules is the mean-squared-error between the actual output and the expected output from the golden measure. This is typically the approach that is adopted for modules in the processing pipeline that produce an output other than an image, such as a calculated value. In order to further enhance the evaluation, this process can be repeated for different signal to noise ratios (SNR) [11]. These varying SNRs can be generated by adding white Gaussian noise to the input test image.

3) Performance and Speed-up: The purpose of using an FPGA is to accelerate the image processing pipeline and achieve speed up over an equivalent software implementation. Speed-up of the hardware implementation over the golden measure will be determined by timing the critical sections of each implementation. The speed-up will be calculated using equation (8).

$$speed\ up = \frac{golden\ measure\ wallclock\ time}{optimised\ design\ wallclock\ time} \tag{8}$$

The critical section includes the following modules: Sobel filter, Image binarization, Image dilation and Hough Transform. To ensure caching will not be an influencing factor on the performance of the golden measure, the first three runs will be discarded. In addition, ten runs will be averaged together to increase the accuracy of each result. The hardware design should have deterministic performance and thus multiple runs for each test will not be necessary.

IV. Design

The design consists of an image processing pipeline that will be used to detect traffic lanes in an image. This pipeline contains several stages that are required for lane detection. The block diagram in Fig. 7 provides a high-level overview of the proposed system.

The prototype will accept an 520x400px greyscale image at the input, and produce an image at the output with the detected lanes. A warning LED will also be triggered should the system detect that the vehicle is about to drift into neighbouring lanes of traffic. Each of the modules from the block diagram in Fig. 7 will be described in more detail below.

A. Sobel Filter Module

The Sobel Filter emphasises regions in an image where the intensity of the pixels is changing rapidly. It does so by convolving two 3x3 kernels with the image separately, and then combining the result. The first kernel is used to detect vertical edges in the image, and the second kernel is used to detect horizontal edges in the image. The kernels are defined in Fig. 8 with the labels 'a' $->$ 'i' mapping to each index in the kernel.

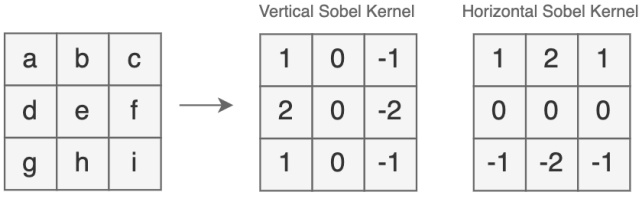

Fig. 8: Vertical and horizontal Sobel kernels

The process of convolution involves sliding the kernel across the image. The output at a specific point in the image relies on the values of the pixels surrounding the point. A buffer network ensures the correct pixel is provided to the correct Sobel kernel at the right time. This is done using a series of row buffers and pixel buffers. As the name suggests, the row buffer stores an entire row of the input of the image and the pixel buffer stores a single pixel. Fig. 9 below shows the proposed model of the module including the network of buffers that will be used to provide the necessary inputs, as well as the two Sobel kernels used to perform edge detection. Note that the labels 'a' $->$ 'i' correspond to the labels used in the diagram of the Sobel kernels shown previously.

B. Image Binarization Module

The image binarization module converts a grayscale image to a binary image. This module will be implemented using a fixed threshold. Initially, the threshold will be chosen to represent the approximate midpoint between the two output values. The output will be a binary image where every pixel has either a value of 0 or 255. The threshold value will be set to a value of 120 in the initial prototype, representing approximately halfway between the output values. Each pixel in the image will be compared to the threshold, with pixels greater than the threshold being assigned an output value of 255 and pixels less than the threshold being assigned an output value of 0. This procedure is illustrated in Fig. 10.

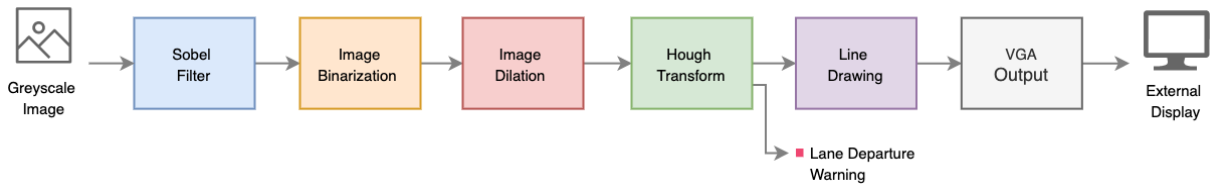

Fig. 7: Proposed Image Processing Pipeline

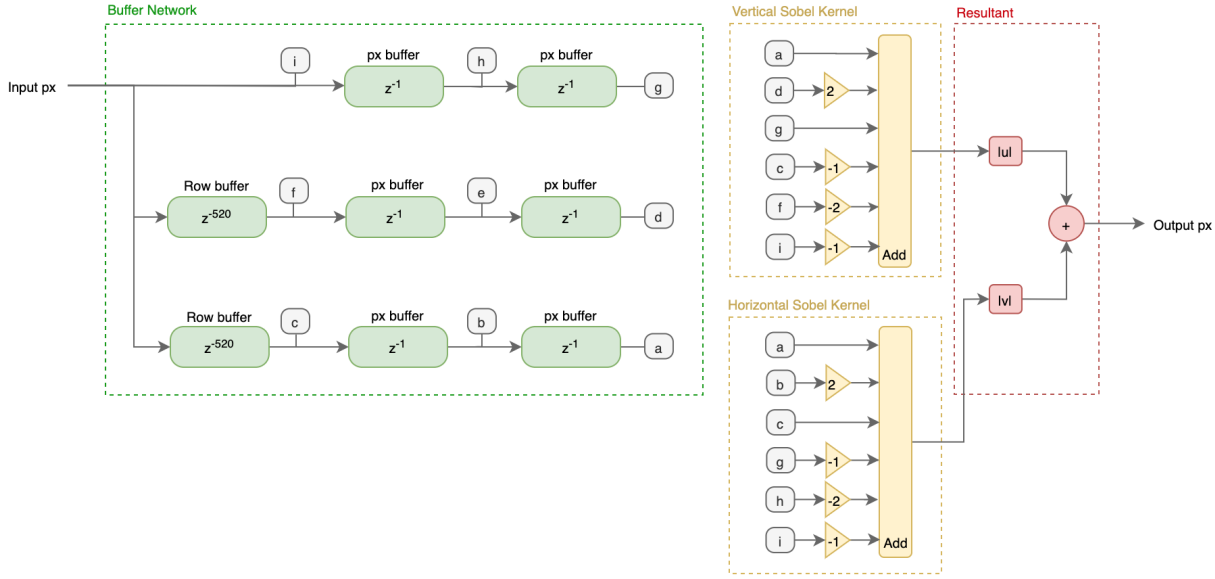

Fig. 9: Functional diagram of Sobel filter module

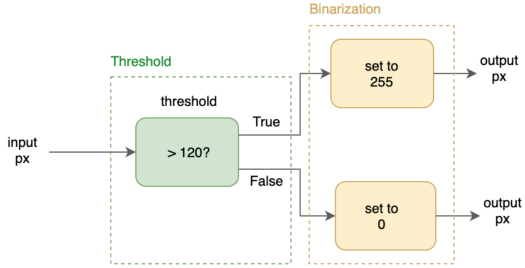

Fig. 10: Functional diagram of image binarization module

Once correctly implemented and time-permitting, an adaptive threshold may be implemented in order for the algorithm to perform well for both day and night conditions. One such method could be to use Otsu's threshold which would allow the binary image to be insensitive to background changes and illumination variance [2].

C. Image Dilation Module

The image dilation module smoothes the edges produced by the Sobel Filter and thus enhances the performance of the Hough Transform [2]. Image dilation can be performed using two equivalent processes. Both processes include moving an NxN window over the image. In the first process, if the centre pixel of the window has a value of 1, all neighbouring pixels in the window are set to 1 in the output image. The alternate process is more suited to hardware implementation. In the alternate process, the NxN window moves over the image and detects whether any of the pixels in the window have a value of 1. If one of the pixels has the value of 1, the centre of the window is set to 1. This has the same effect as the first method.

In order to realise the image dilation algorithm in hardware, a buffer network is needed to store each of the pixels in the 3x3 kernel. The OR operation is then performed in order to determine whether any of the pixels in the kernel have a value of 1. If at least one pixel in the window has a value of 1, the output pixel (centre pixel of the window) is set to 1. This procedure is summarised in Fig. 11 below.

D. Hough Transform Module

The Hough Transform (HT) is used to detect straight lines in the image, with the form $y = mx + b$, through a process of accumulation and voting in Hough Space. The input to the HT is a binary contour image and the outputs constitute the parameters of the detected line (m, b). To improve performance, the HT will only be computed over two specific regions-of-interest (ROI) instead of over the entire input image. The first region-of-interest, called ROI_L,

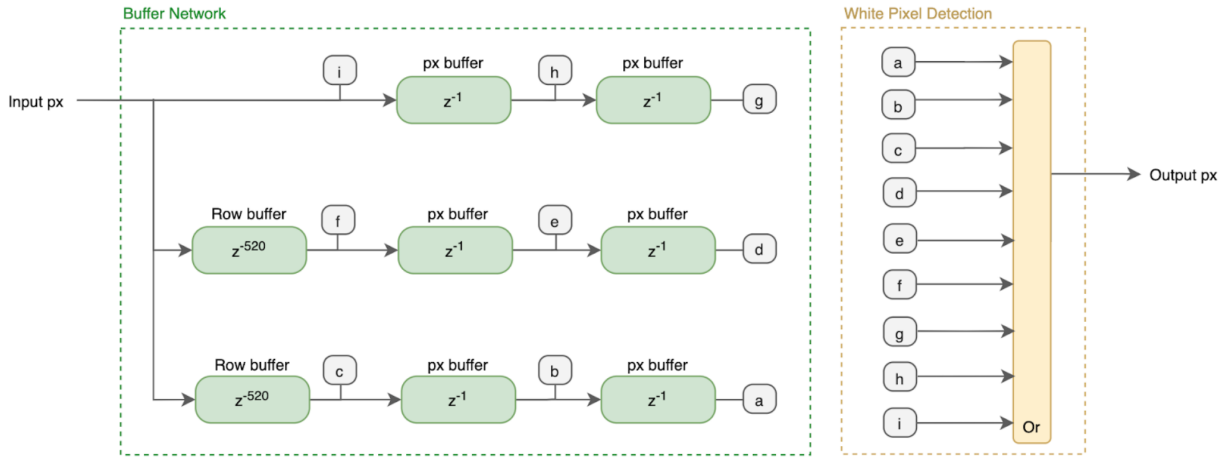

Fig. 11: Functional diagram of image dilation module

corresponds to the region in the image most likely to contain the left hand lane line, while ROI_R corresponds to the region most likely to contain the right lane line. Fig.12 illustrates the position of the two ROIs.

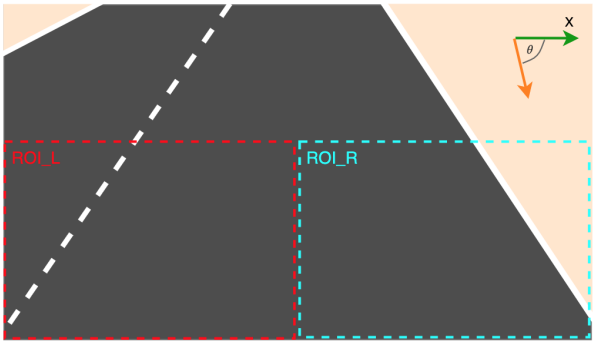

Fig. 12: Illustration of the two regions-of-interest (ROI) in the image.

To further increase performance, the domain of possible straight lines to be detected will be restricted to those that might match lanes on the surface of the road. This means restricting the slope of the line to be within θ_{min} and θ_{max} where θ is the angle made between the line and the x axis as shown. The best value for θ_{min} and θ_{max} will be determined experimentally for both ROI_L and ROI_R. Fig. 13 below shows the proposed model of the HT module.

The first block in this module is responsible for calculating the HT of a given point (x, y) in the image plane. This will be done using two LUTs, one for $cos(\theta)$ and one for $sin(\theta)$. The output from these two LUTs will be multiplied by the x and y coordinates of the point respectively, and then summed together. The point (x, y) will be held at the input while θ is varied from θ_{min} to θ_{max}. Adding the results from each multiplication gives ρ. Fixed-point arithmetic was used in all calculations in this module, with 8 bits reserved for the fractional part of the number.

The pair (ρ, θ) represents a point in Hough Space (HS). Each point in HS is mapped to a particular memory address in the accumulator. The accumulator block is responsible for taking the address provided by the HT calculation block and incrementing the number of votes stored at that address in memory. The process of voting for a particular point (ρ, θ) in HS essentially increases the confidence that a particular line defined by the parameters (ρ, θ) exists in the input image. The vote unit block, shown in Fig. 13 is used to increment the number of votes stored at a given location in the accumulator. The accumulator will be initialized as a dual-port RAM to allow the new number of votes to be written back into the same address in memory.

The peak detector block monitors the number of votes as they are incremented and stores the pair (ρ, θ) that corresponds to the highest number of votes. Once all points in the image plane has been processed, the pair (ρ, θ) stored in the peak detector represent the parameters of the most apparent line in the image plane. To determine the equation of this line in the image plane, the parameters (ρ, θ) must be transformed back using the Inverse Hough Transform (IHF). This process involves two more LUTs and a single multiply. The output of this module is the equation of the line in the image plane given by the equation $y = mx + b$.

E. Line Drawing and VGA Output

The VGA controller module requires a timing circuit to ensure the active display time and blanking intervals are properly managed. The functioning of the timing circuit is depicted in Fig. 14 below, showing the series of blocks needed. The first step involves dividing the 100MHz clock down to a 25MHz clock, also known as the pixel clock. The pixel clock then determines when the horizontal and vertical counters are incremented, with the vertical counter only being enabled when the horizontal counter reaches its maximum

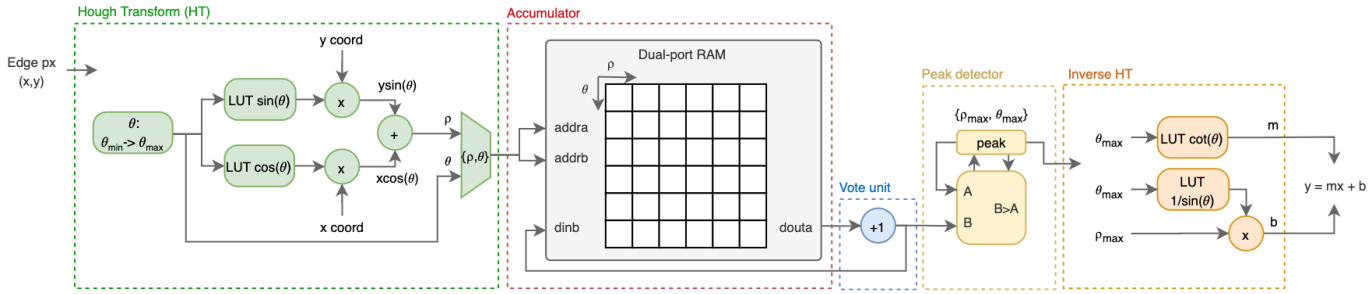

Fig. 13: Functional diagram of Hough Transform (HT) module

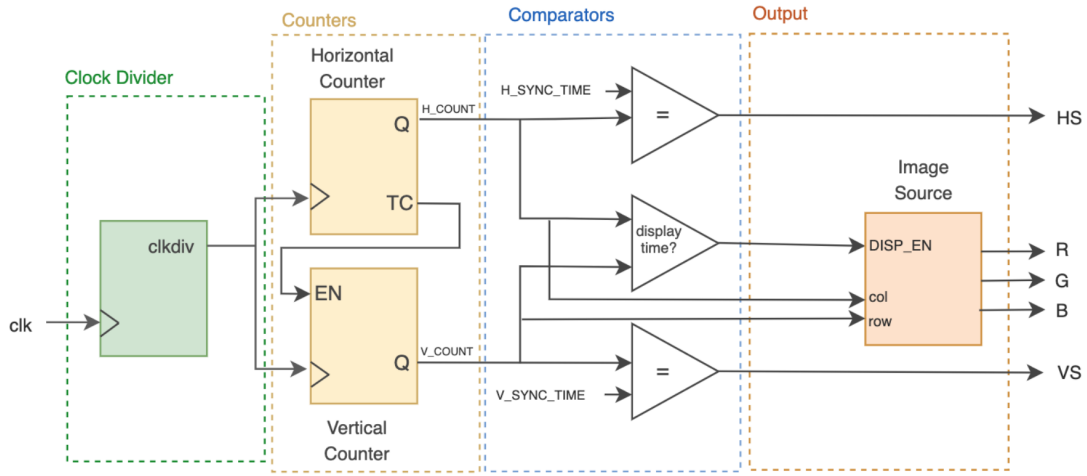

Fig. 14: Functional Diagram of VGA Controller Circuit

value (terminal count). This ensures that the vertical counter is only incremented once the end of a line has been reached. The counters are then used to compare the pixel counts to the given timing parameters needed to implement the display time and blank time. These timing parameters, such as H_SYNC_TIME and V_SYNC_TIME will be set as fixed local parameters in the Verilog module. The counters H_COUNT and V_COUNT keep track of the column and row information as well as dictating when the horizontal sync (HS) and vertical sync signals (VS) will be triggered.

The count values are also compared to the expected values signifying the start and end of display time. When the count values are between the defined display time interval, the DISP_EN of the image source is activated and pixel values are written from the image source to the R, G and B channels. When the count values are outside of the display time range, zeroes will be written to the RGB output, thus creating a blanking interval. The horizontal sync (HS), vertical sync (VS), R, G and B signals are then fed to the VGA connector.

V. Proposed Development Strategy

As mentioned previously, the proposed system has commercial application in driver assistance for autonomous vehicles. The prototype could operate as a stand-alone lane departure warning system or integrate into a larger Advanced Driver Assistance System (ADAS) to provide active lane keep assistance. Either option would help improve road safety by reducing the risk of driver inattention or fatigue. For the system to be used in this way, the design would need to be extended to process video input and to interface with a frontward facing camera on a vehicle.

A. Lane Departure Warning

A lane departure warning system immediately alerts the driver if the vehicle begins to drift out of its current lane. This is mostly used on highways and major roads. The system relies on the driver to react accordingly once the warning is given, and does not intervene and stop the vehicle from changing lanes. At present, the prototype is designed to turn on LED on the development board when it detects a lane departure. This would have limited use in a real driving environment and would have to be adapted to provide better feedback to the driver. On possible approach would be warn the driver using a pulsed vibration in the steering wheel. The prototype could be easily adapted to accommodate for this by redirecting the output trigger to a device in the steering wheel that regulates the vibration.

B. Active Lane-Keep Assist

The second use-case of the prototype is in active lane-keep assist. This type of system takes lane departure warning

one step further by automatically returning the vehicle to the correct lane when necessary. Depending on the level of driving automation, this action could be limited to a corrective maneuver if the vehicle is about to exit the current lane, or alternatively for a vehicle with higher autonomy this action could relieve the driver of the task of steering completely - this is known as auto-steer. The simplest approach would be to feed the output of the prototype into a control system to selectively brake individual wheels until the vehicle is centered back in the lane [12]. This idea is illustrated in Fig. 15 below:

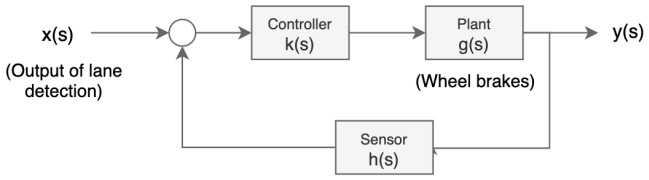

Fig. 15: Concept for active lane-keep assist control system

The binary output from the system would however likely lead to a form of ”bang-bang” control, which may be undesirable. For use in a more sophisticated control system, the prototype would have be re-engineered slightly to provide a slightly more complex output than the binary signal indicating lane departure.

VI. Planned Experimentation

This chapter aims to develop the experimental framework used to evaluate the performance of the system.

A. Experimental Setup

The experimental setup used to test the performance of the hardware accelerated system is described in this section. To begin with, the Verilog HDL was synthesised and converted into a bitstream on the host. The bitstream was used to program the Artix-7 100-T FPGA using JTag over a USB connection. Once programmed, the FPGA remains idle until a button is pressed on the development board which initiates processing. The output image that is produced by the processing pipeline is sent back to the host via the USB connection using UART. The host is configured to run a Python script that monitors a serial port. The Python script reassembles the transmitted image as it arrives at the serial port and writes the result to file. The terminal output produced by this script is shown below:

```
Listening on serial channel COM4 ... (timeout = 20 sec, baud = 8064000)
Recieved 624000 bytes of data
Writing image to file...
Done.
```

Fig. 16: Terminal output from host serial port monitor

Fig. 17 below shows a picture of the experimental setup with the FPGA connected to the host via USB.

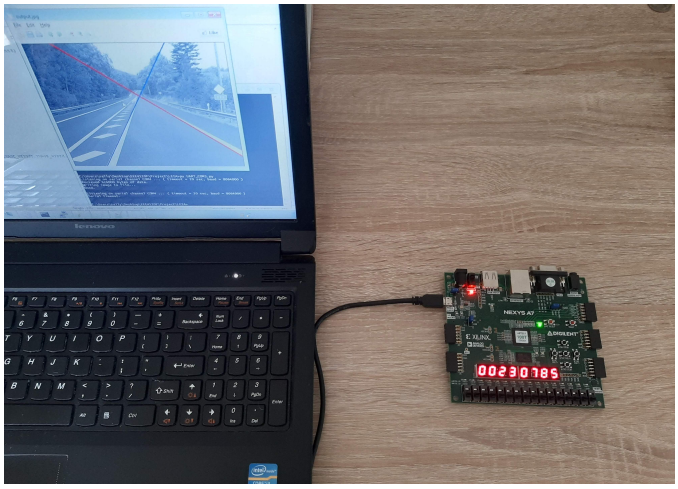

Fig. 17: Experimental setup

B. Development of Golden Measure

In order to develop the golden measure, each building block in the pipeline was implemented in Python using the OpenCV library. The OpenCV library is designed to solve real-time computer vision problems and is thus relevant to our application. It is highly optimised for numerical operations and is also very well tested. The respective library functions used for each stage include a sobel function for edge detection, a threshold function for adaptive thresholding, a dilate image function, and a HoughLines function for straight line detection. These four library functions form the basis of the algorithms that are to be implemented in Verilog for the digital accelerator. An overview of the golden measure implementation is shown in the listing below.

```
# SOBEL FILTERING
filters.sobel(img_grey_cropped,1,imx,cval=0.0)  # axis 1 is x
filters.sobel(img_grey_cropped,0,imy, cval=0.0) # axis 0 is y
img_sobel = np.uint8((abs(imx)+abs(imy))/4)

# THRESHOLD BINARIZATION
ret, img_thresh = cv2.threshold(img_sobel, 78, 255,
    cv2.THRESH_BINARY)

# IMAGE DILATION
# Taking a matrix of size DIALATION_KERNEL_SIZE as the kernel
kernel = np.ones((DIALATION_KERNEL_SIZE,DIALATION_KERNEL_SIZE),
    np.uint8)
img_dilation = cv2.dilate(img_thresh, kernel, iterations=1)

# HOUGH TRANSFORM
img_ROI_L = img_dilation[:, 0:int(width/2)]
img_ROI_R = img_dilation[:, int(width/2):]

lines_left = cv2.HoughLines(img_ROI_L,1,np.pi/180,110)
lines_right = cv2.HoughLines(img_ROI_R,1,np.pi/180,110)
```

Listing 1: Golden measure implementation in Python using OpenCV

It should be noted that the chosen threshold for the golden measure implementation of threshold binarization was set to 78 through experimentation.

C. Testing Procedure

Testing was conducted using the experimental setup described above. This basic test procedure involved first

loading a specific test image into memory on the FPGA, then processing the image, before finally sending the result back to the host for inspection. Tests were conducted using the three different input images shown in Fig. 18:

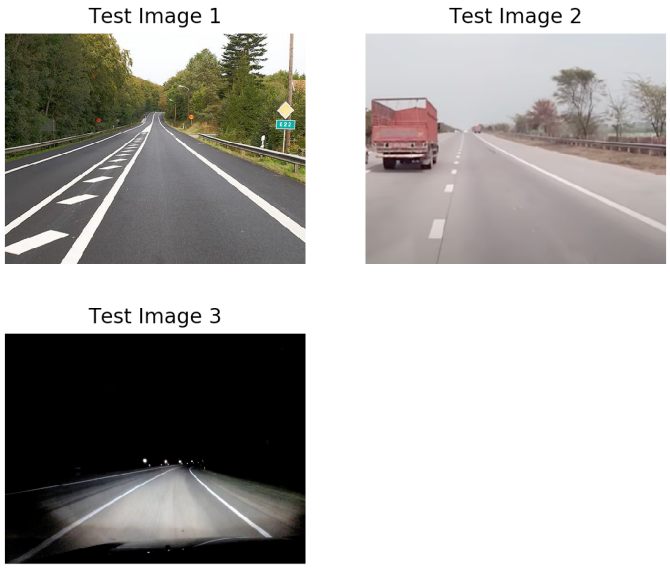

Fig. 18: Test input images 1,2, and 3

The three test images were chosen as follows:
Test Image 1: Tests lane detection in good conditions.
Test Image 2: Tests lane departure warning as the vehicle is about to change lanes
Test Image 3: Tests lane detection at night to ensure that the system still performs reasonably in more difficult conditions.

All test images are 520x400px stored in 8-bit grey scale. As described previously in the validation and testing strategy, the two key performance criteria of the proposed system are accuracy and speed-up. These criteria were benchmarked against the golden measure as described below. Also described below is a strategy for determining suitable bounds for the ROIs.

1) Determining suitable bounds for ROIs experimentally: As mentioned previously in Chapter 4, the domain of possible straight lines that will be detected by the Hough Transform (HT) module is restricted to improve performance. This optimisation can be made as the position of the camera mounted on the vehicle is fixed and known. Thus, any road lines are likely to appear in the regions-of-interest (ROI_L and ROI_R) with an orientation that is fairly uniform and predictable. The slope of the lane in the image can thus be restricted between θ_{min} and θ_{max}. A reasonable value for θ_{min} and θ_{max} was determined experimentally for each ROI using the golden measure implementation. A number of different test images were used to statistically determine the best value for each boundary. Furthermore, a suitable threshold for lane departure warning was also found experimentally. The lane departure warning should be triggered when one of the detected lanes has a slope that is highly upright, indicating that the vehicle is far from the center of the current lane and about to drift into a neighbouring lane.

2) Evaluation of Accuracy: A thorough evaluation of accuracy was conducted by comparing the outputs of the hardware accelerated system with the outputs of the golden measure. This comparison was made for each of the hardware modules to ensure that the output at all points in the processing pipeline was correct. Different validation strategies were adopted for different modules according to the type of output they produced. Table I shows which validation metric was used for each of the modules in the pipeline.

TABLE I
SELECTION OF VALIDATION METRIC FOR EACH MODULE

Module	Metric
Sobel filter	r (cross-correlation)
Image binarisation	PERR (pixel error rate)
Image dilation	r (cross-correlation)
Hough Transform	MSE

The details of each of the metrics shown in Table I were provided in Chapter 3. A Python script was developed to compute each of the validation metrics automatically using well-tested library algorithms.

3) Speed-up: As mentioned previously in Chapter 3, speed up was measured by timing the critical section of the hardware implementation. For these experiments, the critical section is defined as all modules in the image processing pipeline from the Sobel filter up to the Hough Transform. This does not include transmitting the image back to the host using UART. This was excluded from the speed-up calculation as this step would not typically be performed by the system in practice where the output of the system would be the lane departure warning alone. However, this step was required during development to debug the system and validate the output. Timing of the critical section was done using a custom timer on the FPGA. This timer was started as soon as processing was initiated and stopped just before the output image was sent via UART. The measured time was shown on a seven segment display.

VII. RESULTS

This section outlines the results that were obtained from the FPGA implementation in comparison to the golden measure.

A. Baseline results from the golden measure

The golden measure was developed in Python using OpenCV to implement each of the stages in the image processing pipeline. The output produced by the golden measure for test images 1,2,3 is shown in Fig. 19-21.

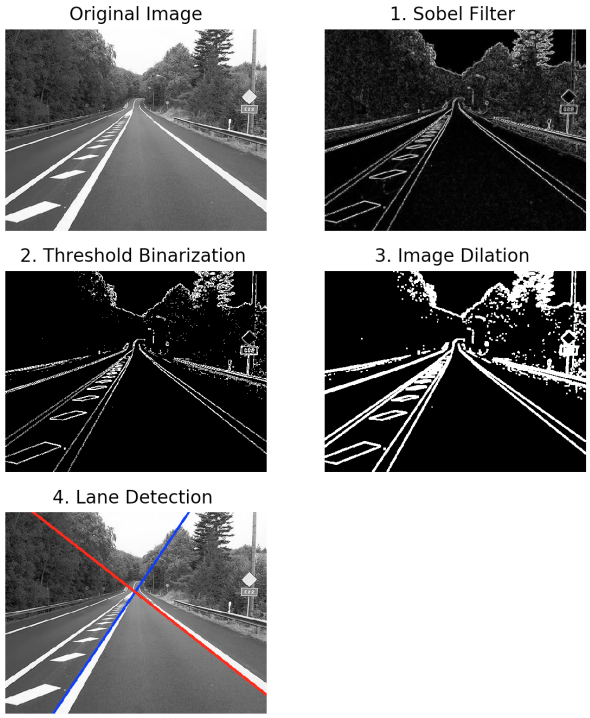

Fig. 19: Output of **Test Image 1** forming baseline results from golden measure.

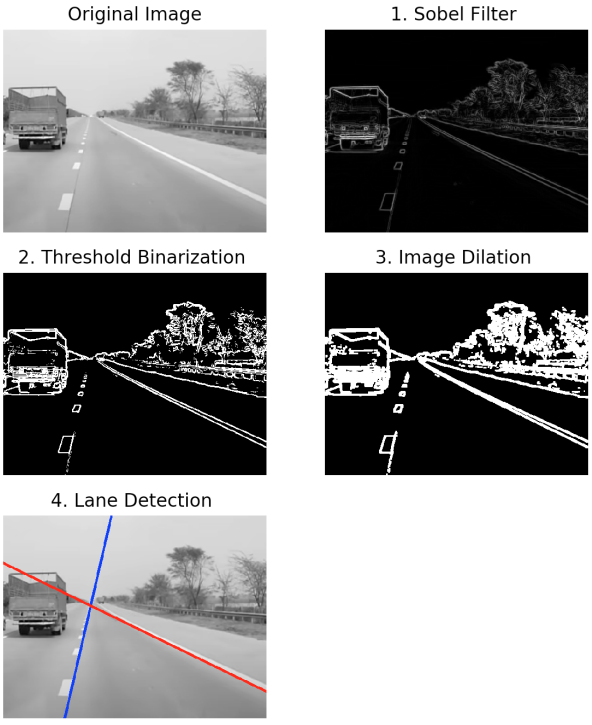

Fig. 20: Output of **Test Image 2** forming baseline results from golden measure.

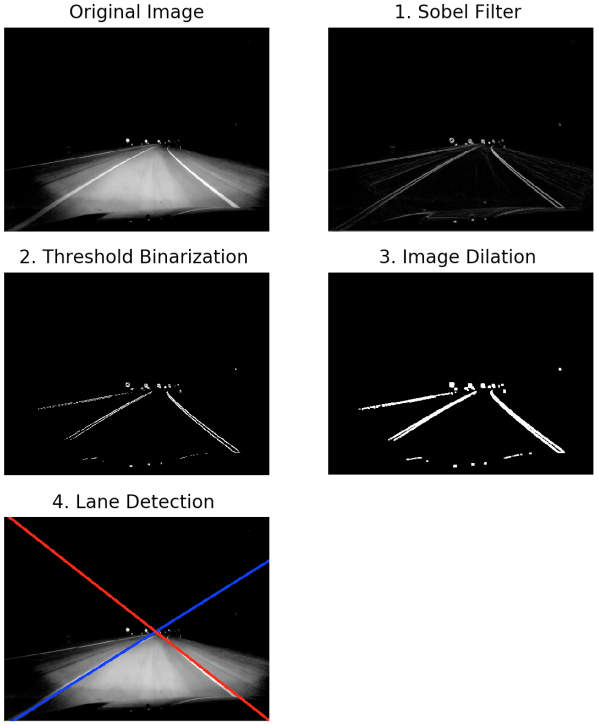

Fig. 21: Output of **Test Image 3** forming baseline results from golden measure.

The critical section of the golden measure was timed using the testing procedure described previously in Chapter 6. The average execution times for test images 1,2, and 3 were **232.2ms**, **233.9ms**, and **230.1ms** respectively[1]. The average execution time did not vary significantly over the different test images as each image was selected to have the same 520x400px dimensions. The golden measure would be able to achieve a frame rate of approximately **4.3 fps**, too low for any real-time application.

B. Determining suitable boundaries for the ROIs

The limits θ_{min} and θ_{max} on the slope of the detected lines were determined experimentally for each region-of-interest (ROI). For ROI_R it was determined that the slope of the detected line should be between $\theta_{min} = 1°$ and $\theta_{max} = 65°$, while for ROI_L the slope of line should be between $\theta_{min} = 115°$ and $\theta_{max} = 179°$. These ranges of values worked best for the input images that were tested. Furthermore, the best threshold for lane departure warning was also found experimentally. The lane departure warning should be triggered when one of the detected lanes has a slope that is very upright, indicating that the vehicle is about to drift into a neighbouring lane. A threshold of $21°$ was chosen for ROI_R and $159°$ for ROI_L. These thresholds proved to be a good compromise in detecting lane departures far enough in advance and preventing unnecessary warnings.

[1]Executed on a 2015 Macbook Pro 2.7GHz Intel Core i5

C. Results obtained from FPGA prototype

The FPGA implementation produced the outputs, shown in Fig. 22-24 below, using the experimental framework developed in Chapter 6.

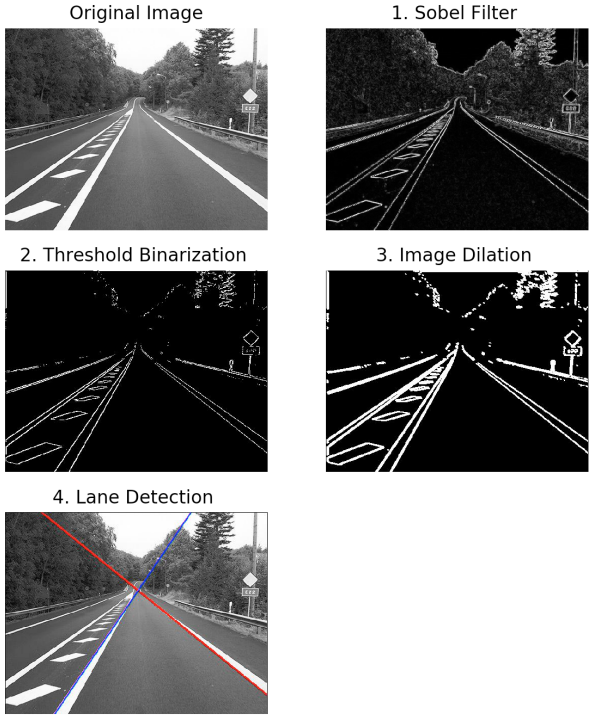

Fig. 22: Output of **Test Image 2** using FPGA prototype.

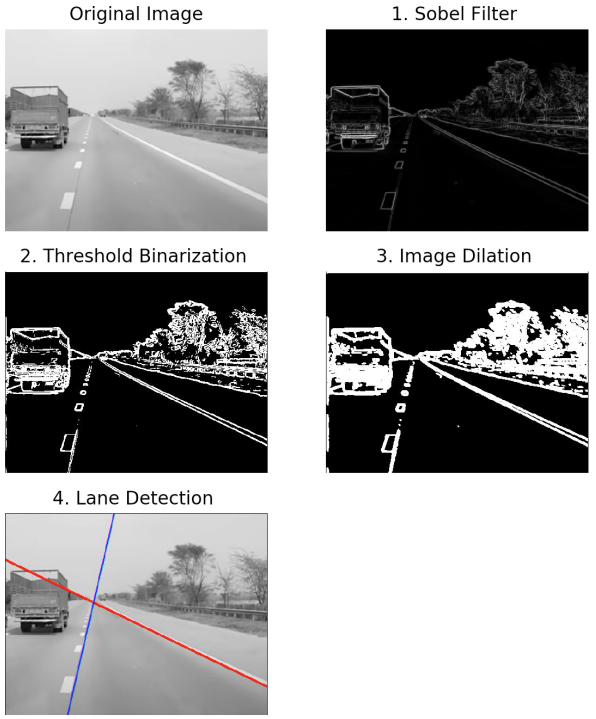

Fig. 23: Output of **Test Image 2** using FPGA prototype.

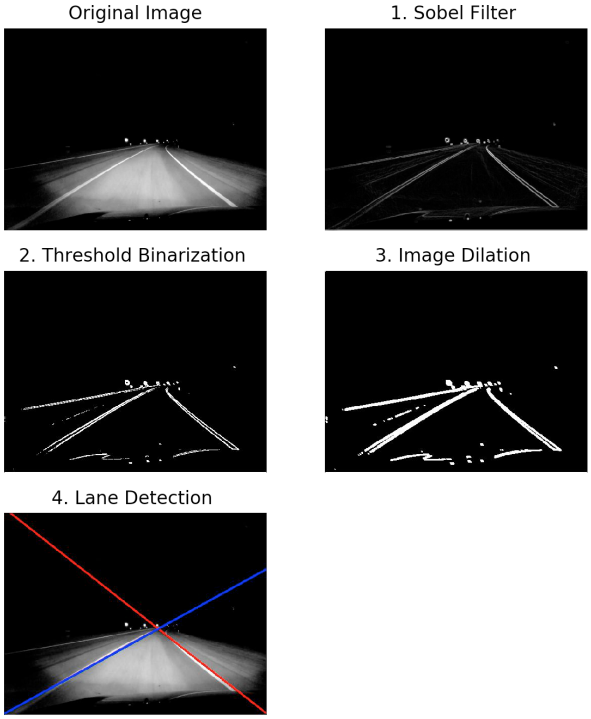

Fig. 24: Output of **Test Image 3** using FPGA prototype.

Through comparison of the observed results, the results obtained from the hardware system seem to match the results obtained from the golden measure, even in more difficult conditions such as night conditions in test image 3. The accuracy of the output will be determined quantitatively in the following section.

D. Validation of Output

The output from the prototype will be validated according to the validation strategy detailed in Chapter 6, using the golden measure as the ground truth. This involves validating each module in the pipeline using an appropriate metric. The validation score for each module is summarised in Table II below.

TABLE II

VALIDATION SCORE FOR EACH MODULE OUTPUT

Module	**Metric**	**Test Im. 1**	**Test Im. 2**	**Test Im. 3**
Sobel Filter	r	0.995	0.991	0.992
Image Binarisation	PERR	2.1%	1.4%	1.9%
Image Dilation	r	0.982	0.989	0.987
Hough Transform	MSE	0.17	0.13	0.14

The validation scores indicate that output from each module is reasonably accurate when compared to the golden measure. One possible source of the error in the output is the loss of precision in calculations due to fixed point arithmetic on the FPGA. In addition, both implementations had a slightly different strategy for dealing edge cases on the border of the

image which might have lead to slightly different results. This error is small and unlikely to impact on the performance of the system in practice.

E. Speed-up Calculation

Having validated the output produced by the FPGA implementation, the speed-up achieved from this hardware acceleration can be calculated. Table III below summarises the processing times for each test input for the golden measure and FPGA implementations. The speed-up achieved for each test image is also included.

TABLE III
AVERAGE PROCESSING TIME AND SPEED-UP

Test	Golden Measure	FPGA	Speed up
Image 1	232.2ms	23.1ms	10.1
Image 2	233.9ms	23.0ms	10.2
Image 3	231.1ms	22.8ms	10.1

As mentioned previously, the execution time does not vary significantly between test images due to the dimensions of each image being uniform. In addition, the processing time for each run on the FPGA is exactly the same as the FPGA is able to provide deterministic performance. The FPGA prototype was able to achieve a processing time of less than **23.1ms** for each frame. This is sufficient for real time processing with a frame-rate of **43.3 fps**. This is an important result as real-time performance is key for effective lane detection in practice.

F. Lane Departure Warning

The proposed system is used to provide lane departure warning. This warning is triggered if one of the detected lanes is excessively upright, indicating that the vehicle is about to switch lanes. For test image 1, the vehicle was centered in the lane and thus the warning should not be triggered. While the second test image the vehicle is in the process of changing lane which should trigger the warning. The warning is indicated by turning on a series of LEDs on the development board. This behaviour is demonstrated in Fig. 25 which shows the state of the LEDs after lane detection has occurred for test images 1 and 2.

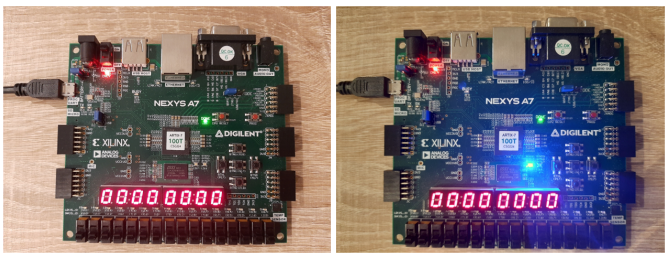

Fig. 25: LED Lane departure warning for Test Image 1 (left) and Test Image 2 (right)

G. UART and VGA Output on the FPGA

In the initial design, it was planned to output the image from the FPGA to an external monitor using VGA. After consideration, it was decided that this step was not essential for the proof-of-concept prototype and would not be implemented on the FPGA. Instead, the output was sent back to the host over UART. This approach was chosen for two reasons. Firstly, the project was undertaken during the COVID-19 pandemic which made it difficult to procure the additional hardware required for a VGA output - specifically the external monitor and VGA cable. Secondly, according to the validation procedure each intermediate output produced by the processing pipeline on the FPGA had to be sent back to the host anyway. A UART interface was developed for this purpose and was easily adapted to send the final output back to the host as well. Simplifying the design in this manner allowed for more time to spend on other aspects of the design.

H. FPGA Resource Utilisation

A summary of the FPGA resource utilisation is shown in Table IV below.

TABLE IV
FPGA RESOURCE UTILISATION

Resource	Available	Usage	%
Slice LUT	63400	1778	2.80%
Slice Registers (FF)	126800	784	0.62%
On-chip memory (BRAM tiles)	135	98	72.59%
Off-chip memory	-	0	0.00%
DSPs	240	5	2.08%
Power	-	0.178W	-

On-chip memory was a significant constraint on the design of the image processing pipeline. The **Xilinx Artix-7 100-T FPGA** only has 4.86Mbits of fast on chip memory distributed over 135 BRAM tiles. The image buffer and the Hough Space accumulator used the most amount of memory. The image buffer was used to store the 520x400px input image for lane detection. This image was stored in 8-bit gray scale consuming 1.66Mbits of memory (52 BRAM tiles). The Hough Space accumulator required 2^{17} unique addresses in memory to fully represent Hough Space (HS). Combining the two HS parameters ρ (11-bits) and θ (6-bits) results in 2^{17} different combinations which are represented by a unique location in memory. The number of votes for each pair (ρ, θ) is stored as an 8-bit number requiring 1.05Mbits of memory (32 BRAM tiles). An additional 14 BRAM tiles are required for other purposes including intermediate image buffers, row buffers, and trig value LUTs. This brings the total memory requirement to 98 BRAM tiles (72.59% of the available BRAM tiles). The memory requirement places an upper limited on the maximum input image resolution and would have to be addressed to process high resolution images.

VIII. Conclusion

A. Accomplishments

This paper presented an FPGA based architecture for lane detection and lane departure warning. The proposed design was able to correctly identify and extract lane markings from an image produced by a forward facing camera on a vehicle. This was achieved by successfully implementing the Hough Transform on the FPGA. Each 520x400px test image was processed in under 23.1ms which resulted in a 10.1x speed up over an equivalent software implementation. The design was capable of operating in real-time with a frame-rate of 43.3fps. Furthermore, the prototype was able to identify when the vehicle was about to drift into a neighbouring lane by analysing the slope of the detected lane markings in the image. The system was able to correctly predict lane departures and trigger the lane departure warning as necessary. The design was tested thoroughly in different driving conditions and detected lane markings correctly in each case. Some tests were completed at night demonstrating that the design was robust and produced the correct output in difficult conditions. The design was able to successfully meet all the objectives required for the proof-of-concept as set at the start of the project.

B. Challenges

One significant obstacle was the tight memory constraints imposed by developing the prototype on an FPGA. Image/video processing is always demanding on memory resources based on the inherent requirement to store the image/video being processed. It was a challenge to implement the image processing pipeline on the FPGA to be memory efficient while not compromising on processing speed. Further adding to the challenge was the inclusion of the Hough Transform in the processing pipeline which itself requires significant memory. This memory requirement grows with the size of the input image as Hough Space must also be expanded to fully represent the set of all possible line parameters. However, despite the unique challenge presented by the combination of all these demands on memory, the FPGA architecture was designed successfully to meet all the required specifications using only 72.59% of on-chip memory.

The ambitious scope of the project also proved to be a challenge with the very limited time available. The proposed architecture was carefully designed and consisted of many interconnected modules. Producing a working prototype within the required time frame required considerable effort. The complexity of implementing the Hough Transform on the FPGA made development particularly difficult, especially when navigating the memory constraints discussed previously. It was decided not to implement the VGA output on the FPGA as per the initial design, as this step was not essential for the proof-of-concept. Instead, the design relied on UART to send the resulting image back to the host for inspection.

C. Future Work

The project could be continued by refining the prototype from a proof-of-concept into a complete product. This product would have practical use in industry and could be used by autonomous vehicles as part of a Lane Keep Assist System (LKAS). The design would have to be extended to process video from the forward facing camera instead of just a single frame. The modifications required to allow for video processing should be fairly straightforward as the current design is already suitable for real-time processing. Another avenue for improvement would be to reduce the memory requirement of the image processing pipeline. Increasing memory efficiency would allow for higher resolution images and improve the fidelity of the output.

References

[1] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 20–37, 2006.

[2] R. Risack, N. Mohler, and W. Enkelmann, "A video-based lane keeping assistant," *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*, pp. 356–361, 2000.

[3] I. E. Hajjouji, S. Mars, Z. Asrih, and A. E. Mourabit, "A novel fpga implementation of hough transform for straight line detection," *Engineering Science and Technology*, 2019.

[4] University of Washington Computer Science Engineering, "Image processing," 2019. [Online]. Available: https://courses.cs.washington.edu/courses/cse557/00wi/lectures/imageprocessing.pdf

[5] J. Zhao, "Video/image processing on fpga," Master's thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, 2015. [Online]. Available: https://pdfs.semanticscholar.org/4920/c7fcefa6ca11bcd907d9ecef7f0181522b06.pdf

[6] S. Perkins, R. Fisher, A. Walker, and E. Wolfart, "Dilation," 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm

[7] Embedded Thoughts, "Driving a vga monitor using an fpga," July 2016. [Online]. Available: https://embeddedthoughts.com/2016/07/29/driving-a-vga-monitor-using-an-fpga/

[8] W. Green, "Fpga vga graphics in verilog part 1," December 2017. [Online]. Available: https://timetoexplore.net/blog/arty-fpga-vga-verilog-01

[9] Digilent, *Nexys A7 Reference Manual*, Pullman, Washington, July 2019. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual

[10] P. Stathis, E. Kavallieratou, and N. Papamarkos, "An evaluation technique for binarization algorithms," *Journal of Universal Computer Science*, vol. 14, no. 18, pp. 3011–3030, 2008.

[11] C. Hari, "Performance evaluation of different line detection algorithms," *International Journal of Modeling and Optimization*, vol. 2, no. 4, 2012. [Online]. Available: http://www.ijmo.org/papers/152-S014.pdf

[12] J. Lee, J. Choi, K. Yi, and M. Shin, "Lane-keeping assistance control algorithm using differential braking to prevent unintended lane departures," *Control Engineering Practice*, vol. 23, no. 1, pp. 1–13, 2014. [Online]. Available: https://doi.org/10.1016/j.conengprac.2013.10.008

Appendix

Github repository

https://github.com/jasonpilbrough/LEIA