Pertanika J. Sci. & Technol. 29 (2): 707 - 724 (2021)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

High Level Synthesis Optimizations of Road Lane Detection Development on Zynq-7000

Panadda Solod[1]*, Nattha Jindapetch[1], Kiattisak Sengchuai[1], Apidet Booranawong[1], Pakpoom Hoyingcharoen[1], Surachate Chumpol[2] and Masami Ikura[2]

[1]Department of Electrical Engineering, Faculty of Engineering, Prince of Songkla University, Hat Yai, Songkhla 90112, Thailand
[2]Toyota Tsusho Nexty Electronics (Thailand) co, Ltd Bangkok, Thailand

ABSTRACT

In this work, we proposed High-Level Synthesis (HLS) optimization processes to improve the speed and the resource usage of complex algorithms, especially nested-loop. The proposed HLS optimization processes are divided into four steps: array sizing is performed to decrease the resource usage on Programmable Logic (PL) part, loop analysis is performed to determine which loop must be loop unrolling or loop pipelining, array partitioning is performed to resolve the bottleneck of loop unrolling and loop pipelining, and HLS interface is performed to select the best block level and port level interface for array argument of RTL design. A case study road lane detection was analyzed and applied with suitable optimization techniques to implement on the Xilinx Zynq-7000 family (Zybo ZC7010-1) which was a low-cost FPGA. From the experimental results, our proposed method reaches 6.66 times faster than the primitive method at clock frequency 100 MHz or about 6 FPS. Although the proposed methods cannot reach the standard real-time (25 FPS), they can instruct HLS developers for speed increasing and resource decreasing on an FPGA.

ARTICLE INFO

Article history:
Received: 21 September 2020
Accepted: 30 December 2020
Published: 30 April 2021

DOI: https://doi.org/10.47836/pjst.29.2.01

E-mail addresses:
6010120066@psu.ac.th (Panadda Solod)
nattha.s@psu.ac.th (Nattha Jindapetch)
ak.kiatisak@hotmail.com (Kiattisak Sengchuai)
bapidet@eng.psu.ac.th (Apidet Booranawong)
hpakpoom@eng.psu.ac.th (Pakpoom Hoyingcharoen)
surachate@th.nexty-ele.com (Surachate Chumpol)
ikura@th.nexty-ele.com (Masami Ikura)
* Corresponding author

ISSN: 0128-7680
e-ISSN: 2231-8526

Keywords: Array partitioning, FPGA, high level synthesis (HLS), HLS interface, loop pipelining, loop unrolling

INTRODUCTION

Advanced Driving Assistant Systems (ADAS) are enhanced self-driving in autonomous driving cars. There are many sub-systems in ADAS, such as Lane Keeping Assistant System (LKAS) and

Lane Departure Warning System (LDWS) (Chen & Boukerche, 2020). Both LKAS and LDWS contain road lane detection, which is one of the complex systems in ADAS. The important thing about road lane detection, which causes the accuracy of detection is the various environment. Therefore, many algorithms are developed to resolve the problem. The perception model for autonomous vehicles in a variety of environments such as dynamic movements of obstacles, parked and moving vehicles, poor quality lines, shape curve, and the strange lane was reviewed (Feniche & Mazri, 2019). The typical model for lane detection consists of five steps: image cleaning, feature detection, model application, tracking integration, and coordinates translation. Most of lane detection algorithm contains straight line detection, which is Hough Transform (HT).

Due to the requirements of road lane detection, which are real-time computation and low-power consumption. A real-time lane detection was implemented using simple filter and Kalman filter on a high-performance device (IMX6Q) (Lee et al., 2017) and Canny-Hough is modified (Hwang & Lee, 2016) to achieve high-speed computation. To achieve both requirements to develop on a hardware device, Lu et al. (2013), Guan et al. (2017), and Marzotto et al. (2010) developed parallel architectures for the road lane detection on hardware devices (Virtex-5, ML505, ALTERA DE2, and ALTERA DE2-115). Although the high-speed computation with low-power consumption can be recompense by using hardware devices, the development time still long and high cost. In Promrit and Suntiamorntut (2017), blob detection and HT were applied for road lane detection on the low-cost hardware device (Zynq-7000 family). To decrease the development time, Khongprasongsiri et al. (2018) and Panda et al. (2018) considered the optimization technique on HLS (loop-pipelining and loop-unrolling) to resolve the bottleneck computation and found out the suitable factor for loop algorithm. However, the HLS optimization techniques on hardware devices analysis with complex algorithms such as lane detection have not been completely analyzed yet.

In this study, we proposed HLS optimization processes and analysis methods on hardware devices (Zynq-ZC7010) for implementing the complicated algorithms, especially the nested-loop for high-speed and low-resource usage on low-cost devices. The contributions are the proposed HLS optimization processes that should be done sequentially. In each process, we proposed an analysis method to consider the suitable optimization techniques should be applied to implement on a device. Road lane detection in Solod et al. (2018) was selected to be a case study, which contained four main steps: pre-processing, edge detection, line detection, and angle calculation. The experimental results illustrated the improvement after applied each HLS optimization process that can be the instructions for speed increasing and resource decreasing on an FPGA.

PRELIMINARIES

This section explains basic knowledge of lane detection (edge detection and Hough transform) and HLS optimization techniques (pipelining, loop unrolling, array partitioning,

HLS interface). Since this paper does not propose either the new architecture or algorithms for the lane detection, the original theories, as well as the HLS optimization techniques are explained. This is a case study to be applied with the proposed HLS optimization processes.

Edge Detection

There are many methods to perform edge detection. Prewitt, Sobel, Robert, and Canny edge detections use the gradient method to detect the edge by looking at the maximum and minimum value in the first derivative of the image. The gradient method of Prewitt, Sobel, and Robert to detect edge has different gradient values of x and y direction as respectively shown in Figures 1, 2 and 3.

Meanwhile, Canny edge detection also uses the gradient method, but it is more complicated steps than the others. In the edge detection process, there are six steps: (i) reducing the noise by using Gaussian filter, (ii) finding edge by taking the gradient of the image (the gradient value is the same as Sobel edge detection), (iii) finding edge strength, (iv) finding the edge direction by the Equation 1, (v) tracing the related edge direction to the direction in an image, then perform non-maximum suppression, and (vi) finally eliminating streaking by hysteresis.

-1	0	1
-1	0	1
-1	0	1

G_x

-1	-1	-1
0	0	0
1	1	1

G_y

Figure 1. X and Y direction gradient value of Prewitt Edge Detection

-1	0	1
-2	0	2
-1	0	1

G_x

1	2	1
0	0	0
-1	-2	-1

G_y

Figure 2. X and Y direction gradient value of Sobel Edge Detection

1	0
0	-1

G_x

0	1
-1	0

G_y

Figure 3. X and Y direction gradient value of Robert Edge Detection

$$theta = tan^{-1}(G_x/G_y) \quad [1]$$

Hough Transform

For lines and circles identification, Hough transform is one of a feature extraction to process by using a voting procedure to detect incomplete instances of objects of a particular type of shape. Generally, according to Equation 2, the straight line is capable to represent the linear equation with two important parameters: slope (m) and intercept

(c). Equation 3 represents the polar form of a line. Figure 4 shows the transformation of Hough, where ρ represents the vertical distance of the straight line on x-y plane from the origin position. The angle between ρ and horizontal axes is represented by θ. A position on the blue straight line on x-y plane indicates to each curve on θ-ρ plane. An identical straight line will cause an identical intersection position of the curve on θ-ρ plane. Therefore, the length of the straight line is able to represent the number of intersection curves.

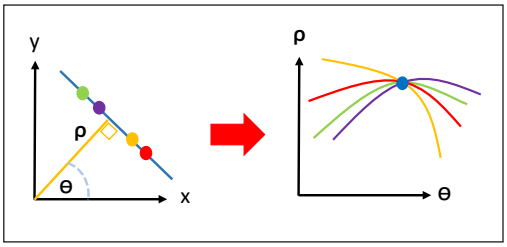

Figure 4. Hough transform concept

$$y = mx + c \quad [2]$$

$$\rho = x\,cos\theta + y\,sin\theta \quad [3]$$

Pipelining

Pipelining is a method for rapidity optimization, which decreases the latency of the operations by the initiation interval (II) decreasing for a function or loop. Figure 5 is a sample of the pipelining technique applied to a for-loop, which contains 3 operations: Rd, Cmp, and Wr. The left-hand side of Figure 5 is an operating sequence of the process without pipelining. Each iteration needs three clock cycles for processing. Six clock cycles are needed for two iterations processing due to the last operation of the first iteration must be completed before the first operation in the consecutive iteration starts. In contrast, pipelining is applied to the same for-loop with II equal to 1. In the second clock cycle, the first operation in the consecutive iteration can be immediately operated as shown on the

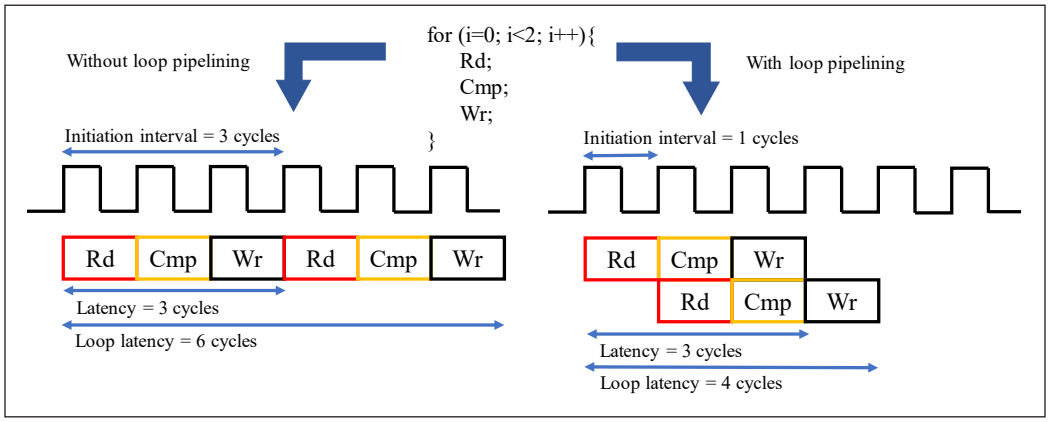

Figure 5. The example of loop pipelining for for-loop

right-hand side of Figure 5. The number of clock cycles for processing is reduced from six to four clock cycles.

Unrolling

Unrolling is a performance optimization technique as well. It allows some or all iterations to be concomitant. Figure 6 is an example of applying loop unrolling. Without unrolling, each element of the array needs 1 clock cycle for processing. Therefore, six iterations are needed to process six elements of array whereas only two iterations are needed in case of using unrolling with factor equals three due to three elements of array are processed in 1 clock cycle.

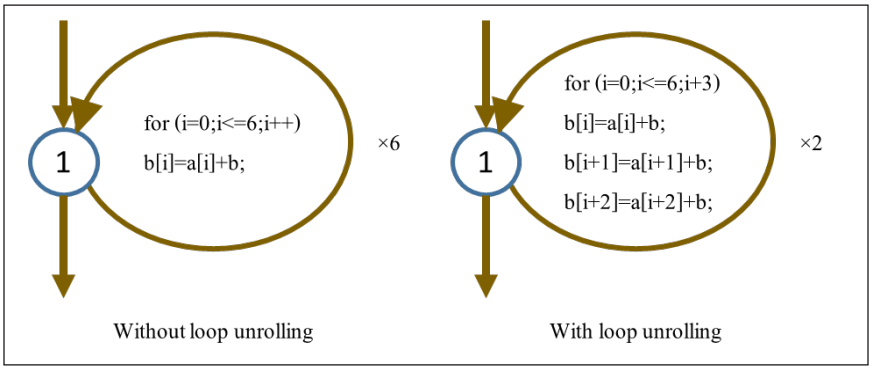

Figure 6. The example of loop unrolling for for-loop

Array Partitioning

Array partitioning is a method for speed and region optimization provided by Xilinx Vivado HLS. An original array is divided into sub-group of smaller arrays and stored into separate banks. Figure 7 demonstrates three types of array partitioning, which are block, cyclic, and complete types. In the case of the block type, a large array is divided into balanced blocks whose array elements are consecutively arranged. In the case of the cyclic type, a large array is divided into a balanced block and interleaved with elements. In the case of the complete type, an array is divided into single elements.

HLS Interface

Xilinx Vivado HLS supports the specification of I/O protocol types. Port interface is created by the synthesis of interface based on efficacy industry-standard interface and manual interface specifications, with the manner of the interface is illustrated in the input source code. Three port types on RTL design, including clock and reset port, block-level interface protocols, and port-level interface protocols are created by Xilinx Vivado. Typically, the

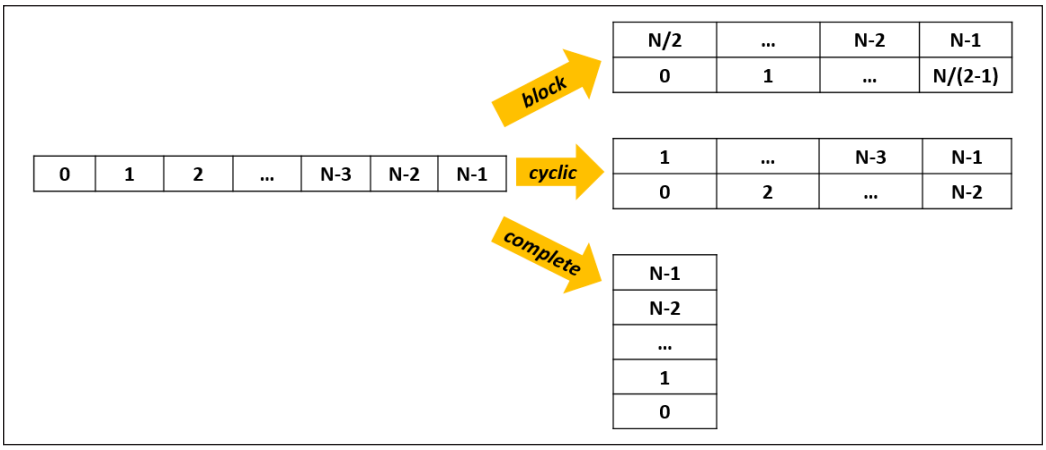

Figure 7. The type of array partitioning

protocol of the block-level interface is gathered in the design. These signals control the block are autonomous to port-level I/O protocols. These ports determine when the block can begin data processing, demonstrate when new inputs can be asserted new inputs, and demonstrate whether the system is idle or the operation completed. After the block-level protocol has been used to start the operation of the block, the port-level I/O protocols are used to sequence data into and out of the block.

METHODS

Road Detection Framework

A framework of lane tracking proposed in this paper is shown in Figure 8. There are four main steps: 1. pre-processing, 2. edge detection, 3. line detection, and 4. angle calculation. Edge detection and line detection are implemented on the PL part to increase the operation speed, and others are implemented on the PS part. Pre-processing expanded in Figure 9 includes image cropping, image dividing into two parts (left side image and right side image) to separate left and right line, an RGB to grayscale image conversion, and grayscale to binary image conversion. According to the result in Solod et al. (2018), Robert edge detection is selected to perform edge detection. Hough line transform is then

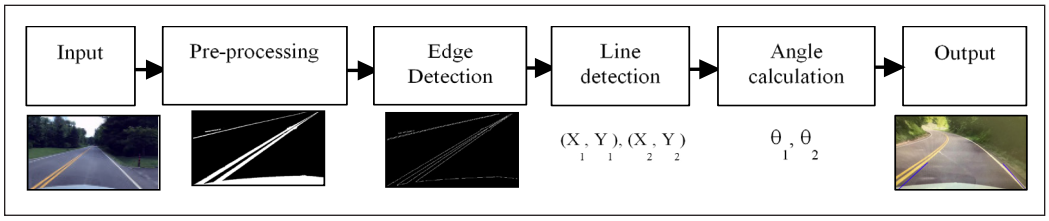

Figure 8. Road lane detection framework

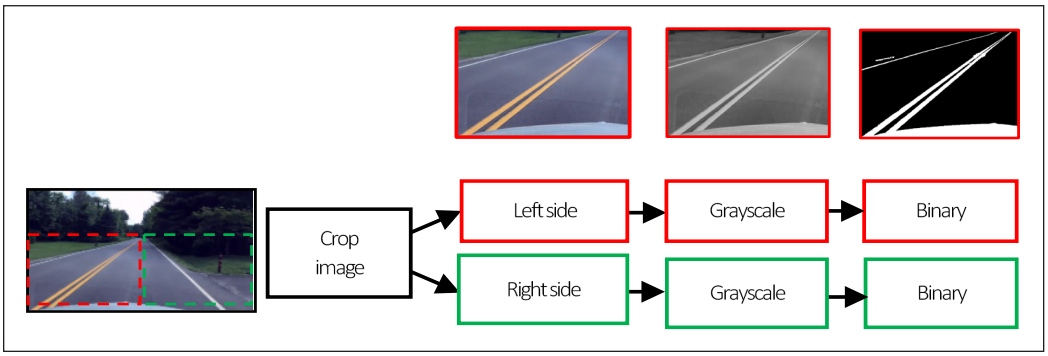

Figure 9. Pre-processing step of road lane detection

applied to the line detection. The positions of (x_1, y_1) and (x_2, y_2) obtained from the line detection step are used to calculate the angles (Θ_1 and Θ_2).

The angle calculation is divided into three cases to recognize the road path curve and prevent fallibility from the previous step. Θ_1 represents the angle of the left side and Θ_2 represents the angle of the right side. The first case is straight lane as shown in Figure 10, which Θ_1 must be over 90º and Θ_2 must be less than 90º. The second case is the left curve as shown in Figure 11, in which Θ_1 and Θ_2 must be less than 90º. The last case is the right curve as shown in Figure 12, in which Θ_1 and Θ_2 must be over 90º.

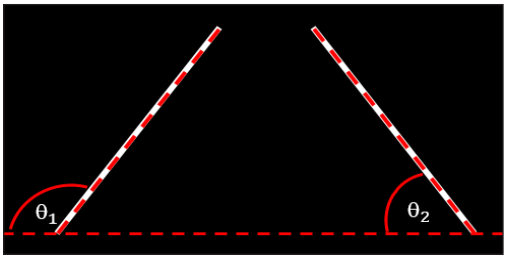

Figure 10. The measurement angle of straight lane

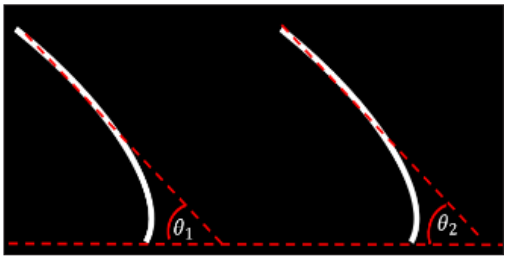

Figure 11. The measurement angle of left curve lane

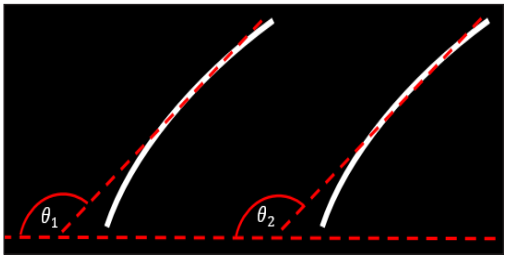

Figure 12. The measurement angle of right curve lane

HLS Optimization Processes

According to the operation time of road lane detection spends most time in the process. Edge detection and line detection are implemented on PL using HLS on Zynq-7000, which allows C and C++ languages and authorizes efficient method as loop unrolling, loop pipelining, or array

partitioning, etc. The proposed HLS optimization processes are divided into four steps as follows.

- Step 1: array sizing is performed to decrease the resource usage on PL part.
- Step 2: loop analysis is performed to determine which loop must be loop unrolling or loop pipelining.
- Step 3: array partitioning is performed to resolve the bottleneck of loop unrolling and loop pipelining.
- Step 4: HLS interface is performed to select the best block level and port level interface for array argument of RTL design.

These four HLS optimization processes should be done sequentially. In each process, an analysis method is proposed to consider the suitable optimization techniques should be applied to implement on an FPGA device. If the optimization processes are not done in suggested sequence, the optimization resistant result may be obtained. This behavior is illustrated in the experimental results and discussions section.

Array Sizing. The first point to consider is the array sizing. Analysis and calculation of the appropriate array size for block memory storage capacity can decrease over the allocation of block RAM (BRAM) by considering the amount of BRAM in Equation 4. At array sizing step, 2 parameters are considered. The first parameter is memory depth, which caused by the amount of array element. For example, consider a 16-bits 1-dimension array with 2050 word lines. Due to the requirement of the memory depth, the memory depth must over 2050. The at least memory depth for this array is 4096(2^{12}), the data width is 16 bits and the size of BRAM is 18K bits (18KBRAM). According to Equation 4, 4(2^2) 18K BRAM is allocated to keep the array. Likewise, the word line of the array is reduced to 2048 we need to allocate 4(2^2) BRAM 18K to store an array. Likewise, the size of the array is reduced to 2048, the amount of BRAM is decreased to 2(2^1).

In this work, we reduced the input image from 1920×1080 pixels to 200×200 pixels. From the Hough line transform theory, the maximum diagonal is reduced from 1445 to 224, which equates to the value of rho in the first dimension size of hough1[rho][theta] and hough2[rho][theta] array. The second parameter is the data width. According to the concept of Hough line transform, the value of hough1[rho][theta] and hough2[rho][theta] will not over the value of rho parameter, which has the maximum value to the diagonal line of the image, so the data width can be reduced from 16 bits to 8 bits. Theta also decreases from 0º-360º to 30º-150º. Thus the second dimension is reduced from 360 to 120, which is sufficient for road lane detection.

$$Block\ RAM = \frac{memory\ depth\ x\ data\ width}{block\ RAM\ size} \quad [4]$$

Loop Analysis. From the theory of Robert edge detection and Hough line transform, we summarize into four main nested loops that are agreeable to C language as shown in Figure 13. The first step is started with a 2-layer nested loop of a 16-bit-2-dimension array (hough1[rho][theta] and hough2[rho][theta]) is initialized to 0 for intersection times counting that starts with zero. The iteration of the inner layer equals the number of theta and the outer layer equals the number of rho. The second step is divided into two minor nested loops, which are a 4-layer nested loop for edge detection and a 3-layer nested loop to count intersection times in θ-ρ plane for every pixel. The third step is a 2-layer nested loop for the Hough line transform normalization by considering the value of all elements of hough1[rho][theta] and hough2[rho][theta]. The final step is a 3-layer nested loop for lane tracking by array hough1[rho][theta] and hough2[rho][theta] checking to determine the position of the line in the image. From all steps of edge detection and line detection in Figure 13, we found that the nested loop is in every step, which caused speed reduction. In this work, we consider each nested loop with an optimization technique that is appropriate with the nested loop type. Both 2-dimension arrays hough1[rho][theta] and hough2[rho][theta] are initialized in the first nested loop, which results in each iteration is independent. All array elements can be initialized simultaneously. Generally, as shown in Figure 14, two iterations in the first nested loop need four clock cycles to operate. In the case of loop unrolling is added, two iterations need only two clock cycles to operate. Edge detection in the first minor nested loop is dependent value in each iteration. According to Robert edge detection theory, both x_weight and y_weight are convolution results, which are cumulative variables. Therefore, loop unrolling is not necessary for the first minor nested loop.

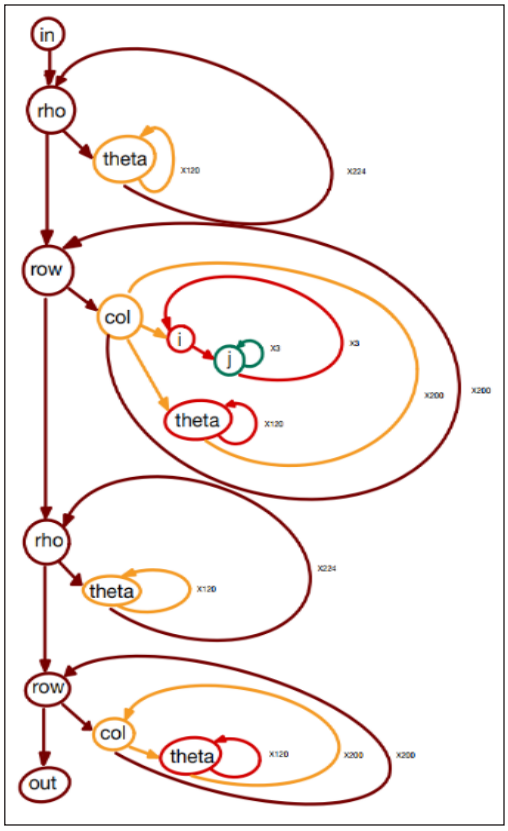

Figure 13. The nested loops of C language for edge and line detection

$$hough_{(1,2)}[rho][theta] = hough_{(1,2)}[rho][theta] + 1 \qquad [5]$$

The second minor nested loop is counting times of intersection in θ and ρ space according to the Hough line transform. Equation 5 demonstrates that hough1[rho][theta] and hough2[rho][theta] are cumulative variables, which is dependent on other iteration as well. Therefore, loop pipelining is appropriate to add in the second minor nested loop. The schedule of this step will be changed following Figure 15, without loop pipelining the next element of hough1[rho][theta] and hough2[rho][theta] cannot be started reading before the previous element finished processing, whereas adding loop pipelining, the next element of hough1[rho][theta] and hough2[rho][theta] can start reading before previous element finish processing.

hough1[rho][theta] and hough2[rho][theta] are normalized in third nested loop. This step is similar to the first nested loop, in which the result in each iteration is independent. Loop unrolling will get less latency than loop pipelining, thus loop unrolling should be selected if the resource is sufficient.

The value of every element of hough1[rho][theta] and hough2[rho][theta] is checked to mark the position of the lane so the result in each iteration is not influencing to the other. Loop unrolling should be selected as well as the third nested loop. However, loop

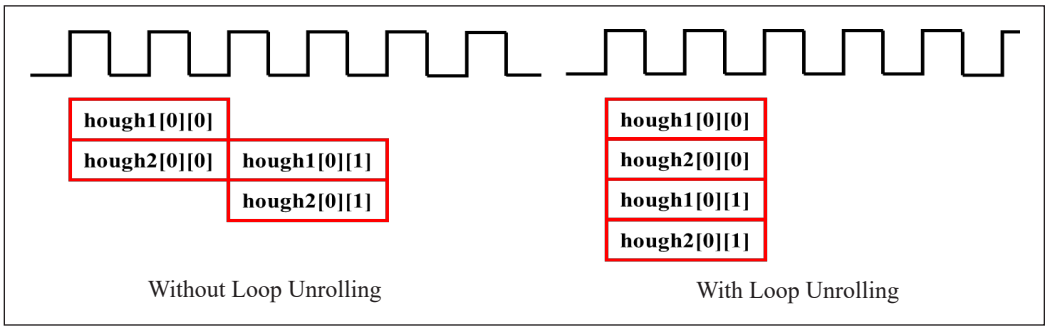

Figure 14. Loop unrolling applying in 1st nested loop

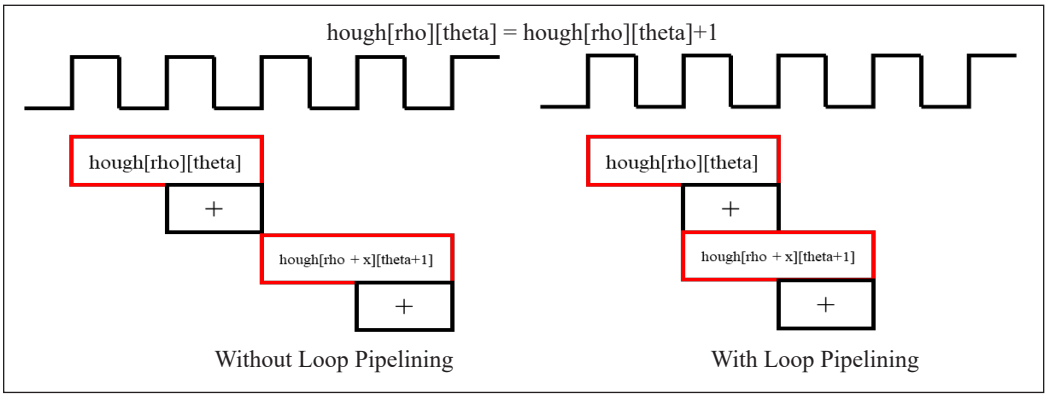

Figure 15. Loop pipelining applying in 2nd minor nested loop

pipelining must be selected instead of loop unrolling because of the limitation of resources on the device.

Array Partitioning for Unrolling and Pipelining. Array partitioning significantly reduces that can decrease the latency. This technique encourages loop pipelining and loop unrolling, such as in the second minor nested loop. Although loop pipelining is added, hough1[rho][theta] and hough2[rho][theta] are still stored in the same bank of memory, the bottleneck will occur because each BRAM allows only one read operation of one element at that time. Loop pipelining will not get the highest efficiency as expected. Thus, the second element (hough1[rho +x] [theta+1] and hough2[rho+x][theta+1]) of array hough1[rho][theta] and hough2[rho][theta] cannot be accessed until the first element (hough1[rho][theta] and hough2[rho][theta]) is successfully accessed. Due to the bottleneck issue, loop unrolling in Figure 14 cannot get the highest efficiency as well. We found that array partition can be resolve or ameliorate this problem. In this part, an array partitioning block type, F equal to 6, and D equal to 2 are applied to hough1[rho][theta] and hough2[rho][theta] in Figure 16. The original array on the top is stored in one bank of memory. This array is divided into 6 banks of memory after array partitioning is applied. Memory depth in each bank equals 8,192.

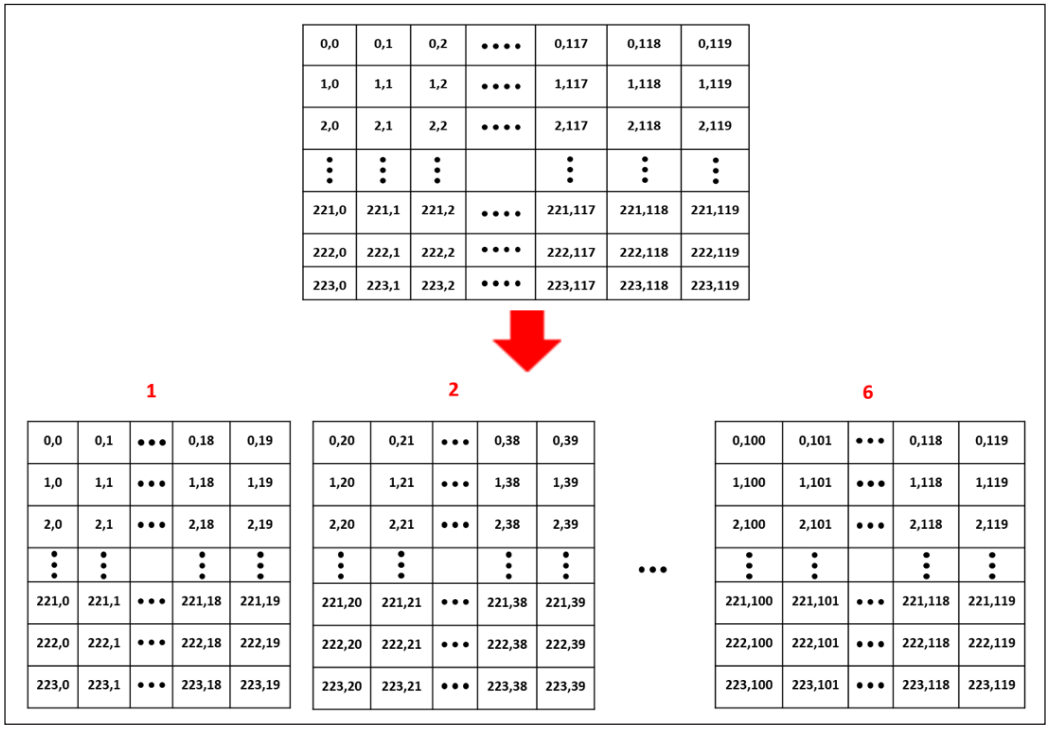

Figure 16. Array partitioning type of hough1[rho][theta] and hough2[rho][theta]

HLS interface for Array Type. Register Transfer Level (RTL) description is created by Xilinx HLS, in which an input/output operation must be performed through a port in the design. In this work, an RTL has 1-input port and 1-output port, which is all 1-dimension array. We consider port design that suitable for an array type. There are ap_ctrl_none, ap_ctrl_hs and ap_ctrl_chain for block-level interface. There are axis, s_axilite, m_axi, ap_hs, ap_memory, bram, ap_fifo, and ap_bus for data transmission.

RESULTS AND DISCUSSIONS

This experiment aims to increase the speed of road lane detection by adding HLS optimization techniques including array sizing, loop unrolling, loop pipelining, and HLS interface management. The comparison of resource usage and operating time are discussing as well.

Process Profiling

In the beginning, the profile of the operation time of each process in road lane detection is extracted on Intel® Core™ i7-7500U CPU @ 2.70 GHz, which input file has size 720x1280 pixels and frame rate 24 fps. The detailed profiling of process activities illustrates the time-consuming processes that must be implemented as hardware accelerators on the PL part. We found that the step of edge detection and line detection step spent most processing time as shown in Table 1.

Table 1
Comparison of operation time in each process

Optimization	Resource (%)				Latency
	DSP	BRAM	CLB	FF	
Default	17	1,710	11	4	15,847,183,592
Array sizing	13	57	11	3	103,709,185

Array Sizing Results

The results as shown in Table 1, we found that edge detection and line detection should select to be hardware accelerator to increase operation speed. However, the resource, which is spent on PL part is over the limitation. Therefore, array sizing should be considered by the method described in the previous section. The input image is cropped and resized from 1920×1080 to 200×200 pixels. Therefore, hough1[rho][theta] and hough2[rho][theta] is resized from 2203×120 to 224×120 according to HT theory, which in rho is the feasible perpendicular length of the input image. As shown in Table 2 latency is reduced about 152 times faster and BRAM is reduced form 1,710% to 57%.

Table 2
Comparison of default and array sizing

Lane detection process	Processing Time (%)	Processing Time (FPS)	Processing Time (s/frame)
Pre-processing	30.60	0.8928	0.1049
Edge detection	32.30	0.9424	0.1107
Line detection	36.10	1.0533	0.1238
Angle calculation	1.00	0.0292	0.0034

Loop Unrolling and Loop Pipelining Results

The optimization techniques, loop unrolling, and loop pipelining are considered into inner-nested loop of edge and line detection to analyze the suitable usability. In case of considering to least latency, loop unrolling is sufficient to add along with first, third, and last nested loop of Figure 13. Loop pipelining is sufficient to add along with the second nested loop (both of edge detection and line detection nested loop). Although loop unrolling can reduce the latency more than loop pipelining in third and last nested loop, the number of CLBs is more than the limitation. Therefore, loop pipelining is sufficient to add along with third and last nested loop instead of loop unrolling. The proposed method as shown in Table 3 is the combination of sufficient optimization techniques in all nested loop that can reduce latency at clock frequency 100 MHz from 103,709,185 clock cycles to 15,789,018 clock cycles or about 6.57 times is reduced. Both of loop unrolling and loop pipelining is latency reducing optimization by throughput increasing or initiation interval decreasing. In case of hough1[rho][theta], hough2[rho][theta] and two more arrays, sin[theta] and cos[theta],

Table 3
Comparison of loop unrolling and loop pipelining in each loop

Process		Method	Resource (%)				Latency (clock cycle)
			DSP	*BRAM*	*CLB*	*FF*	
All		Default	12	30	9	3	103,709,185
2nd	1st	Unrolling	12	30	27	3	103,695,297
		Pipelining	12	30	10	3	103,708,738
	Edge detection	Unrolling	13	30	9	3	102,739,561
		Pipelining	13	30	10	3	98,012,241
	Line detection	Unrolling	135	30	228	72	61,853,749
		Pipelining	13	30	10	3	46,056,556
	Edge + Line detection	Combination	15	30	10	3	45,046,530
	3rd	Unrolling	12	30	177	79	98,457,416
		Pipelining	12	30	15	6	98,457,432
	4th	Unrolling	11	30	372	56	72,509,185
		Pipelining	12	30	11	3	74,828,791
Proposed method (1)			15	30	34	6	15,789,018

which are constants stored in the same block of memory, the loop pipelining and loop unrolling cannot get the best efficiency because of the limitation of memory accessing.

Array Partitioning Results

Since hough1[rho][theta], hough2[rho][theta] and two more arrays, sin[theta] and cos[theta] are constants stored in the same block of memory, the loop pipelining and loop unrolling cannot get the best efficiency cause of the limitation of memory accessing Table 4 is the result of adding array partitioning to hough1[rho][theta], hough2[rho][theta], sin[theta] and cos[theta] compare to the first proposed method. Both hough1[rho][theta] and hough2[rho][theta] arrays are sufficient with array partitioning block type with F equal to 6 at equal to is 2. The latency from the experiment is reduced from 15,789,018 clock cycles to 15,777,837 clock cycles, which is only 0.99 times faster. Although Adding array partitioning complete type to these arrays instead of array partitioning block type would be faster, the resources on the device in this work will not enough to implement. In contrast, cos[theta] and sin[theta] are sufficient with array partitioning complete type and the latency is reduced from 15,789,018 clock cycles to 15,697,034 clock cycles compared to the second proposed method.

Table 4
Comparison of array partitioning

Optimization		Resource (%)				Latency (clock cycle)
		DSP	*BRAM*	*CLB*	*FF*	
Proposed method (1)		15	30	34	6	15,789,018
hough1 & hough2	block type, F 2, D 2	15	30	27	6	15,782,316
	block type, F 6, D 2	17	44	24	8	15,777,837
	block type, F 12, D 2	17	44	26	9	15,803,597
	cyclic type, F 2, D 2	15	30	28	6	25,774,041
	cyclic type, F 6, D 2	17	44	24	7	25,769,578
	cyclic type, F 12, D 2	15	30	28	6	25,774,041
sin & cos	Complete type	17	42	39	8	15,697,034
	Block type, F 2	18	42	27	9	15,777,845
	Block type, F 6	17	42	27	9	15,777,837
	Block type, F 12	17	42	27	9	15,777,846
Proposed method (2)		17	42	39	8	15,697,034

HLS Interface Management Results

In the case of the HLS interface, there are three types of block type interfaces available for arrays, namely ap_ctrl_none, ap_ctrl_hs, and ap_ctrl_chain. There are many types of port-level interfaces available for arrays, namely axis, s_axilite, m_axi, ap_hs, ap_memory, bram,

ap_fifo, and ap_bus. In this experiment, the block-level interface and port-level interface are matched to the input and output of RTL that Xilinx HLS has created. Following Figure 17, the first column represents a block-level interface, the second column represents a port-level interface for the input and the final column is represent a port-level interface for output. The result of the HLS interface management after all nested loop is added by the optimization technique is shown in Table 5, which compares to the default of a block-level interface and port-level interface, generated by Xilinx HLS. The least latency occurs with all of the block-level interfaces while the port-level interface of input should be axis and the port-level interface of output should be ap_memory.

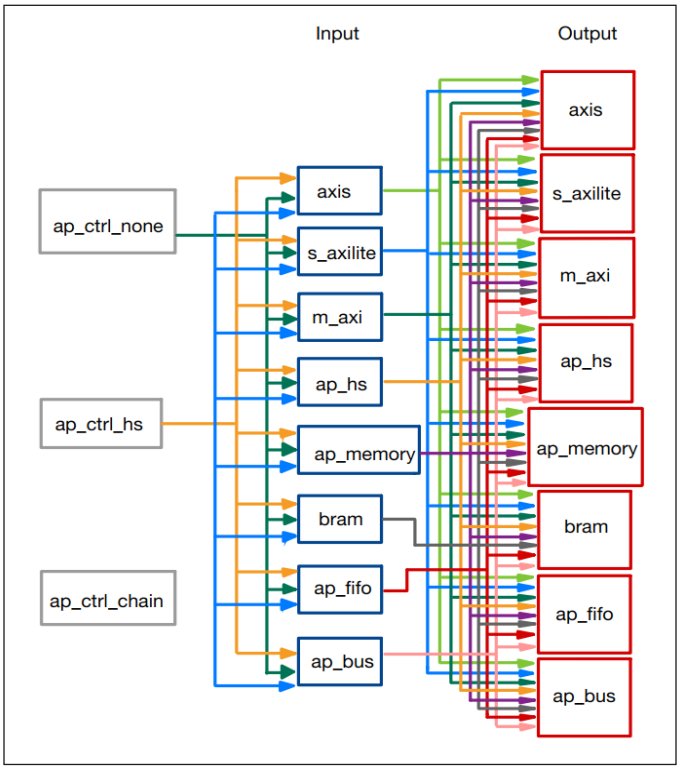

Figure 17. HLS interface matching for array type

Table 5
Comparison of HLS interface

Method	Latency (clock cycles)	Processing Performance (FPS)
Default	15,847,183,592	0.006
Proposed (1)	15,789,018	6.334
Proposed (2)	15,697,034	6.371
Proposed (3)	15,616,232	6.404

CONCLUSION

This paper has presented the optimization for complex road lane detection method, which supports lane curve recognition by angle calculation and loop analysis with optimization technique to increase the rapidity of operation underneath the limitation of resources on Xilinx Zynq-7000 (ZC7010).

The sequence of the optimization contains four steps: array sizing to reduce the resources, loop unrolling and loop pipelining to increase operation speed by parallel operation, array partitioning to prevent the bottleneck that can occur from the loop unrolling and loop pipelining step, and HLS interface management to get the best data transmission.

From the experiment, we found that an array sizing is suitable for low memory devices and adjustable to the method or experiment. Loop unrolling and loop pipelining is latency reducing optimization techniques that suitable to different kinds of loops. In the case of loop unrolling is suitable for dependent loops, which are loops that the result of each iteration in loop does not affect to other iteration. In case of loop pipelining is suitable for dependent loops, which are loops that the result of each iteration in loop effect to other iteration. Array partitioning can be both of area reducing and latency reducing optimization. This technique supports loop unrolling and loop pipelining to achieve efficiency as much as possible. The determination of factor (F) and dimension (D) of array partitioning depends on the sequence of the algorithm. The type of C argument, which is an input and output in RTL will be suitable with a different type of HLS interface.

An array sizing, loop unrolling, loop pipelining, and array partitioning have been applied to edge detection and line detection using Xilinx Vivado HLS. The summary resource and latency of the proposed optimization techniques compared to the default method are shown in Table 6. Following Tables 7 and 8, the resource utilization and speed are compared between the proposed method on Zybo-ZC7010 with the other algorithms on different architectures. Most of the performance in other algorithms on larger FPGA devices is faster according to the clock frequency but spent a lot of resources as well. Although the proposed method cannot be reached in real-time (25 FPS), it could reduce resource consumption and the latency from 103,951,104 clock cycles to 15,616,232 clock cycles, or latency is reduced 6.66 times at clock frequency 100 MHz.

In addition, the proposed method in this work can reach higher performance by implementing on the high-performance devices. The optimizations can further gain the benefit on the devices with sufficient resources. Since our procedure optimization method only considered the inner-layer of nested-loop. Therefore, the speed performance still can be increased by nested-loop layer analysis along with the procedure of the proposed method that can be an instruction for the HLS developers.

Table 6
The summary of the proposed optimization techniques compared the default method

Optimization	Resource				Latency (clock cycle)
	DSP	*BRAM*	*CLB*	*FF*	
Default	17	42	39	8	15,697,034
HLS interface	16	42	39	8	15,616,232
Proposed method (3)	16	42	39	8	15,616,232

Table 7
The resource utilization comparison of the proposed method with other methods on different architectures

Architectures	Resource Utilization				
	Slices	*LUTs*	*On-chip Memory (Kbit)*	*DSP*	Embedded Multiplier
Virtex-5 (ML505) (El Hajjouji et al., 2020)	1,119	1,996	1,625	1	1
Altera DE2 (Marzotto et al., 2010)	14,945	14,945	1,555	-	15
Altera DE2-115 (Lu et al., 2013)	29,431	2,589	3,052	8	8
Altera Stratix (Guan et al., 2017)	41,115	1,459	1,604	48	16
Zybo (ZC7010-1) (proposed)	2,816	6,864	101	13	-

Table 8
The speed comparison of the proposed method with other methods on different architectures

Method	F (MHz)	Resolution	Performance (FPS)
Virtex-5 (ML505) (El Hajjouji et al., 2020)	200	640 × 480	68
Altera DE2 (Marzotto et al., 2010)	115	1920 × 1080	25
Altera DE2-115 (Lu et al., 2013)	200	1,024 × 768	64
Altera Stratix (Guan et al., 2017)	200	1,024 × 768	185
Zybo (ZC7010-1) (Proposed)	100	1920 × 1080	6

ACKNOWLEGEMENTS

This research is financed by Prince of Songkla University, Hat Yai, Songkhla, Thailand.

REFERENCES

Chen, Y., & Boukerche, A. (2020). A Novel Lane Departure Warning System for Improving Road Safety. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE Conference Publishing. https://doi.org/10.1109/ICC40277.2020.9149085

El Hajjouji, I., Mars, S., Asrih, Z., & El Mourabit, A. (2020). A novel FPGA implementation of hough transform for straight lane detection. *Engineering Science and Technology, an International Journal*, *23*(2), 274-280. https://doi.org/10.1016/j.jestch.2019.05.008

Feniche, M., & Mazri, T. (2019). Lane detection and tracking for intelligent vehicles: A survey. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)* (pp. 1-4). IEEE Conference Publishing. https://doi.org/10.1109/ICCSRE.2019.8807727.

Guan, J., An, F., Zhang, X., Chen, L., & Mattausch, H. J. (2017). Real-time straight-line detection for XGA-size videos by hough transform with parallelized voting procedures. *Sensors*, *17*(2), Article 270. https://doi.org/10.3390/s17020270

Hwang, S., & Lee, Y., (2016). FPGA-based real-time lane detection for advanced driver assistance systems. In *Proceedings of 2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)* (pp. 218-219). IEEE Conference Publishing. https://doi.org/10.1109/APCCAS.2016.7803937

Khongprasongsiri, C., Kumhom, P., Suwansantisuk, W., Chotikawanid, T., Chumpol, S., & Ikura, M. (2018). A hardware implementation for real-time lane detection using high-level synthesis. In *2018 International Workshop on Advanced Image Technology (IWAIT)* (pp. 1-4). IEEE Conference Publishing. https://doi.org/10.1109/IWAIT.2018.8369730

Lee, D. K., Shin, J. S., Jung, J. H., Park, S. J., Oh, S. J., & Lee, I. S. (2017). Real-time lane detection and tracking system using simple filter and Kalman filter. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)* (pp. 275-277). IEEE Conference Publishing. https://doi.org/10.1109/ICUFN.2017.7993792

Lu, X., Song, L., Shen, S., He, K., Yu, S., & Ling, N. (2013). Parallel hough transform-based straight line detection and its FPGA implementation in embedded vision. *Sensors, 13*(7), 9223-9247. https://doi.org/10.3390/s130709223

Marzotto, R., Zoratti, P., Bagni, D., Colombari, A., & Murino, V. (2010). A real-time versatile roadway path extraction and tracking on an FPGA platform. *Computer Vision and Image Understanding*, *114*(11), 1164-1179. https://doi.org/10.1016/j.cviu.2010.03.015

Panda, P. R., Sharma, N., Kurra, S., Bhartia, K. A., & Singh, N. K. (2018). Exploration of loop unroll factors in high level synthesis. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)* (pp. 465-466). IEEE Conference Publishing. https://doi.org/10.1109/VLSID.2018.115

Promrit, P., & Suntiamorntut, W. (2017, July). Design and development of lane detection based on FPGA. In *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)* (pp. 1-4). IEEE Conference Publishing. https://doi.org/10.1109/JCSSE.2017.8025909

Solod, P., Sengchuai, K., Booranawong, A., Hoyingcharoen, P., Chumpol, S., Ikura, M., & Jindapetch, N. (2018, October 30 - November 2). Model based design approach for road lane tracking [Paper presentation]. In *Asia Pacific Conference on Robot IoT System Development and Platform 2018* (APRIS2018). Prince of Songkla University (PSU) Phuket Campus, Thailand.