



กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูงสำหรับ
การปรับปรุงระบบตรวจจับเลนถนน
**Digital Circuit Optimization Process Using High Level Synthesis for Road Land
Detection Improvement**

ปนัดดา โสพล

Panadda Solod

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า
มหาวิทยาลัยสงขลานครินทร์

**A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering
Prince of Songkla University**

2563

ลิขสิทธิ์ของมหาวิทยาลัยสงขลานครินทร์



กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูงสำหรับ
การปรับปรุงระบบตรวจจับเลนถนน
**Digital Circuit Optimization Process Using High Level Synthesis for Road Land
Detection Improvement**

ปนัดดา โสพล
Panadda Solod

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า
มหาวิทยาลัยสงขลานครินทร์

**A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering
Prince of Songkla University**

2563

ลิขสิทธิ์ของมหาวิทยาลัยสงขลานครินทร์

ชื่อวิทยานิพนธ์ กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่
ระดับสูงสำหรับการปรับปรุงระบบตรวจจับเลนถนน

ผู้เขียน นางสาวปนัดดา โสภส

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

คณะกรรมการสอบ

.....
(รองศาสตราจารย์ ดร. ณัฐฐา จินดาเพ็ชร)

.....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.คุณดาว บุรณะพาณิชย์
กิจ)

.....กรรมการ
(ดร.กิตติคุณ ทองพูล)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.พิทักษ์ บุญนุ่น)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้บัณฑิตวิทยาลัย
เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชา
วิศวกรรมไฟฟ้า

.....
(ศาสตราจารย์ ดร.ดำรงศักดิ์ ฟ้ารุ่งแสง)

คณบดีบัณฑิตวิทยาลัย

ขอรับรองว่า ผลงานวิจัยนี้เป็นผลมาจากการศึกษาวิจัยของนักศึกษาเอง และขอแสดงความขอบคุณ
บุคคลที่มีส่วนเกี่ยวข้อง

ลงชื่อ.....

(รองศาสตราจารย์ ดร. ฉัฐฐา จินดาเพชร)

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

ลงชื่อ.....

(นางสาวปนัดดา โสพต)

นักศึกษา

(4)

ข้าพเจ้าขอรับรองว่า ผลงานวิจัยนี้ไม่เคยเป็นส่วนหนึ่งในการอนุมัติปริญญาในระดับใดมาก่อนและ
ไม่ได้ถูกใช้ในการยื่นขออนุมัติปริญญาในขณะนี้

ลงชื่อ.....

(นางสาวปนัดดา โสพล)

นักศึกษา

ชื่อวิทยานิพนธ์	กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูงสำหรับการปรับปรุงระบบตรวจจับเลนถนน
ผู้เขียน	นางสาวปนัดดา โสฬส
สาขาวิชา	วิศวกรรมไฟฟ้า
ปีการศึกษา	2563

บทคัดย่อ

ในงานวิจัยนี้มีวัตถุประสงค์เพื่อพัฒนาการออกแบบระบบตรวจจับเลนถนนซึ่งเป็นส่วนหนึ่งของ ระบบช่วยรักษาเลน (Lane Keeping Assistant System: LKAS) และระบบเตือนออกนอกเลน (Lane Departure Warning System: LDWS) โดยตัวแปรสำคัญของระบบนี้คือตำแหน่งของเลนถนนจริงบนภาพที่หาได้ในกรณีที่สภาพแวดล้อมเปลี่ยนแปลงไป โดยเฉพาะอย่างยิ่งกรณีเลนถนนทางโค้งซึ่งในงานวิจัยนี้ได้ปรับใช้การคำนวณมุมที่ได้จากเลนบนถนนเพื่อแยกถนนที่มีลักษณะเป็นทางโค้งซ้ายและขวาทำให้เพิ่มความแม่นยำในการตรวจจับเลนถนนจริงและยังสามารถใช้เป็นตัวแปรสำหรับการควบคุมในระบบ LKAS และ LDWS ต่อไป นอกจากออกแบบระบบตรวจจับเลนถนนแล้วในงานวิจัยนี้ได้ทำการพัฒนาความเร็วและจัดการทรัพยากรที่ใช้ของอัลกอริทึมโดยการนำเสนอกระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูง (High Level Synthesis: HLS) โดยมีขั้นตอนตามลำดับดังนี้ การปรับขนาดอาร์เรย์ (array sizing), การคลี่ลูป (loop unrolling) การทำไปป์ไลน์ลูป (loop pipelining) การแบ่งอาร์เรย์ (array partitioning) และการจัดการอินเทอร์เฟซ HLS (HLS interface) ภายใต้อำนาจของทรัพยากรที่มีบนอุปกรณ์และความเร็วที่ใช้ในการประมวลผลบนชิพเอฟพีจีเอตระกูล Xilinx Zynq-7000 (Zybo z7-10) จากการทดลองพบว่าวิธีการที่นำเสนอในงานวิจัยนี้สามารถเพิ่มความเร็วที่ใช้ในการประมวลผลจากเดิมขึ้น 6.66 เท่า ที่ความถี่ของนาฬิกา 100 MHz

คำสำคัญ การสังเคราะห์ระดับสูง, เอฟพีจีเอ (อุปกรณ์ลอจิกแบบโปรแกรมได้), การคลี่ลูป, การทำไปป์ไลน์ลูป, การแบ่งอาร์เรย์, อินเทอร์เฟซ HLS

Thesis Title	Digital Circuit Optimization Process Using High Level Synthesis for Road Lane Detection Improvement
Author	Ms. Panadda Solod
Major Program	Electrical Engineering
Academic Year	2020

ABSTRACT

In this work, road lane detection is proposed to reciprocate the requirements of Lane Keeping Assistant System (LKAS) and Lane Departure Warning System (LDWS), which are the position of lane line on image and the tolerance to the unexpected road lane, especially curve lane. The angle calculation is proposed to realize the curve's direction. The speed and memory usage of an algorithm are improved as well by adding the High Level Synthesis (HLS) optimization techniques. Array sizing, loop unrolling, loop pipelining, array partitioning and HLS interface management are respectively applied according to the limitation of resources and the speed of operation time using HLS development on Xilinx Zynq-7000 family (Zybo z7-10). From the experimental results, the proposed method reaches 6.66 times faster than the original at clock frequency 100 MHz.

Keyword High Level Synthesis (HLS), FPGA, loop unrolling, loop pipelining, array partitioning, HLS interface

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จสมบูรณ์ได้ด้วยดีเพราะได้รับความกรุณาชี้แนะและช่วยเหลืออย่างดียิ่งจาก รองศาสตราจารย์ ดร.ณัฐฐา จินดาเพ็ชร์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.คุณดาว บุรณะพานิชย์กิจ ประธานกรรมการสอบวิทยานิพนธ์ และดร.กิตติคุณ ทองพูล และผู้ช่วยศาสตราจารย์ ดร.พิทักษ์ บุญนุ่น กรรมการผู้ทรงคุณวุฒิสอบวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.รักกฤตว์ ดวงสร้อยทอง ประธานคณะกรรมการบริหารหลักสูตรที่ให้คำแนะนำและตรวจแก้ไขข้อบกพร่องมาโดยตลอด ตั้งแต่เริ่มต้นจนสำเร็จเรียบร้อย ผู้วิจัยขอกราบขอบพระคุณด้วยความเคารพอย่างสูงไว้ ณ โอกาสนี้

ขอขอบพระคุณ อาจารย์เกียรติศักดิ์ เส็งช่วย และอาจารย์ภาคภูมิ หอยิ่งเจริญ และ ดร.อภิเดช บุรณวงษ์ ที่กรุณาให้คำแนะนำการทำวิทยานิพนธ์

ขอขอบพระคุณ บัณฑิตวิทยาลัย และ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์ ที่สนับสนุนเงินอุดหนุนวิจัย

ขอขอบพระคุณบิดา มารดา ที่สนับสนุนและให้กำลังใจจนงานวิจัยสำเร็จด้วยดี คุณค่าและประโยชน์อันพึงมีจากการศึกษาวิจัยนี้ ผู้วิจัยขอน้อมบูชาพระคุณบิดามารดาและบูรพาจารย์ทุกท่าน ที่ได้อบรมสั่งสอนวิชาความรู้ และให้ความเมตตาแก่ผู้วิจัยมาโดยตลอด และเป็นกำลังใจสำคัญ ที่ทำให้การศึกษาวิจัยฉบับนี้สำเร็จลุล่วงได้ด้วยดี

นางสาวปนัดดา โสภส

สารบัญ

สารบัญ	(8)
รายการตาราง	(11)
รายการภาพประกอบ	(12)
บทที่ 1 บทนำ	1
1.1 ความสำคัญและที่มาของงานวิจัย	1
1.2 การตรวจสอบเอกสาร บทความและงานวิจัยที่เกี่ยวข้อง	2
1.3 วัตถุประสงค์ของโครงการ	9
1.4 ประโยชน์ที่คาดว่าจะได้รับ	9
1.5 ขอบเขตการวิจัย	9
1.6 วิธีการวิจัย	10
1.7 แผนการดำเนินงาน	11
1.8 สถานที่ทำวิจัย	12
บทที่ 2 ทฤษฎีและหลักการ	13
2.1 เทคนิคการหาขอบภาพ	13
2.1.1 วิธีการหาขอบภาพของ Prewitt	13
2.1.2 วิธีการหาขอบภาพของ Sobel	14
2.1.3 วิธีการหาขอบภาพของ Robert	15
2.1.4 วิธีการหาขอบภาพของ Canny	15
2.2 เทคนิคการหาเส้นตรงโดยใช้ Hough Transform (HT)	16
2.3 เทคนิคการเพิ่มประสิทธิภาพ	17
2.3.1 เทคนิคการคลี่ลูป (Loop unrolling)	17
2.3.1 เทคนิคการไปป์ไลน์ลูป (Loop pipelining)	17
2.3.2 การแบ่งอาร์เรย์ (Array partitioning)	18
2.3.3 อินเทอร์เฟซ AXI (AXI interface)	18
2.3.4 การจัดการอินเทอร์เฟซ HLS (HLS interface management)	19
บทที่ 3 กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูง	24
3.1 การออกแบบระบบตรวจจับเลนถนน	24

สารบัญ (ต่อ)

3.1.1 การเตรียมภาพ (Pre-processing)	24
3.1.2 การหาขอบภาพ (Edge detection)	24
3.1.3 การหาเส้นตรง (Line detection)	25
3.1.4 การคำนวณมุมของเส้นตรง (Angle calculation)	25
3.2 การออกแบบระบบบนอุปกรณ์ฮาร์ดแวร์	27
3.3 เครื่องมือที่ใช้สำหรับออกแบบ	28
3.4 กระบวนการเพิ่มประสิทธิภาพใน High-Level Synthesis	34
3.4.1 การกำหนดขนาดอาร์เรย์	35
3.4.2 การวิเคราะห์ลูป	37
3.4.3 การแบ่งอาร์เรย์สำหรับการคลี่ลูปและการไปป์ไลน์ลูป	41
3.4.4 การจัดการอินเทอร์เฟส HLS สำหรับอาร์เรย์	45
3.5 สรุป	47
บทที่ 4 ผลการทดลองและการวิเคราะห์	49
4.1 ผลการทดลองและวิเคราะห์ระยะเวลาในการประมวลผลแต่ละขั้นตอนสำหรับระบบตรวจจับ เลนถนน	49
4.2 ผลการทดลองและวิเคราะห์การใช้วิธีการหาขอบภาพต่างๆของระบบตรวจจับเลนถนน	50
4.3 ผลการทดลองและวิเคราะห์ระยะเวลาในการใช้เทคนิคการเพิ่มประสิทธิภาพระบบที่ระดับสูง HLS สำหรับ HA	50
4.3.1 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 ในการพิจารณาถึง การปรับขนาดของอาร์เรย์	51
4.3.2 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 สำหรับขั้นตอนการ วิเคราะห์ลูป	52
4.3.3 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 สำหรับวิธีการแบ่ง อาร์เรย์	55
4.3.4 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 ในการใช้อินเทอร์เฟส HLS	56
4.3.5 ผลการจำลองประสิทธิภาพเพิ่มเติมบน xczu9eg	57

สารบัญ (ต่อ)

บทที่ 5 บทสรุปและข้อเสนอแนะ	66
5.1 สรุป	66
5.2 ปัญหา	67
5.3 ข้อเสนอแนะ	67
บรรณานุกรม	68
ภาคผนวก 1	70
ภาคผนวก 2	73
ภาคผนวก 3	78
ประวัติผู้เขียน	118

รายการตาราง

ตารางที่ 1 - 1 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม	4
ตารางที่ 1 - 2 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)	5
ตารางที่ 1 - 3 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)	6
ตารางที่ 1 - 4 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)	7
ตารางที่ 1 - 5 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)	8
ตารางที่ 1 - 6 แผนการดำเนินการวิจัย	11
ตารางที่ 2 - 1 อินเทอร์เน็ต HLS สำหรับแต่ละชนิดของข้อมูล	23
ตารางที่ 4 - 1 การเปรียบเทียบระยะเวลาในการประมวลผลของแต่ละกระบวนการ	50
ตารางที่ 4 - 2 การเปรียบเทียบระยะเวลาและความถูกต้องของวิธีการหาขอบภาพแต่ละวิธีสำหรับระบบตรวจจับเลนถนน	52
ตารางที่ 4 - 3 การเปรียบเทียบระยะเวลาและจำนวนทรัพยากรที่ใช้ก่อนและหลังการทำการปรับขนาดของอาร์เรย์	53
ตารางที่ 4 - 4 การเปรียบเทียบวิธีการคลี่รูปและการไปป์ไลน์รูปในแต่ละรูป	55
ตารางที่ 4 - 5 การเปรียบเทียบระยะเวลาและทรัพยากรในการทำแบ่งอาร์เรย์	58
ตารางที่ 4 - 6 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	59
ตารางที่ 4 - 7 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	60
ตารางที่ 4 - 8 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	61
ตารางที่ 4 - 9 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	62
ตารางที่ 4 - 10 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	63
ตารางที่ 4 - 11 ผลการทดลองการเลือกใช้อินเทอร์เน็ต HLS	64
ตารางที่ 4 - 12 การเปรียบเทียบทรัพยากรและระยะเวลาในการจัดการอินเทอร์เน็ต HLS	65
ตารางที่ 4 - 13 การเปรียบเทียบประสิทธิภาพการทำงานในแต่ละวิธีการในหน่วย FPS	65
ตารางที่ 4 - 14 ผลการเปรียบเทียบระยะเวลาและทรัพยากรบน Zybo z7-10 และ xczu9eg	65

รายการภาพประกอบ

ภาพประกอบ 1.1 วิธีการวิจัย	10
ภาพประกอบ 2.1 โอเพอร์เรเตอร์ของ Prewitt	14
ภาพประกอบ 2.2 โอเพอร์เรเตอร์ของ Sobel	14
ภาพประกอบ 2.3 โอเพอร์เรเตอร์ของ Robert	15
ภาพประกอบ 2.4 Hough Transform (HT)	16
ภาพประกอบ 2.5 ตัวอย่างเทคนิคการคลี่ดูป	18
ภาพประกอบ 2.6 ตัวอย่างการทำไปป์ไลน์ดูป	18
ภาพประกอบ 2.7 ประเภทของการแบ่งอาร์เรย์	20
ภาพประกอบ 2.8 รูปแบบการส่งข้อมูลของ AXI4	21
ภาพประกอบ 2.9 รูปแบบการส่งข้อมูลแบบ AXI4-Lite	22
ภาพประกอบ 2.10 รูปแบบการส่งข้อมูลแบบ AXI-Stream	22
ภาพประกอบ 3.1 การออกแบบระบบตรวจจับเลนถนน	24
ภาพประกอบ 3.2 ขั้นตอนการเตรียมภาพ	24
ภาพประกอบ 3.3 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางโค้งซ้าย	26
ภาพประกอบ 3.4 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางโค้งขวา	26
ภาพประกอบ 3.5 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางตรง	26
ภาพประกอบ 3.6 ระบบที่ใช้สำหรับทดสอบ	27
ภาพประกอบ 3.7 การออกแบบฮาร์ดแวร์ด้วยบอร์ด Zybo z7-10 ตระกูล Zynq-7000	28
ภาพประกอบ 3.8 การออกแบบแพลตฟอร์มด้วยบอร์ด Zybo z7-10 ตระกูล Zynq-7000	28
ภาพประกอบ 3.9 ลำดับการใช้เครื่องมือสำหรับออกแบบระบบตรวจจับเลนถนน	29
ภาพประกอบ 3.10 Xilinx Vivado HLS design flow	29
ภาพประกอบ 3.11 ตัวอย่างการใช้งาน HLS pragmas ในรหัสต้นฉบับ	30
ภาพประกอบ 3.12 ตัวอย่างการใช้งาน HLS pragmas ในแถบคำสั่ง	30
ภาพประกอบ 3.13 การ Run C Synthesis	31
ภาพประกอบ 3.14 การวิเคราะห์ข้อมูลบน Vivado HLS Analysis	31
ภาพประกอบ 3.15 ตัวอย่าง Performance Profile	32
ภาพประกอบ 3.16 SDSoC Development Environment	32
ภาพประกอบ 3.17 การทำโปรไฟล์บน SDSoC	33

รายการภาพประกอบ (ต่อ)

ภาพประกอบ 3.18 การกำหนดฟังก์ชันเป็น HA	34
ภาพประกอบ 3.19 การประมาณค่าการทำงานของ HA	35
ภาพประกอบ 3.20 โปรแกรมภาษาซีของการหาขอบภาพและเส้นบนภาพ	38
ภาพประกอบ 3.21 ลูปขั้นตอนการทำงานของระบบตรวจจับเลนถนน	39
ภาพประกอบ 3.22 การคลี่ลูปสำหรับขั้นตอนการกำหนดค่าเริ่มต้นของอาร์เรย์ของภาพประกอบ 3.21	40
ภาพประกอบ 3.23 การทำไปป์ไลน์ลูปสำหรับขั้นตอนการหาขอบภาพและการหาเส้นตรงในภาพประกอบ 3.21	41
ภาพประกอบ 3.24 ตัวอย่างการคลี่ลูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2×2 ที่ลูปชั้นนอก	41
ภาพประกอบ 3.25 การแบ่งอาร์เรย์แบบบล็อกในมิติที่ 2	42
ภาพประกอบ 3.26 ตัวอย่างการคลี่ลูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2×2 ที่ลูปชั้นใน	42
ภาพประกอบ 3.27 การแบ่งอาร์เรย์แบบวงกลมในมิติที่ 2	43
ภาพประกอบ 3.28 ตัวอย่างการคลี่ลูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2×2 ที่ลูปชั้นในพร้อมกัน 4 อิลีเมนต์	43
ภาพประกอบ 3.29 การแบ่งอาร์เรย์แบบสมบูรณ์ในมิติที่ 2	44
ภาพประกอบ 3.30 การแบ่งอาร์เรย์สำหรับตัวแปร $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$	45
ภาพประกอบ 3.31 IP block	46
ภาพประกอบ 3.32 การจับคู่อินเทอร์เฟส HLS	47
ภาพประกอบ 4.1 การทดสอบความถูกต้องของระบบตรวจจับเลนถนน	51
ภาพประกอบ 4.2 ระดับลูปซ้อนลูปสำหรับทดสอบการคลี่ลูปและการไปป์ไลน์ลูป บน Zybo z7-10	52
ภาพประกอบ 4.3 ระดับลูปซ้อนลูปสำหรับทดสอบการคลี่ลูปและการไปป์ไลน์ลูปบน xczu9eg	54

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของงานวิจัย

ระบบช่วยการขับขี่ขั้นสูง (Advanced Driving Assistant Systems: ADAS) เป็นการปรับปรุงระบบการขับขี่รถยนต์อัตโนมัติ โดยอาศัยระบบรักษาเลน (Lane Keeping Assistant System: LKAS) หรือระบบเตือนออกนอกเลน (Lane Warning Departure System: LDWS) ต่างก็เป็นส่วนหนึ่งของ ADAS ซึ่งทั้งสองระบบนี้มีการปรับใช้การตรวจจับเลนถนน (Road lane detection) เพื่อควบคุมระบบ โดยมีตำแหน่งเส้นของเลนบนถนนเป็นพารามิเตอร์ที่สำคัญที่สุดสำหรับการควบคุมและการเตือนในการทำงานของ LKAS และ LDWS

การทำงานของระบบตรวจจับเลนถนนที่ดีมีประสิทธิภาพควรเป็นระบบที่สามารถตรวจจับเส้นบนถนนได้เมื่อสภาพแวดล้อมเปลี่ยนแปลงไปไม่ว่าจะเป็นการตรวจจับขณะที่มีวัตถุหรือพาหนะเคลื่อนที่ การตรวจจับเลนที่มีคุณภาพสีของเลนไม่ชัดเจนและการตรวจจับเลนบนเส้นทางคดเคี้ยว นอกจากนี้การทำงานของระบบตรวจจับเลนถนนต้องมีการคำนึงถึงระยะเวลาที่ใช้ในการประมวลผลของระบบที่จะต้องให้ใกล้เคียงหรือเทียบเท่าเรียลไทม์มากที่สุด ซึ่งการประมวลผลแบบเรียลไทม์สามารถทำได้โดยการพัฒนาอัลกอริทึมให้มีความซับซ้อนน้อยลงเพื่อลดระยะเวลาที่ใช้ในการประมวลผลของระบบดังเช่นอัลกอริทึมของการหาขอบภาพและการหาเส้นตรงของภาพที่ค่อนข้างซับซ้อนมากแต่มีความจำเป็นต่อการทำระบบตรวจจับเลนถนน หรือการดำเนินการระบบลงบนอุปกรณ์ฮาร์ดแวร์ที่มีประสิทธิภาพการทำงานสูงอย่างเช่น IMX6Q หรือ Xilinx Zynq-7000 เป็นต้น

การพัฒนาของระบบตรวจจับเลนถนนมีรูปแบบของอัลกอริทึมหลากหลายวิธีการด้วยกัน ทั้งการใช้ตัวกรองอย่างง่ายเข้ามาช่วยในขั้นตอนการเตรียมภาพ [6] การปรับใช้การปรับมุมมองของภาพ [7] การลดขนาดของมุมในขั้นตอน Hough Transform [HT] [11] การใช้ปัญญาประดิษฐ์ [4] การเลือกใช้วิธีการหาขอบภาพ [7] นอกจากการพัฒนาอัลกอริทึมเพื่อลดความซับซ้อนจะช่วยให้การทำงานของระบบเข้าใกล้เรียลไทม์มากขึ้นแล้ว การเลือกฮาร์ดแวร์สำหรับพัฒนาอัลกอริทึมก็เป็นปัจจัยสำคัญ [8] และ [9] เป็นการพัฒนาอัลกอริทึมของระบบตรวจจับเลน

ถนนบนอุปกรณ์เอฟพีจีเอ โดยการดำเนินการบนเอฟพีจีเอสามารถเพิ่มความเร็วในการประมวลผลได้แต่ยังมีปัจจัยอื่นๆบนเอฟพีจีเอที่สามารถช่วยเพิ่มความเร็วในการประมวลผลคือการออกแบบและใช้อินเทอร์เฟสที่เหมาะสมกับลักษณะของข้อมูล [1] - [3] และการปรับใช้วิธีการไปป์ไลน์ลูปที่ระดับ HLS จากวิธีการที่กล่าวไปข้างต้นเป็นวิธีการที่ช่วยเพิ่มประสิทธิภาพการทำงานของระบบตรวจจับเลนถนนและการเพิ่มประสิทธิภาพการทำงานของระบบตรวจจับเลนบนเอฟพีจีเอซึ่งค่อนข้างซับซ้อนและยังขาดการวิเคราะห์เพื่อวิธีการเพิ่มประสิทธิภาพอื่นๆร่วมด้วย

ดังนั้นในงานวิจัยนี้จะมีการดำเนินการออกแบบระบบตรวจจับเลนถนนให้มีการรองรับการตรวจจับเลนถนนที่สภาพแวดล้อมเปลี่ยนแปลงได้มากขึ้นและทำการเพิ่มประสิทธิภาพการทำงานของระบบให้สามารถประมวลผลได้รวดเร็วกว่าระบบเดิมโดยการปรับใช้เทคนิคการเพิ่มประสิทธิภาพบน High-Level Synthesis ร่วมกับการลดความซับซ้อนของอัลกอริทึมบนอุปกรณ์ Xilinx Zynq-7000 ให้มีประสิทธิภาพสูงสุดภายใต้ขีดจำกัดของอุปกรณ์

1.2 การตรวจสอบเอกสาร บทความและงานวิจัยที่เกี่ยวข้อง

จากการศึกษาทบทวนวรรณกรรมในงานวิจัยนี้จะแบ่งเป็น 2 ส่วน 1. การออกแบบและใช้การประมวลผลภาพเพื่อประยุกต์ใช้ในการทำระบบตรวจจับเลนถนน 2. การวิเคราะห์และทดลองเกี่ยวกับการเพิ่มประสิทธิภาพในการประมวลผลของการประมวลผลภาพเพื่อใช้ในการทำระบบตรวจจับเลนถนน โดยจำแนกได้ดังนี้

การออกแบบ โดยใช้การประมวลผลภาพและการใช้ปัญญาประดิษฐ์เพื่อประยุกต์ใช้ในการทำระบบตรวจจับเลนถนน โดย [7] นำเสนอการออกแบบเพื่อรองรับการตรวจจับเส้นโค้งและการเลือกใช้วิธีการหาขอบภาพสำหรับระบบตรวจจับเลนถนนและมีการเปรียบเทียบผลลัพธ์ที่ได้ของการตรวจจับเลนถนนทั้งเส้นโค้งและเส้นตรงเมื่อมีการใช้วิธีการหาขอบภาพที่แตกต่างกัน [5] นำเสนอวิธีการปรับมุมมองของภาพเพื่อใช้สำหรับขั้นตอนการเตรียมภาพและพัฒนาขั้นตอนการหาขอบภาพเพื่อรองรับการตรวจจับเลนถนนบนถนนทางหลวงและชนบทซึ่งให้ผลลัพธ์ที่ดีกว่าการหาขอบภาพด้วยวิธีการทั่วไป [6] นำเสนอการพัฒนาตัวกรองอย่างง่ายขึ้นเพื่อใช้แทนวิธีการหาขอบภาพเดิม [4] นำเสนอวิธีการหาค่าความแตกต่างระหว่างค่าความชันของค่าสีของถนนและเลนถนนและใช้ค่าเทรชโวลต์ปรับตัวได้ประยุกต์ใช้กับขั้นตอนการเตรียมภาพ

การวิเคราะห์และทดลองเกี่ยวกับการเพิ่มประสิทธิภาพในการประมวลผลของการประมวลผลภาพเพื่อใช้ในการทำระบบตรวจจับเลนถนนมีด้วยกันหลากหลายวิธีไม่ว่าจะเป็นการพิจารณาปรับปรุงในส่วนของอัลกอริทึมหรือการพัฒนาอัลกอริทึมลงบนอุปกรณ์เอ็พพีจีเอเพื่อให้ได้ประสิทธิภาพการทำงานมากยิ่งขึ้นเหมาะสมกับการใช้งานมากขึ้น บทความที่ [1] นำเสนอการออกแบบ AXI อินเทอร์เฟซบนอุปกรณ์เอ็พพีจีเอเพื่อใช้ส่งและรับข้อมูล [2] นำเสนอการเปรียบเทียบประสิทธิภาพการทำงานของอินเทอร์เฟซระหว่าง ไมโครคอนโทรลเลอร์และเอ็พพีจีเอกับชนิดของข้อมูลที่แตกต่างกันบนอุปกรณ์เอ็พพีจีเอตระกูล Zynq-7000 [3] นำเสนอการวิเคราะห์ถึงปัจจัยที่ส่งผลต่อความเร็วในการส่งและรับข้อมูลบนอุปกรณ์เอ็พพีจีเอ [8] นำเสนอวิธีการปรับอัตราการสุ่มตัวอย่างในการประมวลผลบนอุปกรณ์เอ็พพีจีเอเพื่อเพิ่มความเร็วในการประมวลผล [9] นำเสนอวิธีการปรับเปลี่ยนการหาขอบภาพของ Canny และวิธีการ HT เพื่อลดความซับซ้อนของอัลกอริทึมสำหรับออกแบบระบบตรวจจับเลนถนน [11] นำเสนอวิธีการลดความซับซ้อนของวิธีการ HT ให้เหมาะสมกับใช้งานจริงเพื่อเพิ่มความเร็วในการประมวลผล [10] นำเสนอวิธีการออกแบบระบบตรวจจับเลนถนนบนอุปกรณ์เอ็พพีจีเอโดยมีการปรับใช้เทคนิคการทำไปป์ไลน์ลูปที่ระดับ HLS

จากงานวิจัยข้างต้นดังตารางที่ 1 - 1 ถึงตารางที่ 1 - 5 ยังมีประเด็นวิจัยที่สามารถพัฒนาต่อได้อีกคือการใช้เทคนิคการคลี่ลูป การไปป์ไลน์ลูปและการจัดการอินเทอร์เฟซ AXI ที่ระดับ HLS ประยุกต์กับระบบตรวจจับเลนถนน

ตารางที่ 1 - 1 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม

บทความ	วิธีการเพิ่มประสิทธิภาพระบบตรวจจับเลน		วิธีการเพิ่มประสิทธิภาพการทำงานบน Zynq-7000			ข้อดี	ข้อเสีย
	การใช้ฮาร์ดแวร์	การเพิ่มประสิทธิภาพ	การเพิ่มประสิทธิภาพ	การเพิ่มประสิทธิภาพ	การเพิ่มประสิทธิภาพ		
[1] Design of AXI Bus Based MPSoC on FPGA					✓	AXI อินเทอร์เน็ตสามารถช่วยลดปัญหาคอขวดระหว่างการสื่อสารของไมโครคอนโทรลเลอร์และเอฟพีซีโอเนื่องจากมีช่องทางในการส่งตำแหน่งและข้อมูลที่แตกต่างกัน	ถึงแม้ว่า AXI อินเทอร์เน็ตจะสามารถช่วยลดปัญหาคอขวดได้แต่ปัญหาการใช้ทรัพยากรค่อนข้างสูงหากกระหว่างการประมวลผล FIFO ที่รองรับไม่วางโปรเซสเซอร์ต้องทำการรอตัวจัดการส่งข้อมูล
[2] Comparison of On-chip Communications in Zynq-7000 All Programmable Systems-on-Chip					✓	ผู้ใช้งานสามารถเลือกใช้ชนิดของ AXI อินเทอร์เน็ตในการส่งข้อมูลระหว่างไมโครคอนโทรลเลอร์และเอฟพีซีโอได้ถูกต้องตรงกับประเภทและขนาดของชุดข้อมูลผู้ใช้งานสามารถเลือกใช้โหมดในการเข้าถึงหน่วยความจำในส่วนของ PS ให้สอดคล้องกับชนิดของอินเทอร์เน็ต AXI3	ในการเข้าถึง OCM ด้วย Cache จะสามารถเพิ่มประสิทธิภาพการทำงานได้เพียงระยะหนึ่งคือ 128 - 2048 ไบต์ หากมากกว่า 2048 ไบต์ ประสิทธิภาพการทำงานจะลดลง

ตารางที่ 1 - 2 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)

บทความ	วิธีการเพิ่มประสิทธิภาพระบบตรวจจับเลน		วิธีการเพิ่มประสิทธิภาพการทำงานบน Zynq-7000		ข้อดี	ข้อเสีย
	ใช้ GPU	เพิ่ม Cache	ปรับ Cache	ปรับ Cache		
[3] Analysis and Comparison of Attainable Hardware Acceleration in All Programmable Systems-on-Chip					ทำให้ทราบว่า การส่งข้อมูลจะช้าหรือเร็วขึ้นขึ้นอยู่กับขนาดของข้อมูล, ชนิดของช่องทางที่ใช้ส่งและการตั้งค่าอื่นๆ เช่น การใช้ cache ร่วมด้วย	ถึงแม้ว่าพอร์ต ACP จะมีความสามารถสูงสุดในการส่งข้อมูล แต่ก็มีข้อจำกัดในเรื่องของขนาดที่มากเกินไป หากข้อมูลมากประสิทธิภาพการทำงานของพอร์ต ACP จะลดลง เนื่องจากที่จัดจำกดของขนาดของหน่วยความจำ
[4] Lane Detection and Tracking For Intelligent Vehicles: A Survey	✓				วิธีการที่ใช้ในการทำระบบตรวจจับเลนถนนค่อนข้างหลากหลายและครอบคลุม สถานะการที่เปลี่ยนแปลงบนท้องถนน ทั้งกรณีถนนที่เป็นทางชันหรือสภาพแออัดในเมือง	ยังไม่มีการกล่าวถึงระยะเวลาที่ใช้ในการประมวลผลของระบบ

ตารางที่ 1 - 3 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)

บทความ	วิธีการเพิ่มประสิทธิภาพระบบตรวจจับเลน		วิธีการเพิ่มประสิทธิภาพการทำงานบน Zynq-7000		ข้อดี	ข้อเสีย
	การใช้ฮาร์ดแวร์	การใช้ซอฟต์แวร์	การเพิ่มประสิทธิภาพ	การเพิ่มประสิทธิภาพ		
[5] Near Real-Time Ego-Lane Detection in Highway and Urban Streets	✓			AXI	วิธีการตรวจจับเลนบนถนนสามารถใช้ได้กับถนนทางหลวงและถนนชนบทโดยระยะเวลาที่ใช้มีความเข้ากันได้เร็วดีไหมและมีความถูกต้องมากขึ้นจากการทำปรับมุมมองภาพ	ขั้นตอนการปรับมุมมองภาพอาจส่งผลให้การทำงานของระบบช้าลง เนื่องจากขั้นตอนที่เยอะขึ้นและซับซ้อนมากขึ้น
[6] Real-time lane detection and tracking system using simple filter and Kalman filter	✓				เป็นการปรับใช้ตัวกรองอย่างง่ายในการออกแบบระบบตรวจจับเลนถนน ซึ่งมีความซับซ้อนน้อยเหมาะสำหรับปรับใช้กับระบบที่ต้องการเร็วดีไหม	ยังไม่มีการทดสอบความแม่นยำของระบบมากนัก
[7] Model Based Design Approach for Road Lane Tracking	✓	✓			วิธีการตรวจจับเลนถนนสามารถบอกถึงกรณีเมื่อเส้นทางเป็นทางโค้ง ซึ่งสามารถปรับใช้กับระบบควบคุมต่อไปได้	ยังไม่มีการศึกษาทดลองบนอุปกรณ์ฮาร์ดแวร์อื่นๆ เช่น อุปกรณ์เอพฟิเจ็อ

ตารางที่ 1 - 4 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)

บทความ	วิธีการเพิ่มประสิทธิภาพระบบ		วิธีการเพิ่มประสิทธิภาพการทำงานบน Zynq-7000		ข้อดี	ข้อเสีย
	ตรวจสอบ	เพิ่ม	เพิ่ม	เพิ่ม		
[8] Design and development of lane detection based on FPGA	✓	✓	✓	✓	ทำการเปรียบเทียบประสิทธิภาพวิธีการที่ใช้สำหรับหาถนน และ ใช้วิธีการปรับอัตราการสุ่มตัวอย่างเพื่อเพิ่มความเร็วในการประมวลผล	ค่าพารามิเตอร์ที่ได้จากการปรับอัตราการสุ่มตัวอย่างมีการเปลี่ยนแปลง
[9] FPGA-based real-time lane detection for advanced driver assistance systems	✓				อัลกอริทึมของวิธีการหาขอบภาพของ Camy และ HT ที่ได้มีความซับซ้อนน้อยลงทำให้ระยะเวลาที่ใช้ในการประมวลผลลดลงสามารถปรับใช้กับระบบที่ต้องการเร็วได้	ยังไม่มีการทดสอบที่สภาวะแวดล้อมเปลี่ยนแปลง
[10] A hardware implementation for real-time lane detection using high-level synthesis	✓		✓		มีการปรับใช้เทคนิคการทำไปป์ไลน์รูปเพื่อเพิ่มความเร็วในการประมวลผลของระบบ	การทำไปป์ไลน์รูปเป็นเทคนิคที่ช่วยลดระยะเวลาที่ใช้ในการประมวลผล แต่ในขณะเดียวกันก็มีการใช้ทรัพยากรที่เพิ่มขึ้นด้วย

ตารางที่ 1 - 5 สรุปและเปรียบเทียบประเด็นวิจัย ข้อดีและข้อเสียในการทบทวนวรรณกรรม (ต่อ)

บทความ	วิธีการเพิ่มประสิทธิภาพระบบตรวจจับเลน		วิธีการเพิ่มประสิทธิภาพทำงานบน Zynq-7000			ข้อดี	ข้อเสีย
	เสนอ	เปรียบเทียบ	เปรียบเทียบ	เปรียบเทียบ	เปรียบเทียบ		
[11] Research on Lane Detection and Tracking Algorithm Based on Improved Hough Transform	✓	✓				เป็นการปรับใช้เทคนิคของ HT ให้เหมาะสมกับการใช้งานจริงเพื่อเพิ่มความเร็วในการประมวลผลของระบบ	ยังไม่สามารถตรวจจับเส้นทางโค้งได้
Proposed	✓	✓	✓	✓	✓	มีการปรับใช้เทคนิคการทำไปป์ไลน์ดูป การคัดลอกและการจัดการอินเทอร์เฟซ HLS กับระบบตรวจจับเลนถนน	

1.3 วัตถุประสงค์ของโครงการ

- 1.3.1 เพื่อศึกษาและออกแบบระบบตรวจจับเลนถนนที่สามารถวิเคราะห์ทางโค้งได้
- 1.3.2 เพื่อศึกษาและออกแบบเทคนิคการเพิ่มประสิทธิภาพของการสังเคราะห์วงจรที่ระดับสูง HLS

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- 1.4.1 สามารถออกแบบระบบตรวจจับเลนถนนบน Zybo z7-10 ที่ระดับ HLS
- 1.4.2 สามารถออกแบบกระบวนการเพิ่มประสิทธิภาพเพื่อปรับใช้กับอุปกรณ์ที่ระดับ HLS

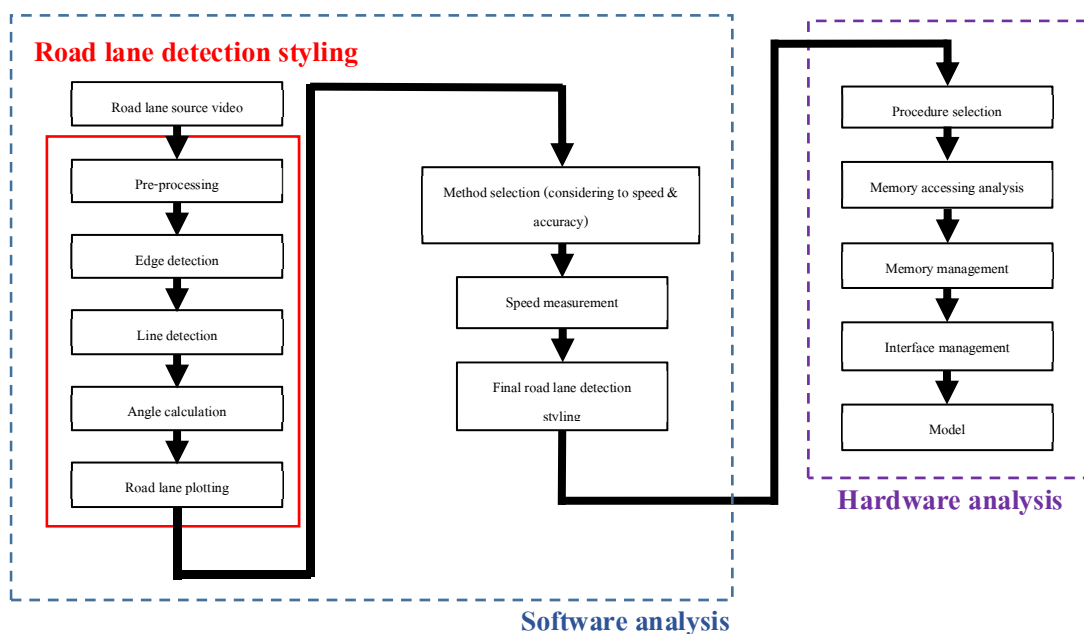
1.5 ขอบเขตการวิจัย

ขอบเขตการวิจัย ประกอบด้วย

- 1.5.1 ใช้อุปกรณ์ Zybo z7-10 ซึ่งมี Zynq-7000 ในการดำเนินการ
- 1.5.2 ใช้โปรแกรม Vivado High-Level Synthesis และ SDSoC
- 1.5.3 วิเคราะห์และออกแบบการเข้าถึงข้อมูลที่ระดับ HLS (High Level Synthesis) โดยใช้ภาษาซี
- 1.5.4 วิเคราะห์วิธีการเพิ่มประสิทธิภาพโดยใช้วิธีการที่มีรองรับบน Vivado High-Level Synthesis
- 1.5.5 ใช้ไฟล์อินพุตขนาด 1080p

1.6 วิธีการวิจัย

วิธีการวิจัยดังกล่าวประกอบ 1.1 จะถูกแบ่งออกเป็น 2 ส่วนด้วยกันคือ 1. ส่วนที่เป็น การออกแบบและวิเคราะห์การทำงานของระบบบนซอฟต์แวร์เพื่อวิเคราะห์ผลลัพธ์การทำงานเพื่อ เลือกรูปแบบที่เหมาะสมกับการทำระบบตรวจจับเลนถนน โดยมีการพิจารณาถึงความถูกต้องของ การตรวจจับเลนถนนและความเร็วในการประมวลผลเป็นหลัก 2. การวิเคราะห์การทำงานของ ระบบสำหรับดำเนินการบนอุปกรณ์เอพฟิเจตระกูล Zynq-7000 หลังจากการออกแบบระบบ ตรวจจับเลนถนนบนซอฟต์แวร์แล้วจะเป็นการวิเคราะห์การทำงานและการเข้าถึงหน่วยความจำเพื่อ ปรับใช้เทคนิคการเพิ่มประสิทธิภาพที่ระดับ HLS ได้ถูกต้องเมื่อทำการดำเนินการบนเอพฟิเจ



ภาพประกอบ 1.1 วิธีการวิจัย

1.8 สถานที่ทำวิจัย

1.8.1 ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์
วิทยาเขตหาดใหญ่

บทที่ 2

ทฤษฎีและหลักการ

บทนี้นำเสนอหลักการ และเทคนิควิธีการที่ใช้ในการวิจัย โดยงานวิจัยนี้ใช้วิธีการต่างๆของการประมวลผลภาพเพื่อแยกแยะและตรวจจับเลนถนน เพื่อนำไปประยุกต์ใช้ต่อสำหรับระบบรักษาเลนหรือระบบเตือนนอกเลนรวมไปถึงการใช้เทคนิคบน HLS ซึ่งจากการทบทวนวรรณกรรมต่างๆจะเห็นได้ว่าการออกแบบระบบตรวจจับเลนถนนมีความหลากหลายของวิธีการประมวลผลภาพที่ใช้สำหรับหาขอบภาพและการหาเส้นตรงบนภาพ รวมไปถึงเทคนิคต่างๆที่ใช้ในการเพิ่มประสิทธิภาพโดย HLS บน Zybo z7-10 ตระกูล Zynq-7000 ดังนั้นจะขออธิบายวิธีการประมวลผลภาพสำหรับการหาขอบภาพและการหาเส้นตรงบนภาพ รวมไปถึงวิธีการเพิ่มประสิทธิภาพโดย HLS ที่ใช้ในงานวิจัยนี้และวิธีการอื่นๆที่เกี่ยวข้องดังนี้

2.1 เทคนิคการหาขอบภาพ

การหาขอบภาพเป็นการหาเส้นขอบของวัตถุที่อยู่ในภาพหาได้จากการใช้หลักการหาความชันของค่าความเข้มดังสมการที่ (2-1) หากรูปนั้นไม่มีขอบภาพ ค่าความเข้มของแต่ละพิกเซลจะใกล้เคียงกัน เทคนิคการหาขอบภาพมีด้วยกันหลายวิธีการ ความซับซ้อนและความเหมาะสมของการใช้งานของแต่ละเทคนิคก็แตกต่างกันออกไป

$$\frac{df}{du}(u) = \frac{f(u+1)+f(u-1)}{2} \quad (2-1)$$

2.1.1 วิธีการหาขอบภาพของ Prewitt

เทคนิคการหาขอบภาพของ Prewitt มีโอเปอเรเตอร์ทั้งแกน x และ y ดังภาพประกอบ 2.1 โดยการคำนวณและเปรียบเทียบความชันรอบพิกเซลระหว่างฝั่งซ้ายและฝั่งขวาของจุดที่สนใจ จากนั้นจึงนำมาหาขนาดโดยใช้สมการที่ (2-4) โดยค่า D_x และค่า D_y สามารถคำนวณได้จากสมการที่ (2-2) และ (2-3) ซึ่งเกิดจากการทำคอนโวลูชันระหว่างค่าความเข้ม (intensity) กับโอเปอเรเตอร์ทั้งแกน x และแกน y

+1	0	-1
+1	0	-1
+1	0	-1

+1	+1	+1
0	0	0
-1	-1	-1

ภาพประกอบ 2.1 โอเปอเรเตอร์ของ Prewitt

$$D_x = H_x * I \quad (2-2)$$

$$D_y = H_y * I \quad (2-3)$$

$$E(u, v) = \sqrt{(D_x(u, v))^2 + (D_y(u, v))^2} \quad (2-4)$$

2.1.2 วิธีการหาขอบภาพของ Sobel

หลักการทำงานของการหาขอบภาพของ Sobel มีขั้นตอนเหมือนกันกับการหาขอบภาพของ Prewitt แต่จะมีความแตกต่างกันเฉพาะโอเปอเรเตอร์ในแนวแกน x และแกน y ดังภาพประกอบ 2.2

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

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

ภาพประกอบ 2.2 โอเปอเรเตอร์ของ Sobel

2.1.3 วิธีการหาขอบภาพของ Robert

หลักการทำงานของการหาขอบภาพของ Robert มีขั้นตอนเหมือนกันกับการหาขอบภาพของ Prewitt แต่จะมีความแตกต่างกันเฉพาะโอเปอร์เรเตอร์ในแนวแกน x และแกน y ดังภาพประกอบ 2.3

+1	0
0	-1

0	+1
-1	0

ภาพประกอบ 2.3 โอเปอร์เรเตอร์ของ Robert

2.1.4 วิธีการหาขอบภาพของ Canny

วิธีการหาขอบภาพของ Canny เป็นหนึ่งในวิธีการหาขอบรูปภาพ โดยประกอบด้วยขั้นตอนหลายขั้น โดยมีวัตถุประสงค์เพื่อเพิ่มความสามารถในการลดขนาดของขอบภาพลงและทำให้สามารถเลือกช่วงความเข้มของขอบภาพที่ต้องการได้ โดยกระบวนการจะประกอบไปด้วยทั้งหมด 5 ขั้นตอน

2.1.4.1 การกรองสัญญาณรบกวนของเกาส์เซียน

ขั้นตอนการกรองสัญญาณรบกวนของเกาส์เซียนมีวัตถุประสงค์เพื่อลดสัญญาณรบกวนออกจากภาพโดยอาศัยคุณสมบัติการกำจัดสัญญาณจากภาพต้นแบบ ด้วยเมทริกซ์ของ ตัวกรองเกาส์เซียน เนื่องจากจะทำให้ภาพละมุนขึ้น ทำให้ไม่เกิดขอบภาพที่ไม่ต้องการ

2.1.4.2 การปรับขนาดของขอบภาพและตำแหน่งทิศทางการวางตัวของขอบภาพ

ขั้นตอนการปรับขนาดของขอบภาพและตำแหน่งทิศทางการวางตัวของขอบภาพของรูปภาพที่ผ่านการกรองสัญญาณรบกวนของเกาส์เซียนมาแล้ว โดยการใช้กระบวนการแบบการหาขอบภาพของ Prewitt หรือการหาขอบภาพของ Sobel ก็ได้

2.1.4.3 การปรับช่วงค่าของขอบภาพ

ขั้นตอนการทำปรับช่วงค่าของขอบภาพมีวัตถุประสงค์เพื่อนำไปประยุกต์ใช้ในกระบวนการการลบค่าที่ไม่ต้องการในแต่ละพิกเซลซึ่งจำเป็นต้องเปลี่ยนค่าให้อยู่ในช่วงที่สามารถระบุพิกัดเป็นตำแหน่งของพิกเซลโดยรอบได้

2.1.4.4 การลบค่าที่ไม่ต้องการในแต่ละพิกเซล

เมื่อเราทำการแปลงค่าตำแหน่งทิศทางการวางตัวของขอบภาพให้เป็นค่าที่สามารถระบุเป็นตำแหน่งพิกเซลโดยรอบได้แล้วนั้น ในกระบวนการนี้จะมีการใช้ค่าตำแหน่งทิศทางการวางตัวของขอบภาพเพื่อเลือกคู่พิกเซลโดยรอบมาทำการเปรียบเทียบค่าขนาดของขอบภาพของพิกเซลนั้นๆ หากค่าของพิกเซลกลางนั้นมีค่ามากกว่าพิกเซลข้างเคียงที่เลือกมา ให้เก็บค่าพิกเซลนั้นๆไว้

2.1.4.5 ฮิสเทอรีซิส

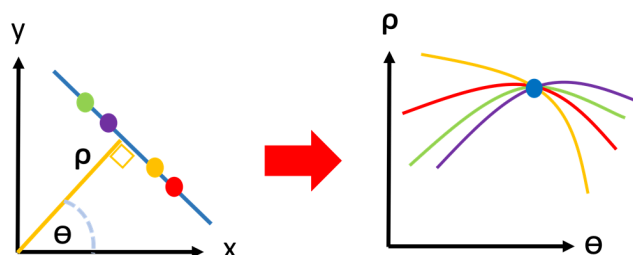
ฮิสเทอรีซิสเป็นขั้นตอนสุดท้ายของการหาขอบภาพของ Canny ภาพที่ได้จากขั้นตอนก่อนหน้ายังมีความเป็นไปได้ที่จะยังหลงเหลือผลกระทบจากสัญญาณรบกวนที่ไม่เกี่ยวข้องอยู่จำนวนหนึ่ง เพื่อตัดสิ่งรบกวนเหล่านี้ออกไปจึงมีการนำฮิสเทอรีซิสเข้ามาช่วยโดยทำให้สามารถเลือกช่วงค่า ขนาดของขอบภาพที่ต้องการแสดงได้และค่าที่ต่ำกว่าในขอบเขตที่ระบุให้กำจัดค่าของพิกเซลนั้นทิ้ง ส่วนค่าที่มากกว่าให้เป็นขอบภาพความเข้มสูงและส่วนที่อยู่ในช่วงให้เป็นขอบความเข้มจางซึ่งจะนำมาพิจารณาต่อไป

2.2 เทคนิคการหาเส้นตรงโดยใช้ Hough Transform (HT)

วิธีการ HT เป็นหนึ่งในวิธีการแยกคุณสมบัติเพื่อใช้สำหรับหาทั้งเส้นตรง วงกลม หรือวงรี เป็นต้น โดยวิธีการนี้จะเป็นการเลือกนำเอกลักษณ์ของวัตถุมาพิจารณา ในกรณีของการใช้

$$y = mx + c \quad (2-5)$$

$$\rho = x \cos \theta + y \sin \theta \quad (2-6)$$



ภาพประกอบ 2.4 Hough Transform (HT)

HT ในการวิเคราะห์เส้นตรงจะมีการวิเคราะห์จาก สมการที่ (2-5) และสมการที่ (2-6) โดยสมการที่ (2-5) คือสมการเส้นตรงบนระนาบ $x-y$ และสมการที่ (2-6) เป็นการแปลงค่าของตำแหน่งบนระนาบ $x-y$ ให้อยู่บนระนาบ $\rho-\theta$ ดังแสดงในภาพประกอบ 2.4 โดยจุดทุกตำแหน่งบนเส้นตรงบนระนาบ $x-y$ จะถูกแปลงเป็นเส้นโค้งในระนาบ $\rho-\theta$ และหากจุดทุกจุดบนระนาบ $x-y$ อยู่บนเส้นตรงเส้นเดียวกันเมื่อถูกแปลงเป็นเส้นโค้งบนระนาบ $\rho-\theta$ จะเกิดจุดตัดที่ตำแหน่งเดียวกัน

2.3 เทคนิคการเพิ่มประสิทธิภาพ

2.3.1 เทคนิคการคลี่ลูป (Loop unrolling)

เทคนิคการคลี่ลูปเป็นวิธีการเพิ่มประสิทธิภาพการทำงานโดยการทำงานบางรอบหรือทุกรอบสามารถทำงานพร้อมๆ กัน ได้ดังตัวอย่างภาพประกอบ 2.5

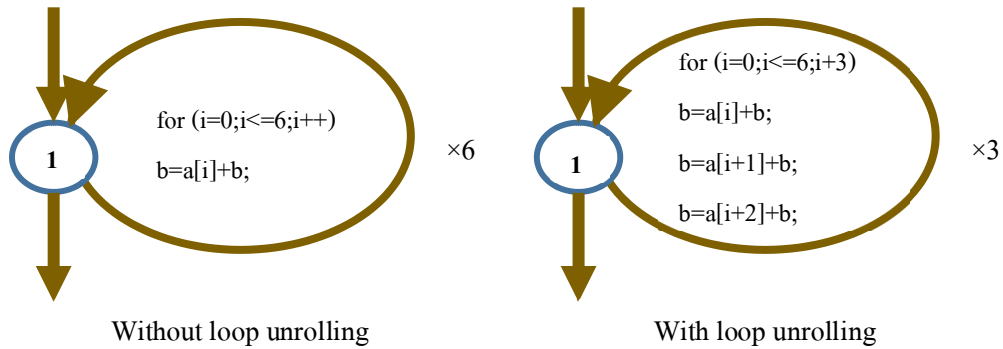
จากตัวอย่างเทคนิคการคลี่ลูป ลักษณะการทำงานแบบลำดับของลูปเดิมจำเป็นต้องมีการทำงานทั้งหมด 6 รอบด้วยกัน แต่หากมีการคลี่ลูปในแต่ละรอบของการทำงานจะมีการคำนวณมากขึ้นทำให้จำนวนรอบของการทำงานทั้งหมดลดลงจาก 6 รอบการทำงานเหลือเพียง 3 รอบการทำงาน

2.3.1 เทคนิคการไปป์ไลน์ลูป (Loop pipelining)

เทคนิคการทำไปป์ไลน์ลูปเป็นวิธีการเพิ่มประสิทธิภาพการทำงานอีกวิธีการหนึ่งโดยวิธีการนี้จะเป็นการลดจำนวนช่วงเวลาเริ่มต้นการทำงานครั้งถัดไป (Initiation Interval: II) ของการทำงานของฟังก์ชันหรือลูป ดังตัวอย่างในภาพประกอบ 2. 6 ภายในการทำงาน ของลูปซึ่งประกอบไปด้วย 3 ขั้นตอนด้วยกัน (Rd, Cmp, Wr) โดยการทำงานแบบเป็นลำดับทั่วไป ต้องใช้จำนวนระยะเวลาถึง 6 รอบนาฬิกา สำหรับการทำงาน 2 รอบของลูปเนื่องจากการทำงานใน รอบที่ 2 จะสามารถเริ่มต้นทำงานได้ก็ต่อเมื่อการทำงานในรอบที่ 1 เสร็จสมบูรณ์ แต่หากมีการทำ ไปป์ไลน์ลูปการทำงานในรอบที่ 2 สามารถเริ่มต้นทำงานได้ก่อนที่การทำงานในรอบที่ 1 จะเสร็จ สมบูรณ์ทำให้จำนวนระยะเวลาที่ใช้ลดลงจาก 6 รอบนาฬิกา เหลือเพียง 4 รอบนาฬิกาหรือ II ถูก ลดลงจาก 3 รอบนาฬิกาเหลือเพียง 1 รอบนาฬิกา

2.3.2 การแบ่งอาร์เรย์ (Array partitioning)

เทคนิคการแบ่งอาร์เรย์เป็นวิธีการเพิ่มประสิทธิภาพการทำงานทั้งในด้านของการใช้ทรัพยากรที่มีอย่างเหมาะสมและการเพิ่มความเร็วในการประมวลผล โดยเทคนิคนี้มี



ภาพประกอบ 2.5 ตัวอย่างเทคนิคการคลี่ลูป

หลักการคือการแบ่งอาร์เรย์ขนาดใหญ่ 1 อาร์เรย์ ออกเป็นอาร์เรย์ย่อยๆ หลายๆ กลุ่ม ซึ่งวิธีการของการแบ่งอาร์เรย์นี้ถูกแบ่งออกเป็น 3 วิธีการย่อยดังภาพประกอบ 2.7 คือ 1. การแบ่งอาร์เรย์แบบบล็อก (Block) โดยอาร์เรย์ขนาดใหญ่ 1 กลุ่ม จะถูกแบ่งออกเป็นอาร์เรย์กลุ่มย่อยๆ ซึ่งตำแหน่งของอาร์เรย์แต่ละตำแหน่งจะมีการจัดเรียงต่อกัน โดยที่แต่ละกลุ่มจะมีจำนวนอีลิเมนต์ที่เท่ากัน 2. การแบ่งอาร์เรย์แบบวงกลม (Cyclic) โดยลักษณะการแบ่งกลุ่มของอาร์เรย์จะเหมือนกันกับวิธีการแบ่งอาร์เรย์แบบบล็อกแต่จะแตกต่างกันตรงตำแหน่งของอาร์เรย์ในแต่ละกลุ่ม ลักษณะของตำแหน่งของอาร์เรย์ในวิธีการนี้จะมีการเรียงตำแหน่งเรียงสลับกันระหว่างกลุ่ม ไปจนครบทุกตำแหน่งของอาร์เรย์ และ 3. การแบ่งอาร์เรย์แบบสมบูรณ์ (Complete) วิธีการสุดท้ายนี้จะมีความแตกต่างกับวิธีการทั้ง 2 ก่อนหน้านี้คือ อาร์เรย์ขนาดใหญ่ 1 กลุ่ม จะถูกแต่เป็นกลุ่มย่อยๆ ตามจำนวนตำแหน่งของอาร์เรย์ที่มีทั้งหมด ดังนั้นกลุ่มย่อยๆ ของอาร์เรย์ทุกกลุ่มจะมีลักษณะเป็นอิสระต่อกัน และตัวแปรสำคัญสำหรับการแบ่งอาร์เรย์อีก 2 ตัวแปรคือ Factor และ Dimension โดยค่า Factor จะเป็นตัวบ่งบอกถึงจำนวนกลุ่มย่อยของอาร์เรย์ที่ต้องการทำแบ่งซึ่งจะมีผลกับการแบ่งอาร์เรย์แบบบล็อกและวงกลม ส่วนตัวแปร Dimension จะเป็นตัวกำหนดมิติของอาร์เรย์ที่ต้องการทำการแบ่งซึ่งจะมีผลก็คือเมื่อผู้ใช้งานมีการประกาศตัวแปรอาร์เรย์มากกว่า 1 มิติ

2.3.3 อินเทอร์เฟซ AXI (AXI interface)

โปรโตคอล AXI (Advanced eXtensible Interface) เป็นส่วนหนึ่งของโปรโตคอล ARM AMBA ของไมโครคอนโทรลเลอร์

อินเทอร์เฟซ AXI มีการพัฒนาด้วยกัน 2 รุ่นคือ AXI และ AXI4 โดย AXI4 แบ่งออกเป็น 3 ชนิดด้วยกันคือ 1. AXI4 2. AXI4-Lite และ 3. AXI4-Stream โดยหลักการทำงานของ AXI คือเป็น อินเทอร์เฟซสำหรับแลกเปลี่ยนข้อมูลกันระหว่าง AXI master และ AXI slave

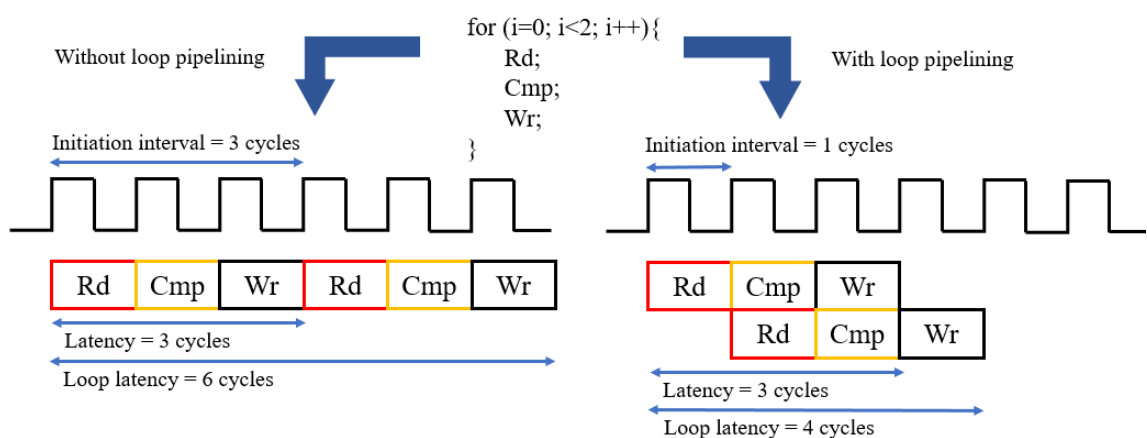
สถาปัตยกรรมของทั้ง AXI4 และ AXI4-Lite จะประกอบไปด้วย 5 ช่องสัญญาณที่แตกต่างกันคือ read address channel, read data channel, write address channel, write data channel และ write response channel โดยข้อมูลสามารถรับและส่งระหว่าง master และ slave ได้พร้อมๆกัน แต่ AXI4 จะมีรูปแบบการส่งข้อมูลแบบ Single Address Multiple Data (SAMD) ดังภาพประกอบ 2.9 ส่วน AXI4-Lite จะมีรูปแบบการส่งข้อมูลแบบ Single Address Single Data (SASD) ดังภาพประกอบ 2.8

สถาปัตยกรรมสำหรับ โปรโตคอล AXI4-Stream จะแตกต่างออกไปจาก AXI4 และ AXI4-Lite คือมีเฉพาะช่องสัญญาณสำหรับส่งหรือรับข้อมูลโดยไม่มีช่องสัญญาณสำหรับอ่านหรือเขียนตำแหน่งที่อยู่ของข้อมูลนั้นๆ ซึ่งการส่งข้อมูลในลักษณะนี้จะมีประสิทธิภาพมากสำหรับการอ่านหรือเขียนข้อมูลเป็นลำดับไปเรื่อยๆดังภาพประกอบ 2.10

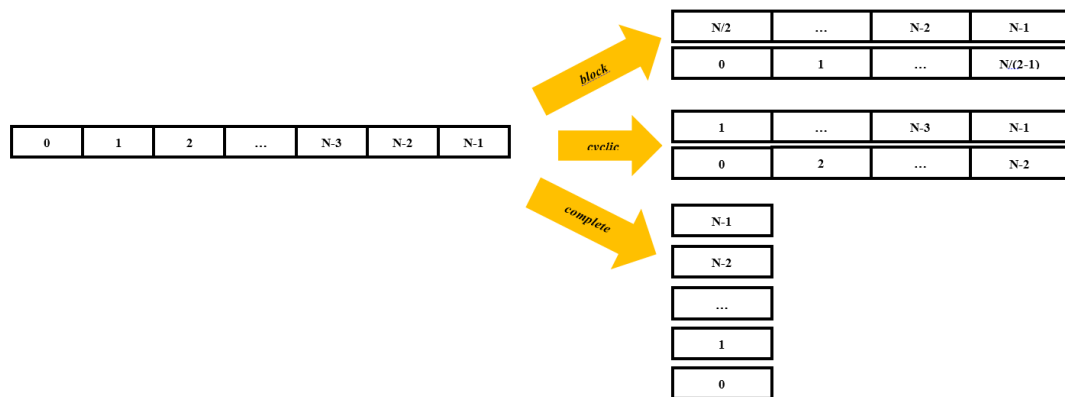
2.3.4 การจัดการอินเทอร์เฟซ HLS (HLS interface management)

Xilinx Vivado HLS เป็นเครื่องมือที่รองรับการออกแบบวงจรฮาร์ดแวร์บนบอร์ด FPGA ด้วยภาษาที่ระดับ HLS นอกจากผู้ออกแบบจะสามารถออกแบบการทำงานของ IP ได้แล้วยังสามารถกำหนดอินเทอร์เฟซสำหรับรับหรือส่งข้อมูลของ IP นั้นๆ กับ IP อื่นๆได้อีกด้วย

โดยอินเทอร์เฟซที่ใช้จะอยู่บนพื้นฐานของ AXI4, AXI4-Lite และ AXI4-Stream โดยในการสังเคราะห์บล็อก IP ของ Xilinx Vivado HLS จะถูกสร้างมาพร้อมกับอินเทอร์เฟซบน IP นั้นๆ สำหรับรองรับการส่งและรับข้อมูลต่างๆ โดยโปรโตคอล I/O มาตรฐาน

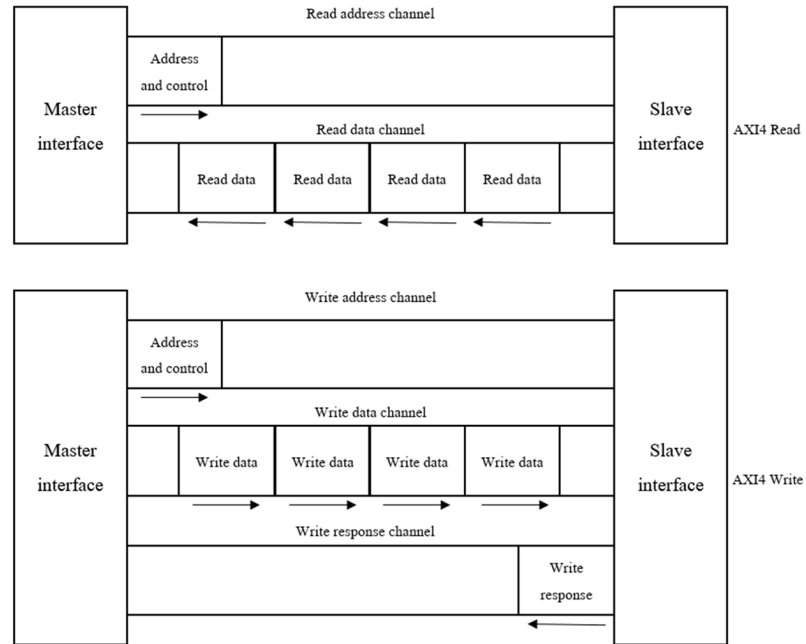


ภาพประกอบ 2. 6 ตัวอย่างการทำไปป์ไลน์ลูป

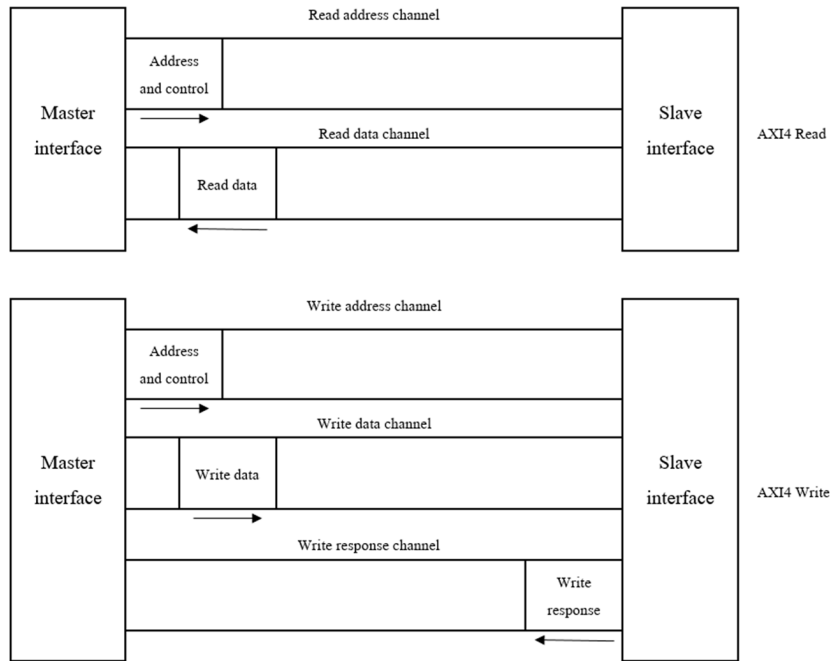


ภาพประกอบ 2. 7 ประเภทของการแบ่งอาร์เรย์

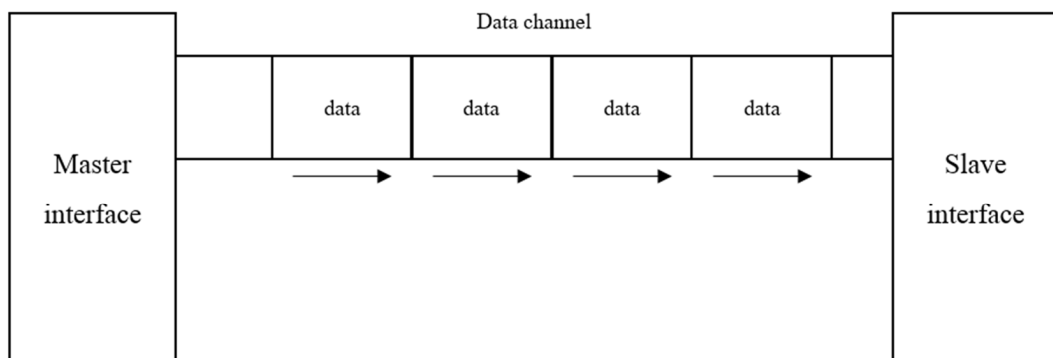
ที่รองรับบน Xilinx Vivado HLS มีด้วยกันหลากหลายโปรโตคอล ซึ่งผู้ออกแบบสามารถกำหนดการใช้งานเองหรือให้ระบบเป็นตัวตัดสินใจก็ได้เช่นกัน โดย Vivado จะทำการสร้างพอร์ต 3 ชนิด บนการออกแบบ RTL คือ 1. Clock และ Reset 2. โปรโตคอลของอินเทอร์เฟซ Block-level โดยสัญญาณจากอินเทอร์เฟซนี้จะเป็นตัวบอกการเริ่มทำงานของบล็อก IP นั้นๆ รวมไปถึงเป็นตัวบ่งบอกสถานะการทำงานทั้งสถานะเมื่อพร้อมจะทำงานรอบถัดไปหรือพร้อมรับข้อมูลชุดถัดไป และสถานะเมื่อมีการทำงานเสร็จสมบูรณ์และ 3. โปรโตคอลอินเทอร์เฟซ Port-level เมื่อโปรโตคอลของ Block-level มีการทำงานและบล็อกเริ่มมีการทำงาน โปรโตคอล Port-level จะทำหน้าที่ในการจัดเรียงข้อมูลสำหรับเข้าและออกจากบล็อก IP ที่ถูกสร้างขึ้นจาก Vivado โดยลักษณะของอินเทอร์เฟซ Block-level และอินเทอร์เฟซ Port-level แต่ละชนิดจะมีคุณสมบัติและความเหมาะสมกับชนิดของข้อมูลขาเข้าและขาออกที่แตกต่างกันออกไปดังแสดงในตารางที่ 2 - 1



ภาพประกอบ 2.8 รูปแบบการส่งข้อมูลของ AXI4



ภาพประกอบ 2.9 รูปแบบการส่งข้อมูลแบบ AXI4-Lite



ภาพประกอบ 2.10 รูปแบบการส่งข้อมูลแบบ AXI-Stream

ตารางที่ 2 - 1 อินเทอร์เฟซ HLS สำหรับแต่ละชนิดของข้อมูล

Argument Type	Scalar		Array			Pointer or Reference			HLS::Stream
	Input	Return	I	I/O	O	I	I/O	O	
ap_ctrl_none									
ap_ctrl_hs									
ap_ctrl_chain									
axis									
s_axilite									
m_axi									
ap_none									
ap_stable									
ap_ack									
ap_vld									
ap_ovld									
ap_hs									
ap_memory									
bram									
ap_fifo									
ap_bus									



Supported D = Default Interface



Not Supported

บทที่ 3

กระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูง (Optimization Process Using High Level Synthesis)

ในหัวข้อนี้จะนำเสนอกระบวนการเพิ่มประสิทธิภาพวงจรดิจิทัลด้วยเทคนิคการสังเคราะห์ที่ระดับสูงประยุกต์ใช้ในการปรับปรุงระบบตรวจจับเลนถนน (Road lane detection) ที่สามารถลดโอกาสการเกิดข้อผิดพลาดในการตรวจจับเส้นบนถนนในกรณีถนนเป็นทางโค้ง และวิธีการลดระยะเวลาที่ใช้ในการประมวลผลการตรวจจับเลนถนน โดยการเพิ่มเทคนิคต่างๆ ไม่ว่าจะเป็นการปรับขนาดอาร์เรย์ การคลี่ลูป การไปป์ไลน์ลูป การแบ่งอาร์เรย์และการจัดการอินเทอร์เฟส HLS ภายใต้อุปกรณ์ที่จำกัดบนอุปกรณ์ฮาร์ดแวร์

3.1 การออกแบบระบบตรวจจับเลนถนน

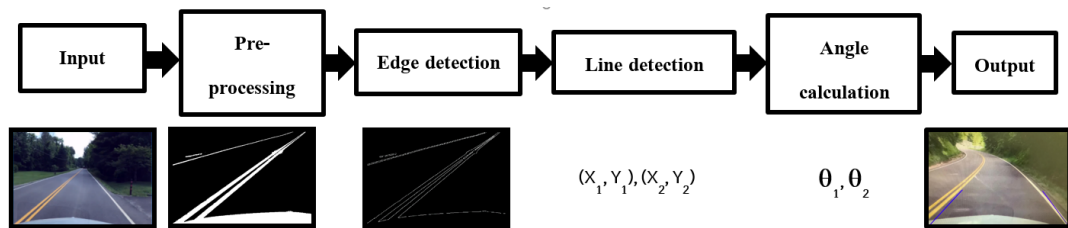
วิธีการระบบตรวจจับเลนถนนที่นำเสนอในงานวิทยานิพนธ์นี้แบ่งขั้นตอนการทำงานออกเป็น 4 ขั้นตอนหลัก ๆ คือ 1. การเตรียมภาพ (Pre-processing) 2. การหาขอบภาพ (Edge detection) 3. การหาเส้นตรงบนภาพ (Line detection) และ 4. การคำนวณมุมของเส้นตรง (Angle calculation) ดังภาพประกอบ 3.1

3.1.1 การเตรียมภาพ (Pre-processing)

สำหรับขั้นตอนการเตรียมภาพจะถูกแบ่งออกเป็นวิธีการย่อย ๆ ออกเป็น 4 ขั้นตอนคือ 1. การตัดภาพ เพื่อจำกัดขอบเขตของภาพเหลือเฉพาะส่วนที่สนใจและยังทำให้ลดการประมวลผล 2. การแบ่งภาพออกเป็นด้านซ้ายและด้านขวา ระหว่างเส้นด้านข้างถนนและเส้นกลางถนน ทำให้มีขอบเขตการทำงานที่ชัดเจน ไม่นำเส้นอื่น ๆ นอกจากนี้มาประมวลผล 3. การแปลงภาพสีเป็นภาพขาวเทา เพื่อลดการประมวลผลภาพสีและ 4. การแปลงภาพขาวเทาเป็นภาพขาวดำ เพื่อให้สามารถนำไปประมวลผลตรวจจับเส้นได้ง่ายขึ้นดังภาพประกอบ 3.2

3.1.2 การหาขอบภาพ (Edge detection)

ในขั้นตอนของการหาขอบภาพเป็นขั้นตอนหนึ่งการในการตรวจจับเส้น วิธีการหาขอบภาพมีหลายวิธีเช่น การหาขอบภาพ Robert การหาขอบภาพของ Prewitt การหาขอบภาพของ Sobel และการหาขอบภาพของ Canny ในวิทยานิพนธ์จะทำการวิเคราะห์วิธีการหา



ภาพประกอบ 3.1 การออกแบบระบบตรวจจับเลนถนน

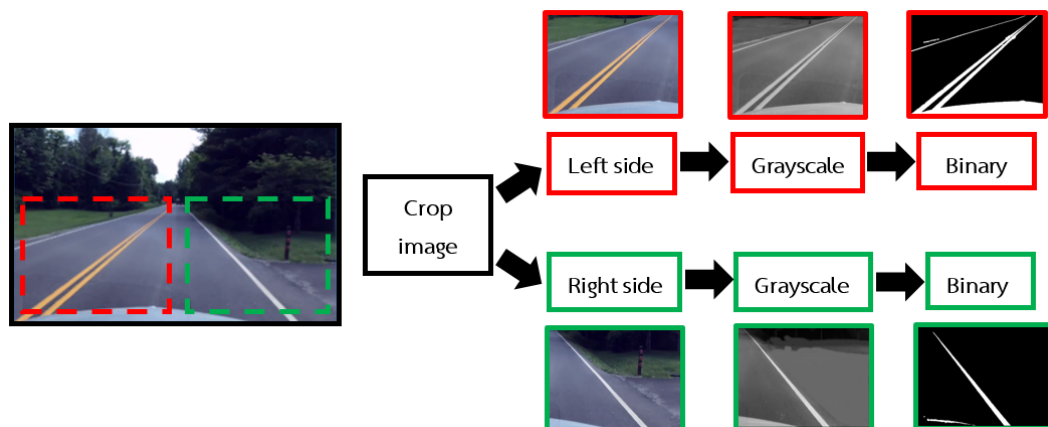
ขอบภาพที่เหมาะสม ในประเด็น ความเร็วในการคำนวณ ทรัพยากรที่ใช้ และความแม่นยำของภาพ เพื่อใช้ในการตัดสินใจเลือกวิธีการที่เหมาะสมที่สุด

3.1.3 การหาเส้นตรง (Line detection)

ขั้นตอนการหาเส้นบนภาพจะมีการใช้วิธีการของ Hough Transform (HT) ซึ่งผลที่ได้จากการทำ HT เป็นค่าตำแหน่งเริ่มต้นของเส้นตรง (X_1, Y_1) และตำแหน่งสุดท้ายของเส้น (X_2, Y_2) โดยทั้งสองตำแหน่งนี้สามารถใช้สำหรับคำนวณหามุมของเส้นตรงได้ในขั้นตอนถัดไป

3.1.4 การคำนวณมุมของเส้นตรง (Angle calculation)

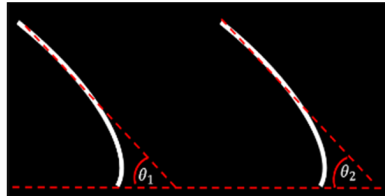
ในขั้นตอนนี้เส้นตรงที่ได้จากการขั้นตอนที่ 3.1.3. ทั้งเส้นด้านซ้ายและเส้นด้านขวาจะถูกนำมาคำนวณหาค่ามุม (θ_1, θ_2) เพื่อจำแนกรูปแบบของเส้นเป็น ทางโค้งและทางตรง โดย θ_1 บ่งบอกถึงมุมที่ได้จากการคำนวณของเส้นตรงด้านซ้ายของถนนและ θ_2 บ่งบอกถึงมุมที่ได้จากการคำนวณจากเส้นตรงด้านขวาของถนนซึ่งสามารถแบ่งมุมที่ได้จากการคำนวณออกได้เป็น 3 กรณีด้วยกันคือ



ภาพประกอบ 3.2 ขั้นตอนการเตรียมภาพ

3.1.4.1 $\theta_1 < 90^\circ$ และ $\theta_2 < 90^\circ$

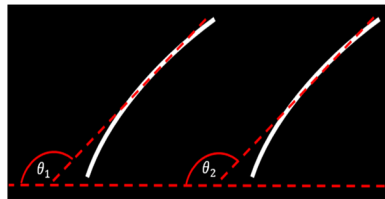
ในกรณีที่ θ_1 มีค่าน้อยกว่า 90° และ θ_2 มีค่าน้อยกว่า 90° เส้นทางบนถนนนี้จะถูกเข้าใจว่าเป็นเส้นทางโค้งซ้ายดังภาพประกอบ 3.3



ภาพประกอบ 3.3 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางโค้งซ้าย

3.1.4.2 $\theta_1 > 90^\circ$ และ $\theta_2 > 90^\circ$

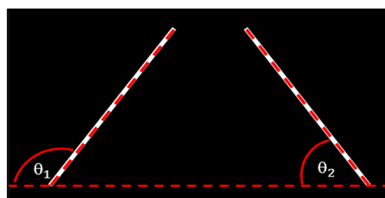
ในกรณีที่ θ_1 มีค่ามากกว่า 90° และ θ_2 มีค่ามากกว่า 90° เส้นทางบนถนนนี้จะถูกเข้าใจว่าเป็นเส้นทางโค้งขวาดังภาพประกอบ 3.4



ภาพประกอบ 3.4 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางโค้งขวา

3.1.4.3 $\theta_1 > 90^\circ$ และ $\theta_2 < 90^\circ$

ในกรณีที่ θ_1 มีค่ามากกว่า 90° และ θ_2 มีค่าน้อยกว่า 90° เส้นทางบนถนนนี้จะถูกเข้าใจว่าเป็นเส้นทางตรงดังภาพประกอบ 3.5

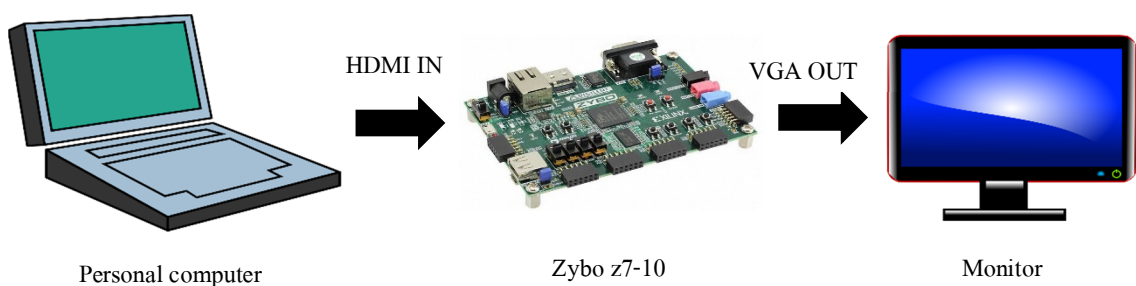


ภาพประกอบ 3.5 การวัดมุม θ_1 และ θ_2 สำหรับกรณีทางตรง

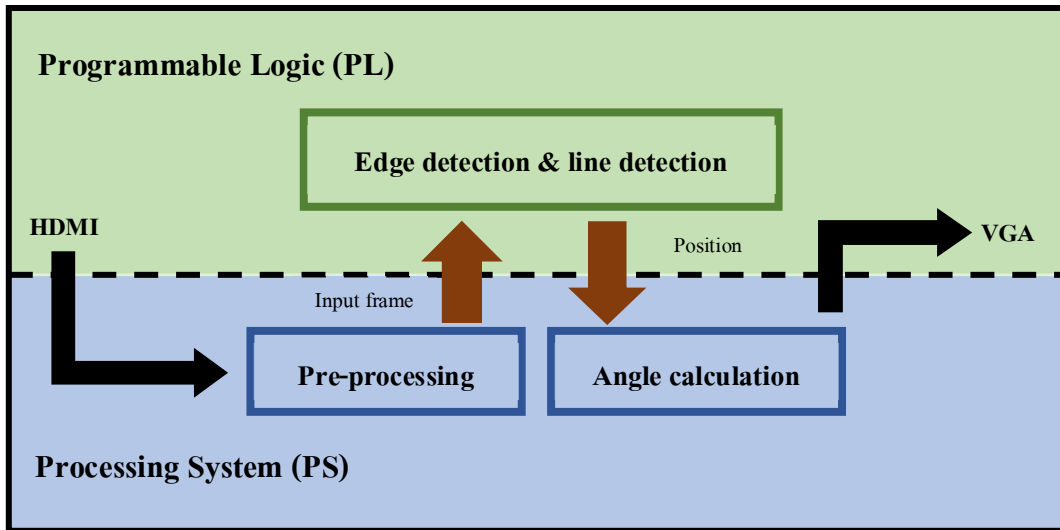
3.2 การออกแบบระบบบนอุปกรณ์ฮาร์ดแวร์

เนื่องจากการทำงานของระบบบนโปรเซสเซอร์เพียงอย่างเดียวพบว่าระยะเวลาที่ใช้ในการประมวลผลค่อนข้างนาน การออกแบบการทำงานจึงมีการเปลี่ยนแปลงจากการประมวลผลบนโปรเซสเซอร์อย่างเดียวเป็นบอร์ดประมวลผล Zybo z7-10 ตระกูล Zynq-7000 ซึ่งเป็นอุปกรณ์ที่รวบรวมการทำงานระหว่างเทคโนโลยีของโปรเซสเซอร์และเอพฟิจีเอไว้ด้วยกัน โดยการทำงานของระบบที่ใช้ทดสอบจะมีการรับภาพหรือวิดีโอจาก PC ผ่าน HDMI และแสดงผลผ่านจอแสดงผลดังกล่าวประกอบ 3.6 ซึ่งกระบวนการของระบบตรวจจับเลนถนนจะถูกแบ่งการทำงานออกเป็น 2 ส่วนด้วยกันคือ การทำงานบน PL และการทำงานบน PS โดยขั้นตอนการทำงานที่ใช้ระยะเวลาในการประมวลผลนานจะถูกดำเนินการอยู่บน PL (ขั้นตอนการหาขอบภาพและขั้นตอนการหาเส้นตรง) และขั้นตอนที่ใช้ระยะเวลาในการประมวลผลเร็วกว่าขั้นตอนอื่นๆจะถูกดำเนินการอยู่บน PS (ขั้นตอนการเตรียมภาพและขั้นตอนการคำนวณมุมของเส้นตรง) ดังแสดงในภาพประกอบ 3.7

จากภาพประกอบ 3.8 เป็นการออกแบบบนอุปกรณ์จริง สัญญาณภาพขาเข้าของระบบจะผ่าน HDMI ซึ่งเป็นภาพ RGB จะถูกจัดลำดับอีกครั้งโดย IP concat โดยสัญญาณบัส 3 สัญญาณ (RGB) จะถูกรวมเป็นสัญญาณบัสเพียง 1 สัญญาณส่งไปยัง PS ผ่านอินเทอร์เฟซ AXI ทั้ง AXI4-stream และ axi vdma ที่มีบน PL และ AXI HP บน PS เพื่อทำการเตรียมภาพหลังจากนั้นการส่งข้อมูลจะถูกส่งต่อจาก PS ไป PL (HLS IP) เพื่อทำการหาขอบภาพและหาเส้นตรงในรูปแบบของกระแสผ่าน AXI GP และ AXI-Stream และมีการส่งกลับอีกครั้งเพื่อทำการคำนวณมุม โดยมี AXI ACP รองรับ



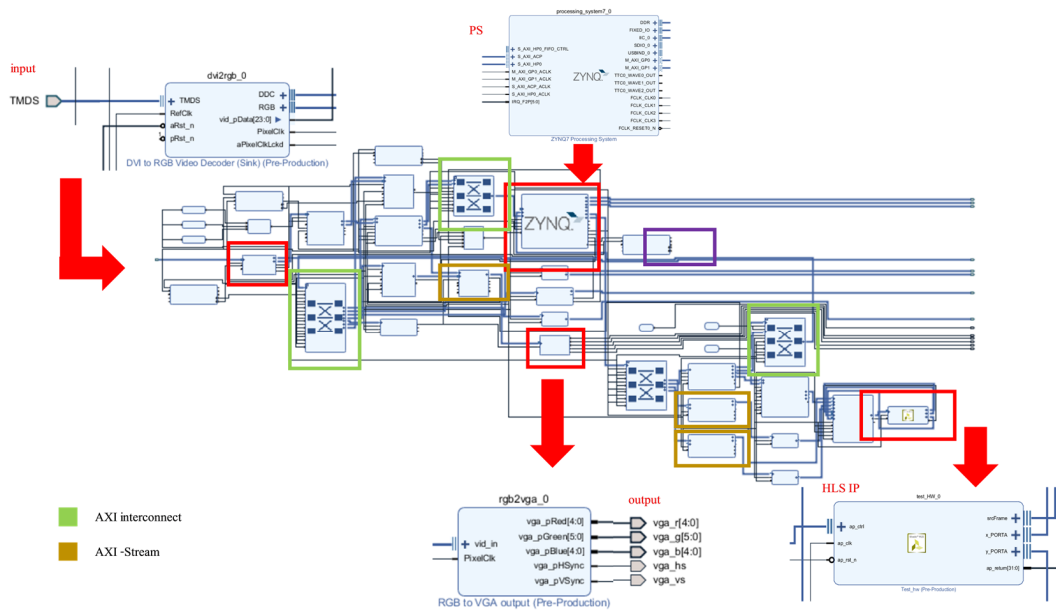
ภาพประกอบ 3.6 ระบบที่ใช้สำหรับทดสอบ



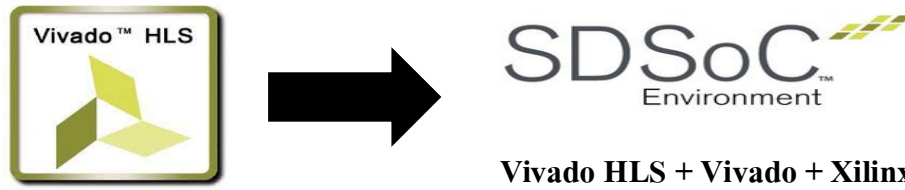
ภาพประกอบ 3.7 การออกแบบฮาร์ดแวร์ด้วยบอร์ด Zybo z7-10 ตระกูล Zynq-7000

3.3 เครื่องมือที่ใช้สำหรับออกแบบ

ในการออกแบบการทำระบบตรวจจับเลนถนนโดยขั้นตอนการหาขอบภาพและหาเส้นตรงบนภาพจะถูกดำเนินการบนฮาร์ดแวร์เพื่อลดระยะเวลาในการประมวลผล ขั้นตอนการออกแบบการหาขอบภาพและหาเส้นตรงบนภาพจะออกแบบบน Xilinx Vivado High-Level Synthesis เพื่อทดสอบการทำงานและความถูกต้องของอัลกอริทึมรวมไปถึงการทำเทคนิคการเพิ่ม



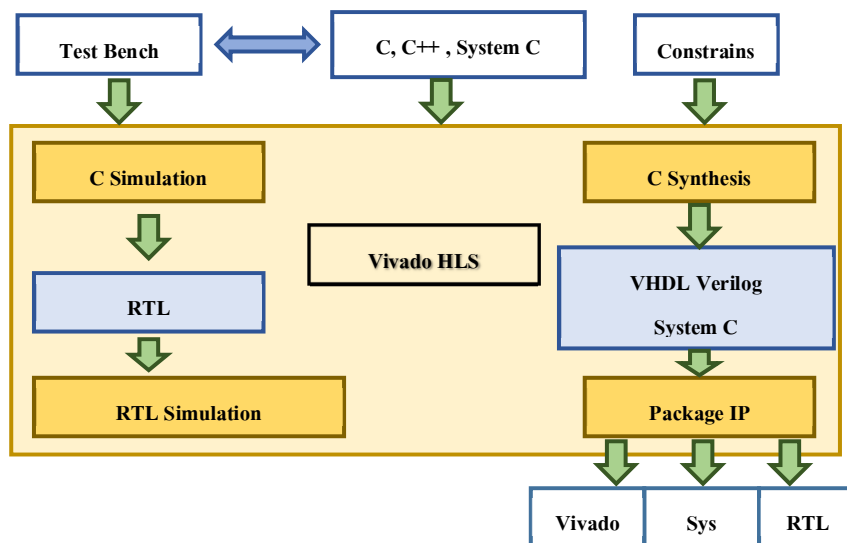
ภาพประกอบ 3.8 การออกแบบแพลตฟอร์มด้วยบอร์ด Zybo z7-10 ตระกูล Zynq-7000



ภาพประกอบ 3.9 ลำดับการใช้เครื่องมือสำหรับออกแบบระบบตรวจจับเลนถนน

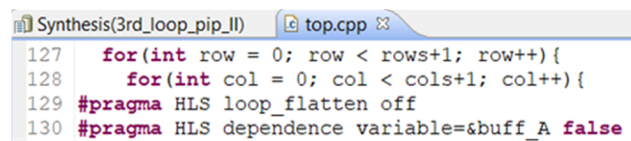
ประสิทธิภาพต่างๆ หลังจากการทดสอบความถูกต้องบน Xilinx Vivado HLS แล้ว อัลกอริทึมที่ได้ จะถูกนำไปรวบรวมบน SDSoC ซึ่งเป็น โปรแกรมที่มีการสร้าง แพลตฟอร์มสำหรับทดสอบ อัลกอริทึมกับอุปกรณ์จริง ลำดับการใช้เครื่องมือสำหรับออกแบบระบบตรวจจับเลนถนนแสดงดัง ภาพประกอบ 3.9 และรายละเอียดของแต่ละเครื่องมือเป็นดังต่อไปนี้

Vivado HLS เป็นเครื่องมือที่ใช้ในการสร้าง IP โดยผู้ใช้งานสามารถ โปรแกรม หรือออกแบบ IP ของตัวเองได้บนภาษา C, C++ หรือ System C ดังลำดับการออกแบบ (Design flow) ของ Xilinx Vivado HLS ภาพประกอบ 3.10 สามารถทดสอบอัลกอริทึมได้โดยการสร้าง ไฟล์ test bench เพื่อทำ C Simulation และ RTL Simulation ตามลำดับ ในขณะที่ผู้ใช้งานออกแบบ IP ด้วย C, C++ หรือ System C สามารถเพิ่มเติมเทคนิคการเพิ่มประสิทธิภาพต่างๆ ได้โดยใช้คำสั่ง HLS pragmas ลงในรหัสต้นฉบับ (source code) ดังภาพประกอบ 3.11 หรือหน้าต่างคำสั่ง (directive) ดัง ภาพประกอบ 3.12 ไม่ว่าจะเป็นการทำการคลี่ลูป การ ไปป์ไลน์ลูป การแบ่งอาร์เรย์หรือแม้กระทั่ง การกำหนดอินเทอร์เฟส HLS นอกจากนี้เรายังสามารถทำการวิเคราะห์ลำดับการทำงานหรือลำดับ การเข้าถึงตัวแปรต่างๆ ได้จากการสังเคราะห์ซี (C Synthesis) ดังภาพประกอบ 3.13 ผลของลำดับ



ภาพประกอบ 3.10 Xilinx Vivado HLS design flow

การทำงานต่างๆสามารถวิเคราะห์ได้จากแถบวิเคราะห์ (Analysis) รวมไปถึงระยะเวลาที่ใช้ในการประมวลผล (Performance profile) และจำนวนทรัพยากร (Resource profile) อย่างละเอียดที่ใช้สำหรับ IP ที่ได้ทำการออกแบบดังภาพประกอบ 3.14 ซึ่งภาพขยายดังภาพประกอบ 3.15 และยังสามารถ ทำการ C Simulation เพื่อทดสอบความถูกต้องของอัลกอริทึม ได้

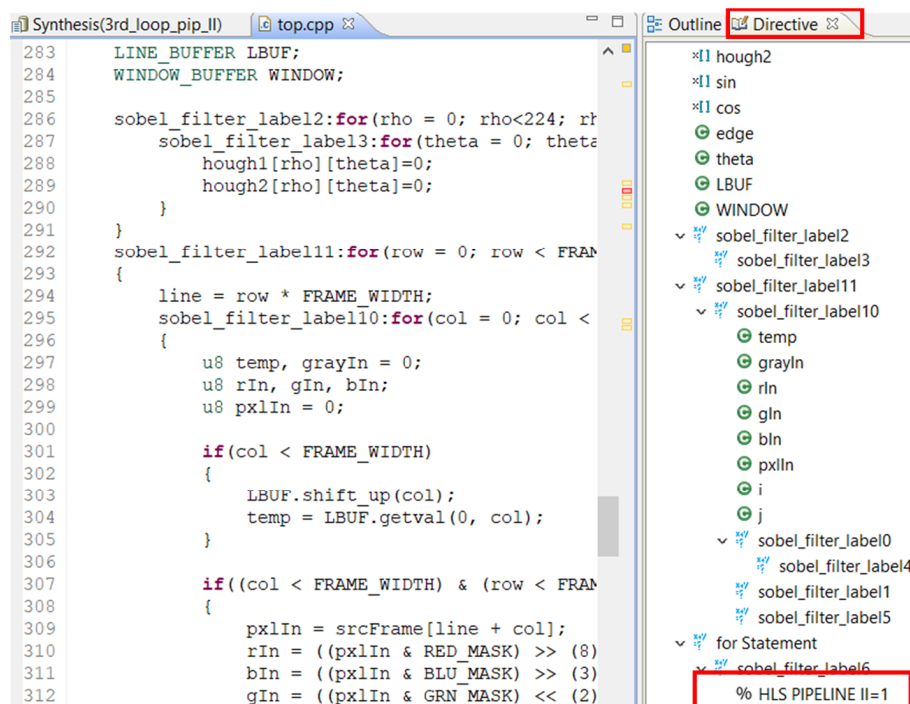


```

Synthesis(3rd_loop_pip_II) top.cpp
127   for(int row = 0; row < rows+1; row++){
128       for(int col = 0; col < cols+1; col++){
129   #pragma HLS loop_flatten off
130   #pragma HLS dependence variable=&buff_A false

```

ภาพประกอบ 3.11 ตัวอย่างการใช้งาน HLS pragmas ในรหัสต้นฉบับ



```

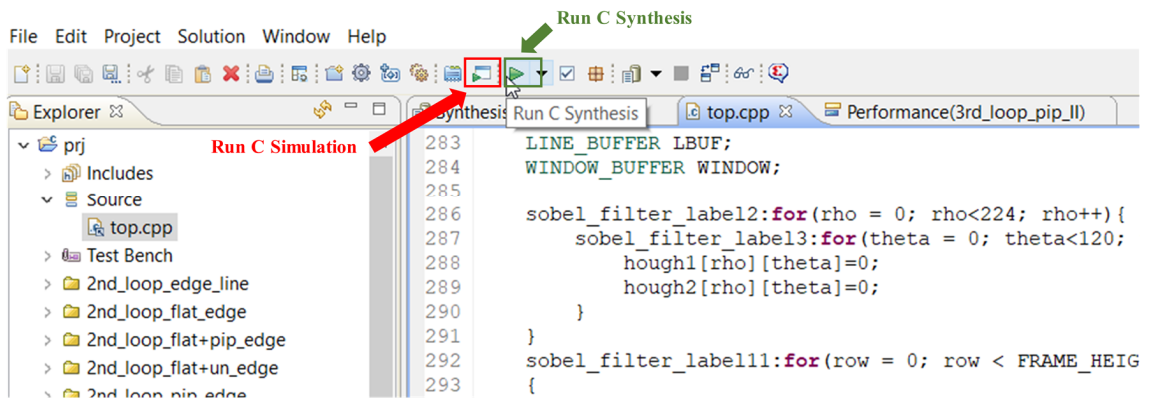
Synthesis(3rd_loop_pip_II) top.cpp
283   LINE_BUFFER LBUF;
284   WINDOW_BUFFER WINDOW;
285
286   sobel_filter_label2:for(rho = 0; rho<224; rh
287       sobel_filter_label3:for(theta = 0; theta
288       hough1[rho][theta]=0;
289       hough2[rho][theta]=0;
290   }
291   }
292   sobel_filter_label11:for(row = 0; row < FRAM
293   {
294       line = row * FRAME_WIDTH;
295       sobel_filter_label10:for(col = 0; col <
296   {
297       u8 temp, grayIn = 0;
298       u8 rIn, gIn, bIn;
299       u8 pxlIn = 0;
300
301       if(col < FRAME_WIDTH)
302       {
303           LBUF.shift_up(col);
304           temp = LBUF.getval(0, col);
305       }
306
307       if((col < FRAME_WIDTH) & (row < FRAM
308       {
309           pxlIn = srcFrame[line + col];
310           rIn = ((pxlIn & RED_MASK) >> (8)
311           bIn = ((pxlIn & BLU_MASK) >> (3)
312           gIn = ((pxlIn & GRN_MASK) << (2)

```

Outline Directive

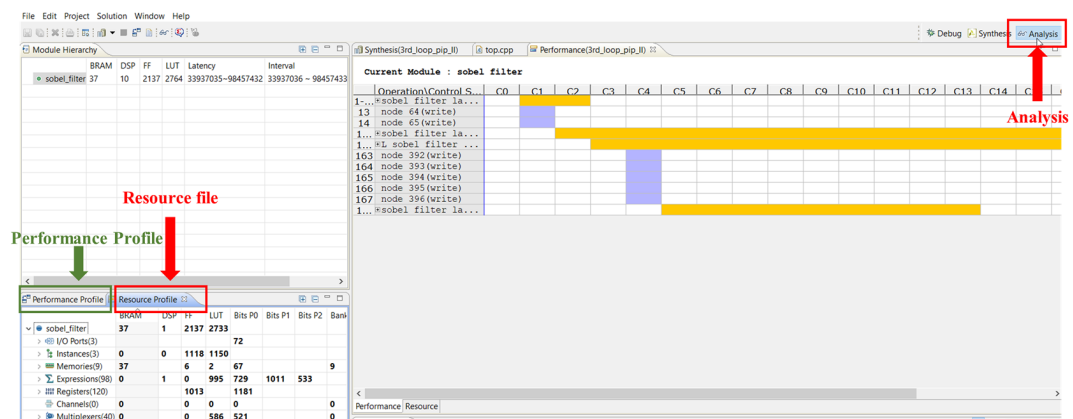
- *l1 hough2
- *l1 sin
- *l1 cos
- edge
- theta
- LBUF
- WINDOW
- sobel_filter_label2
- sobel_filter_label3
- sobel_filter_label11
- sobel_filter_label10
 - temp
 - grayIn
 - rIn
 - gIn
 - bIn
 - pxlIn
 - i
 - j
- sobel_filter_label0
- sobel_filter_label4
- sobel_filter_label1
- sobel_filter_label5
- for Statement
 - sobel_filter_label6
 - % HLS PIPELINE II=1

ภาพประกอบ 3.12 ตัวอย่างการใช้งาน HLS pragmas ในแถบคำสั่ง



ภาพประกอบ 3.13 การ Run C Synthesis

หลังจากการออกแบบ IP และการวิเคราะห์วิธีการเพิ่มประสิทธิภาพต่าง ๆ ที่เหมาะสมกับการหาขอบภาพและการหาเส้นตรงแล้วอัลกอริทึมและคำสั่ง HLS pragmas ที่ออกแบบไว้บน Vivado HLS จะถูกนำมาทดสอบอีกครั้งบนโปรแกรม SDSoC โดย SDSoC เป็นการทำงานร่วมกันของทั้งซอฟต์แวร์และฮาร์ดแวร์ซึ่งมีขั้นตอนการพัฒนาดังภาพประกอบ 3.16 ผู้ใช้งานสามารถทำเพิ่มวิธีการเพิ่มประสิทธิภาพสำหรับ PL ด้วยซอฟต์แวร์ได้เช่นเดียวกันกับบน Vivado HLS แต่การทำเพิ่มประสิทธิภาพบน SDSoC สามารถทำได้โดยการเพิ่มคำสั่งในรหัส

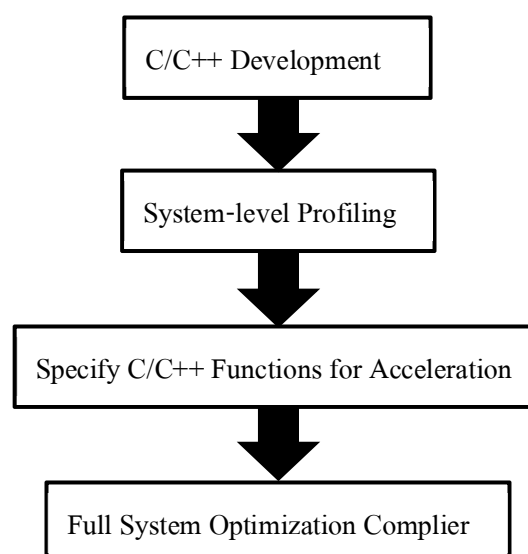


ภาพประกอบ 3.14 การวิเคราะห์ข้อมูลบน Vivado HLS Analysis

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
▼ ● sobel_filter	-	33937035~98457432	33937036 ~ 98457433	-	-
> ● sobel_filter_label2	no	27328	-	122	224
> ● sobel_filter_label11	no	202407 ~ 64722804	-	1007 ~ 322004	201
● L_sobel_filter_label6	yes	26895	1	17	26880
> ● sobel_filter_label9	no	33680400	-	168402	200

ภาพประกอบ 3.15 ตัวอย่าง Performance Profile

ต้นฉบับเท่านั้น ในแต่ละฟังก์ชันของอัลกอริทึมที่มีการออกแบบยังสามารถทำโปรไฟล์ (Profiling) ดังภาพประกอบ 3.17 แสดงผลการวิเคราะห์เวลาประมวลผลเป็นร้อยละของเวลาทั้งหมด โดยเราสามารถเลือกฟังก์ชันที่ต้องการเป็นตัวเร่งการทำงานบนฮาร์ดแวร์ (Hardware Accelerator: HA) ได้จากหน้าต่างโปรแกรมเพื่อดูระยะเวลาที่ใช้ในการประมวลผลของแต่ละฟังก์ชันได้เพื่อปรับการทำงานของฟังก์ชันที่มีร้อยละของเวลาประมวลผลมากเป็น HA ดังภาพประกอบ 3.18 เพื่อลดระยะเวลาในการประมวลผลและยังสามารถเลือกความถี่ของการทำงานบนอุปกรณ์ฮาร์ดแวร์ได้เช่นกัน และเมื่อเปลี่ยนฟังก์ชันเป็นฮาร์ดแวร์แล้ว สามารถดูผลประมาณความเร็วและทรัพยากรที่เปลี่ยนไปได้ดังภาพประกอบ 3.19 แต่เนื่องจากระยะเวลาที่ใช้ในการสร้างแพลตฟอร์มและ IP ใหม่บน SDSoC นั้นใช้เวลา 20-60 นาที เป็นอย่างน้อย ซึ่งค่อนข้างใช้เวลานานสำหรับการทำการทดลองแต่ละครั้งและไม่สามารถเปรียบเทียบผลจากการทำการเพิ่มประสิทธิภาพได้อย่างชัดเจนเหมือน Vivado HLS ดังนั้นในงานวิจัยนี้จะเลือกวิธีการออกแบบฟังก์ชันที่ใช้สำหรับทำ HA และทดสอบ

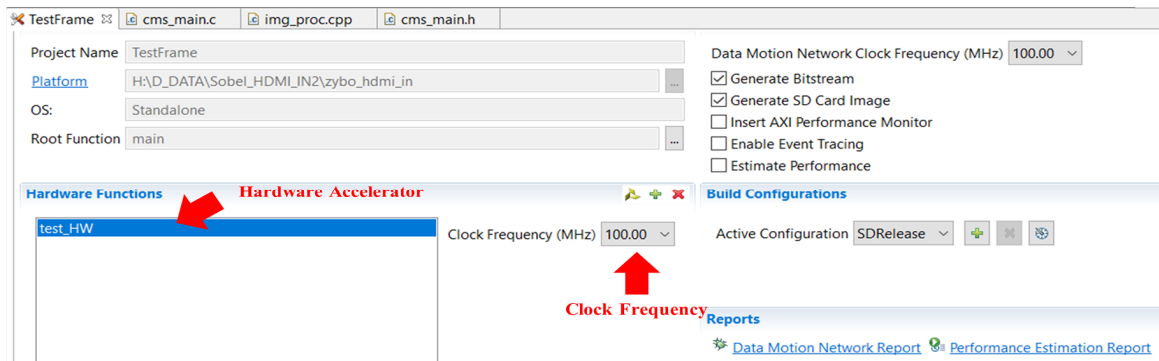


ภาพประกอบ 3.16 SDSoC Development Environment

วิธีการเพิ่มประสิทธิภาพที่เหมาะสมกับ IP บน Vivado HLS ก่อนการดำเนินการจริงบน SDSoC โดยแพลตฟอร์มของ HDMI (in) – VGA (out) สำหรับ Zybo z7-10 สามารถดาวน์โหลดได้ที่ [12]

Profiler running. 4122 samples					
Address	% Exc...	% Incl...	Function	File	Line
00148d64	9.58	9.58	clearUnusedBits	ap_private.h	2069
00106fcc	8.51	95.9	test_HW	img_proc.cpp	224
00148648	7.25	38.8	operator* <16, true>	ap_private.h	1950
00147d24	5.84	59.2	operator* <16, true>	ap_int_sim.h	1277
00148d00	5.84	12.3	ap_private	ap_private.h	1622
001486e8	5.43	14.1	ap_private	ap_private.h	1627
00148dc4	5.07	5.07	clearUnusedBits	ap_private.h	2069
00147f9c	4.85	4.85	clearUnusedBits	ap_private.h	2069
00147d8c	4.36	5.84	~ap_private	ap_private.h	1682
0014767c	4.34	5.50	~ap_private	ap_private.h	1682
00101ffc	3.93	100	main	cms_main.c	93
0014873c	3.17	3.17	check_canary	ap_private.h	1403
001487cc	2.71	2.71	check_canary	ap_private.h	1403
00148758	2.66	2.66	get_VAL	ap_private.h	1410
00147634	2.47	9.02	ap_private	ap_private.h	1613
00148908	2.45	8.51	ashr	ap_private.h	1740
00147e30	2.30	10.8	operator>>	ap_private.h	2424
0014861c	2.20	2.20	get_VAL	ap_private.h	1410
00148d48	2.20	2.20	set_canary	ap_private.h	1404
00147db4	2.06	8.41	operator+ <48, true>	ap_private.h	1997
00148ec4	1.91	6.06	ap_private	ap_private.h	1623
00148784	1.86	5.36	ap_private	ap_private.h	1622
00147ed0	1.84	3.17	operator ap_private <49, true, true...	ap_private.h	1815
Called From					
0011adc4		100	_start		
Child Calls					
00106fcc		96.0	test_HW	img_proc.cpp	224
0011bb48		.049	xil_printf		
0011aea0		.049	Xil_DCacheFlushRange		

ภาพประกอบ 3.17 การทำโปรไฟล์บน SDSoC



ภาพประกอบ 3.18 การกำหนดฟังก์ชันเป็น HA

3.4 กระบวนการเพิ่มประสิทธิภาพใน High-Level Synthesis

เนื่องจากระยะเวลาที่ใช้ในการประมวลผลของผลระบบตรวจจับเลนถนนบน PS เพียงอย่างเดียวค่อนข้างนาน จึงมีการนำขั้นตอนการหาขอบและขั้นตอนการหาเส้นบนภาพให้ทำงานบนส่วน PL โดยใช้ Vivado High-Level Synthesis ในการพัฒนาที่สามารถออกแบบโดยใช้ภาษา C หรือ C++ ได้และยังรองรับการใช้เทคนิคการเพิ่มประสิทธิภาพต่างๆ เช่น การแบ่งอาร์เรย์ การไปป์ไลน์ลูปและการคลี่ลูป เป็นต้น บนอุปกรณ์ Zybo z7-10 โดยขั้นตอนการใช้เทคนิคการเพิ่มประสิทธิภาพในงานวิจัยนี้จะถูกดำเนินการ 4 ขั้นตอนตามลำดับดังนี้

- ขั้นตอนที่ 1 การกำหนดขนาดอาร์เรย์
 - การพิจารณา memory depth
 - การพิจารณา data width
- ขั้นตอนที่ 2 การวิเคราะห์ลูป
- ขั้นตอนที่ 3 การแบ่งอาร์เรย์

- ขั้นตอนี่ 4 การจัดการอินเทอร์เฟซ HLS

Details

Performance estimates for 'test_HW in cms_main.c:154' fun ...

HW accelerated (Estimated cycles)	453438624
-----------------------------------	-----------

Resource utilization estimates for hardware accelerators

Resource	Used	Total	% Utilization
DSP	8	80	10
BRAM	10	60	16.67
LUT	1118	17600	6.35
FF	794	35200	2.26

ภาพประกอบ 3.19 การประมาณค่าการทำงานของ HA

3.4.1 การกำหนดขนาดอาร์เรย์

ในขั้นตอนนี้เป็นขั้นตอนแรกที่ต้องทำการพิจารณา เนื่องจากชนิดของตัวแปร ขนาดของตัวแปรมีผลต่อการใช้ทรัพยากรฮาร์ดแวร์ โดยเฉพาะ Block RAM ซึ่งเป็นหน่วยความจำที่กระจายภายใน PL เพื่อใช้เป็นหน่วยความจำย่อยๆ รองรับการประกาศตัวแปรแบบอาร์เรย์ และ CLB (Configurable Logic Block) ซึ่งเป็นส่วนสร้างลอจิกเกตรวมเป็นวงจรเชิงจัดหมู่ (Combinational logic circuit) และภายในมี Flip-flop เพื่อเป็นทางเลือกในการสร้างวงจรเชิงลำดับ (Sequential logic circuit) โดยจำนวน Block RAM ที่ถูกใช้ภายใน PL จะเป็นไปตามสมการ (3.1) ซึ่งจะเห็นได้ว่าค่าของ memory depth, data width และขนาดของ Block RAM เป็นตัวแปรที่ส่งผลต่อจำนวน Block RAM ที่ใช้ ดังนั้นการกำหนดขนาดอาร์เรย์จึงมีการแบ่งขั้นตอนย่อยๆ ออกเป็น 2 ขั้นตอนคือ 1. การพิจารณา memory depth และ 2. การพิจารณา data width

3.4.1.1 การพิจารณา memory depth

ขนาดของ memory depth จะขึ้นอยู่กับจำนวนอิลีเมนต์ของอาร์เรย์นั้นๆ โดยขนาดของ memory depth ที่ถูกสังเคราะห์จะเป็นไปตามหลักการของเลขฐานสอง เช่น การประกาศตัวแปรอาร์เรย์ (x[2050]) ซึ่งมีจำนวนอิลีเมนต์ทั้งหมด 2,050 อิลีเมนต์ จำนวนของ memory depth ที่ใช้จริงจะมีค่าเท่ากับ 4,096 อิลีเมนต์ ตามจำนวนของเลขฐานสอง ทำให้เกิดพื้นที่ของ memory depth จำนวนมากที่ไม่ได้ใช้งาน

จากทฤษฎี HT การเขียนอัลกอริทึมภาษาซีจำเป็นต้องมีการประกาศตัวแปรอาร์เรย์ 2 มิติ (hough[rho][theta], rho มีค่าเท่ากับ ρ , theta มีค่าเท่ากับจำนวน θ)

เพื่อรองรับจำนวนครั้งที่เกิดจุดตัดของเส้นโค้งบนระนาบ $\rho-\theta$ สำหรับหาความยาวของเส้นตรง และตัวแปรอาร์เรย์ 1 มิติ 2 ตัวคือ $\sin[\theta]$ และ $\cos[\theta]$ สำหรับเก็บค่า sine และ cosine ที่แต่ละองศาเพื่อลดระยะเวลาในการคำนวณ

การพิจารณา memory depth ในวิทยานิพนธ์นี้เกี่ยวเนื่องมาจากขั้นตอนการเตรียมภาพของระบบการตรวจจับเลนถนนซึ่งมีการตัดภาพบริเวณที่ไม่เกี่ยวข้องบนภาพเพื่อลดปัญหาสัญญาณรบกวนที่อาจจะส่งผลต่อความผิดพลาดในการคำนวณของระบบ ทำให้ขนาดของภาพขาเข้าลดขนาดจาก 1920×1080 พิกเซล เหลือ 200×200 พิกเซล ดังนั้นเส้นทะแยงมุมของภาพมีค่าเท่ากับ 224 พิกเซล ซึ่งมีค่าเท่ากับค่า ρ ดังนั้นขนาดของอาร์เรย์ในมิติที่ 1 ของตัวแปร $\text{hough}[\rho][\theta]$ 224 เมื่อคำนวณร่วมกับขนาดของอาร์เรย์ในมิติที่ 2 ซึ่งมีค่าเท่ากับ 360 (θ เท่ากับ 360) memory depth จะมีค่าเท่ากับ 131,076 (2^{17}) โดยจำนวนอิลีเมนต์ของอาร์เรย์ที่ต้องการเท่ากับ 80,640 อิลีเมนต์ ซึ่งจำนวน memory depth ที่ถูกใช้ไปนั้นเกินกว่าที่ต้องการในความเป็นจริงค่อนข้างมาก ดังนั้นจึงมีการลดขนาดของค่า θ ลงจาก 360 เหลือ 180 เนื่องจากองศาที่ใช้งานจริงในระบบตรวจจับเลนถนนในที่นี้ต้องการเพียง 0-180 องศา เท่านั้น จำนวน memory depth ที่ต้องการเพียง 65,536 (2^{16}) โดยจำนวนอิลีเมนต์ที่ต้องการเท่ากับ 40,320 อิลีเมนต์ ซึ่งใกล้เคียงกับจำนวน memory depth ที่ถูกใช้มากขึ้น

$$\text{Number of Block RAM} = \frac{\text{memory depth} \times \text{data width}}{\text{block RAM size}} \quad (3.1)$$

3.4.1.2 การพิจารณา data width

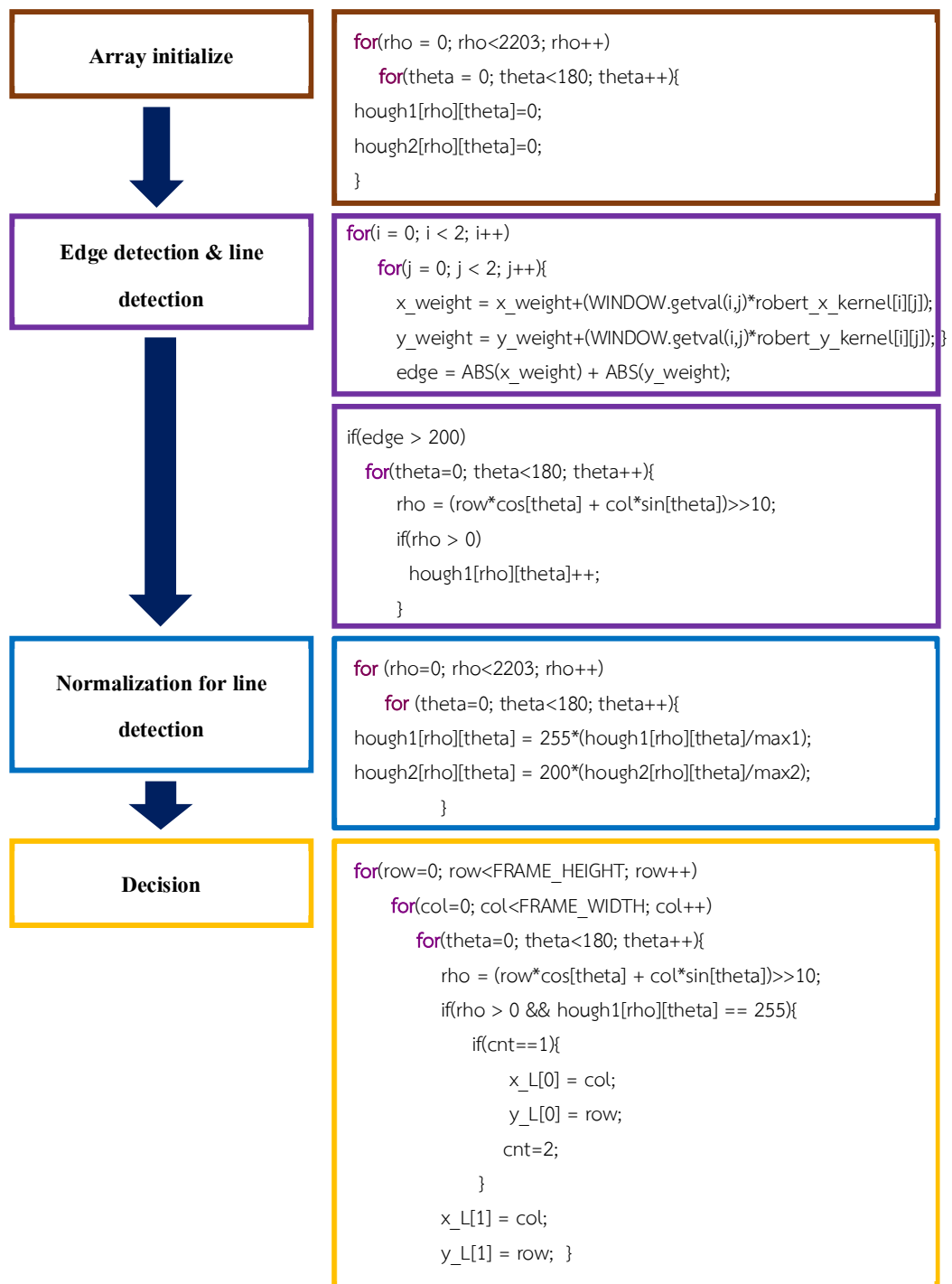
การเลือกประเภทประเภทของตัวแปรไม่ว่าจะเป็น int, float, double หรือ char มีผลมาจากลักษณะของข้อมูล เช่น ข้อมูลที่มีค่าเป็นจำนวนเต็มการประกาศตัวแปรมักจะประกาศเป็น integer (int) และในกรณีที่ข้อมูลมีค่าเป็นเลขทศนิยมการประกาศตัวแปรจะให้เป็นชนิด float ซึ่งประเภทของตัวแปรแต่ละชนิดมีจำนวน data width ที่แตกต่างกันออกไป นอกจากนี้ค่าของข้อมูลสูงสุดและต่ำสุดยังส่งผลต่อการกำหนดขนาดของ data width เช่นกัน เช่นการประกาศตัวแปรที่เป็นจำนวนเต็ม (integer) โดยตัวแปรจะถูกเก็บค่าจำนวนเต็มตั้งแต่ 0-360 จำเป็นต้องมีการประกาศตัวแปรเป็น integer ขนาด 16 บิต แทนที่ 8 บิต เนื่องจากช่วงของข้อมูลที่ถูกเก็บมีขนาดใหญ่

จากหัวข้อ 3.4.1.1 การพิจารณา memory depth พบว่าค่าสูงสุดที่ถูกบันทึกของตัวแปรอาร์เรย์ ($\text{hough}[\rho][\theta]$) ลดลง จากเดิมตัวแปรนี้ถูกประกาศเป็นชนิด

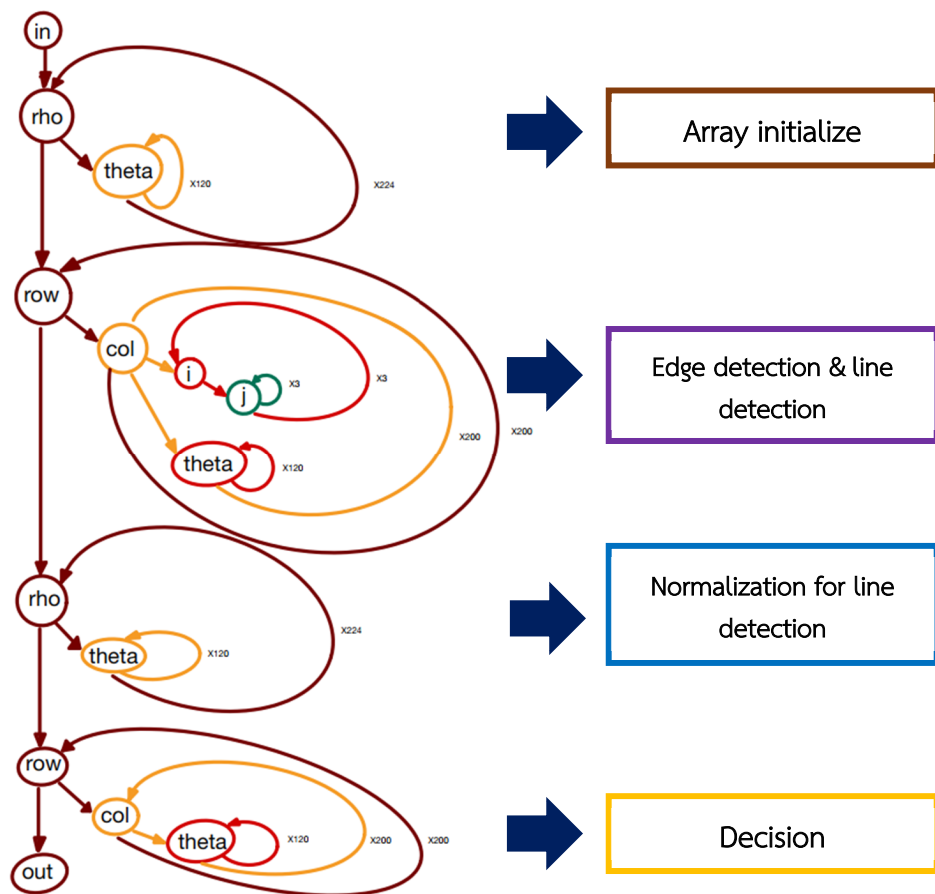
unsigned integer ขนาด 16 บิต สามารถลดลงเหลือ 8 บิต เมื่อคำนวณจำนวน Block RAM ที่ต้องการ ตามสมการ 3.1 พบว่าจำนวน Block RAM ที่ใช้สำหรับตัวแปรนี้ลดลงเช่นกัน นอกจากนี้ยังมีตัวแปร $\sin[\theta]$ และ $\cos[\theta]$ ได้ทำการเปลี่ยนชนิดของตัวแปรจาก float ที่รองรับข้อมูลที่เป็นทศนิยมเป็น integer ที่รองรับเฉพาะจำนวนเต็ม โดยค่าของตัวแปร $\sin[\theta]$ และ $\cos[\theta]$ ทุกค่า จะถูกคูณเข้าไปเป็นจำนวนเต็มและใช้การหารหรือเลื่อนบิตของตัวแปรเข้ามาช่วยในการคำนวณภายหลัง การเพื่อลดขนาดของหน่วยความจำลง

3.4.2 การวิเคราะห์หลูป

จากทฤษฎีของการหาขอบภาพโดยใช้วิธีการของโรเบิร์ตและการหาเส้นบนภาพโดยใช้หลักการของ HT สามารถรวบรวมขั้นตอนทั้งหมดหลัก ๆ ออกเป็น 4 ขั้นตอน ซึ่งถูกออกแบบโดยใช้ภาษาซีดังแสดงในภาพประกอบ 3.20 และถูกแสดงในรูปแบบของลูปซ้อนลูปในภาพประกอบ 3.21 ในขั้นตอนแรกคือการกำหนดค่าเริ่มต้นของอาร์เรย์ (Array initialize) จะเป็นลูปซ้อนลูป 2 ชั้น ซึ่งเป็นการกำหนดค่าเริ่มต้นของตัวแปร $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ (ตัวแปร $\text{hough}[\rho][\theta]$ ถูกแบ่งย่อยออกเป็นสองตัวเพื่อแยกเก็บค่าของถนนทางซ้ายและขวาตามขั้นตอนการเตรียมภาพ 3.1.1) ให้มีค่าเท่ากับศูนย์สำหรับรองรับการนับจำนวนครั้งที่เกิดขึ้นบนระนาบ $\rho-\theta$ ขั้นตอนที่สองคือการหาขอบภาพและการหาเส้นตรงจะถูกแบ่งออกเป็นย่อยออกเป็น 2 ลูปซ้อนลูปโดยลูปซ้อนลูปแรกจะเป็นขั้นตอนการหาขอบภาพที่มีลักษณะเป็นลูปซ้อนลูปขนาด 4 ชั้น สำหรับลูปซ้อนลูปที่สองเป็นขั้นตอนการหาเส้นบนภาพซึ่งมีลักษณะเป็นลูปซ้อนลูปขนาด 3 ชั้น ขั้นตอนที่ 3 คือการปรับค่ามาตรฐานสำหรับการหาเส้นตรง (Normalization for line detection) มีลูปซ้อนลูปลักษณะ 2 ชั้น ขั้นตอนที่สุดท้ายคือขั้นตอนการตัดสินใจ (Decision) มีลูปซ้อนลูปลักษณะ 3 ชั้น สำหรับการเลือกเส้นบนภาพโดยตรวจสอบจากตัวแปรอาร์เรย์ $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ เพื่อหาตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายของเส้นบนภาพ



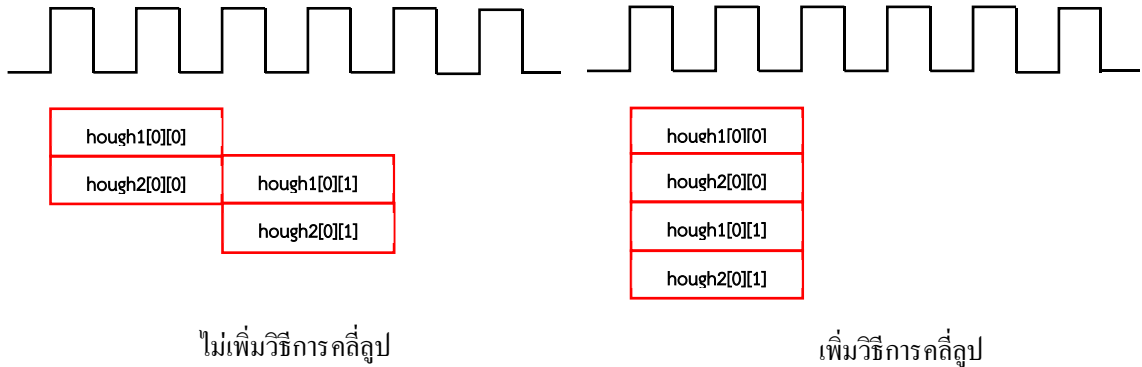
ภาพประกอบ 3.20 โปรแกรมภาษาซีของการหาขอบภาพและเส้นบนภาพ



ภาพประกอบ 3.21 รูปขั้นตอนการทำงานของระบบตรวจจับเลนถนน

จากขั้นตอนของการหาขอบภาพและหาเส้นของภาพจะเห็นได้ว่าทั้งสองขั้นตอนจำเป็นต้องออกแบบโดยการใช้ลูปซ้อนลูปทั้งหมด ซึ่งลูปซ้อนลูปนี้เองเป็นสาเหตุทำให้ระยะเวลาที่ใช้ในการทำงานของระบบช้าลง ในหัวข้อนี้จึงเป็นการนำเอาวิธีการเพิ่มประสิทธิภาพต่างๆ มาปรับใช้ให้เหมาะสมกับลูปซ้อนลูปในแต่ละขั้นตอน ตัวแปรอาร์เรย์ $\text{hough1}[\text{rho}][\text{theta}]$ และ $\text{hough2}[\text{rho}][\text{theta}]$ มีการกำหนดค่าเริ่มต้นในลูปซ้อนลูปแรก โดยการทำงานแต่ละครั้งในลูปซ้อนลูปนั้นเป็นอิสระต่อกัน ดังนั้นทุกอิลิเมนต์ของอาร์เรย์สามารถทำงานพร้อมกันได้ดังเช่นภาพประกอบ 3.22 เป็นการแสดงตัวอย่างการทำงานแบบปกติที่ 2 รอบ ของการกำหนดค่าเริ่มต้นของตัวแปรอาร์เรย์ $\text{hough1}[\text{rho}][\text{theta}]$ และ $\text{hough2}[\text{rho}][\text{theta}]$ ต้องใช้จำนวน รอบนาฬิกาถึง 4 รอบ ในกรณีที่มีการคลี่ลูปด้วยค่า factor เท่ากับ 2 จำนวนรอบนาฬิกาที่ใช้ในการทำงาน 2 รอบ จะลดลงเหลือเพียง 2 รอบนาฬิกา

$$hough_{(1,2)}[rho][theta] = hough_{(1,2)}[rho][theta] + 1 \quad (3.2)$$



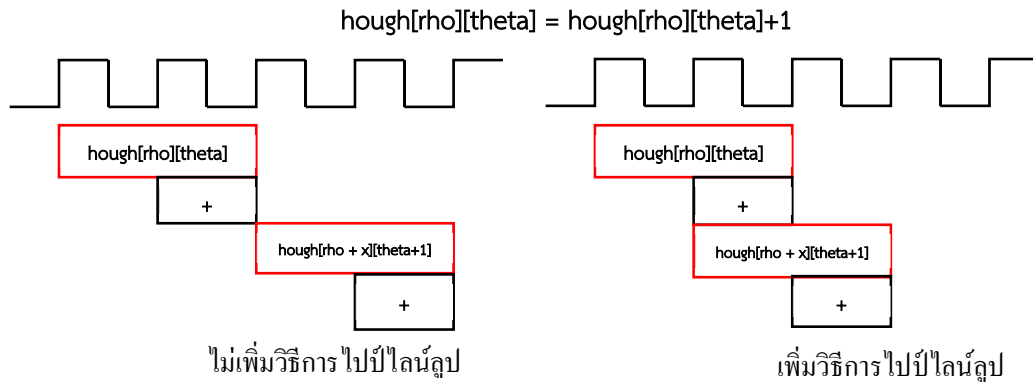
ภาพประกอบ 3.22 การคลี่รูปสำหรับขั้นตอนการกำหนดค่าเริ่มต้นของอาร์เรย์ของภาพประกอบ 3.21

ในขั้นตอนของการทำการหาขอบภาพในรูปซ้อนรูปย่อยรูปแรกมีลักษณะการทำงานในแต่ละรอบเป็นแบบไม่อิสระต่อกัน จากหลักการหาขอบภาพด้วยวิธีการของ Robert จะมีการประกาศตัวแปร x_weight และ y_weight สำหรับรองรับค่าที่ได้จากการทำคอนโวลูชัน (convolution) ซึ่งเป็นตัวแปรสะสมค่า ดังนั้นการคลี่รูปอาจจะไม่เหมาะสมกับรูปซ้อนรูปย่อยนี้

ในขั้นตอนของการทำการหาเส้นบนภาพในรูปซ้อนรูปย่อยที่ 2 เป็นการนับจำนวนครั้งที่เกิดจุดตัดที่ตำแหน่งเดิมบนระนาบ ρ และ θ ตามหลักการของ HT ดังสมการที่ (2.6) จะเห็นได้ว่า $hough1[rho][theta]$ และ $hough2[rho][theta]$ เป็นตัวแปรสะสมเช่นกันซึ่งมีการทำงานแบบไม่อิสระต่อกัน ดังนั้นไปป์ไลน์รูปจึงมีความเหมาะสมกับการปรับใช้ในรูปซ้อนรูปย่อยนี้ ตัวอย่างของลำดับการทำงานในสมการที่ (3.2) ที่ 2 รอบการทำงานเป็นดังภาพประกอบ 3.21

การทำการปรับค่ามาตรฐานของตัวแปรอาร์เรย์ $hough1[rho][theta]$ และ $hough2[rho][theta]$ เกิดขึ้นในขั้นตอนที่สามซึ่งคล้ายกับการทำงานในรูปซ้อนรูปแรกคือการทำงานแต่ละรอบจะเป็นอิสระต่อกัน ดังนั้นการเพิ่มการคลี่รูปในการทำงานมีความเหมาะสมมากกว่าการทำไปป์ไลน์รูป

สำหรับทุกๆอิลีเมนต์ของตัวแปร $hough1[rho][theta]$ และ $hough2[rho][theta]$ จะถูกตรวจสอบค่าผลลัพธ์ที่ได้เพื่อใช้ในการระบุตำแหน่งของเส้นถนนในทุกๆรอบของการทำงาน เนื่องจากการทำงานในแต่ละรอบของรูปซ้อนรูปนี้มีความเป็นอิสระต่อกัน การปรับใช้การคลี่รูปจึงสามารถช่วยลดระยะเวลาที่ใช้ในการประมวลผลได้ดีกว่าการทำไปป์ไลน์รูปแต่ด้วยขีดจำกัดของทรัพยากรที่มีรองรับบนอุปกรณ์ การทำไปป์ไลน์รูปจึงต้องถูกปรับใช้กับรูปซ้อนรูปนี้แทนการคลี่รูปเนื่องจากการใช้ทรัพยากรที่น้อยกว่า

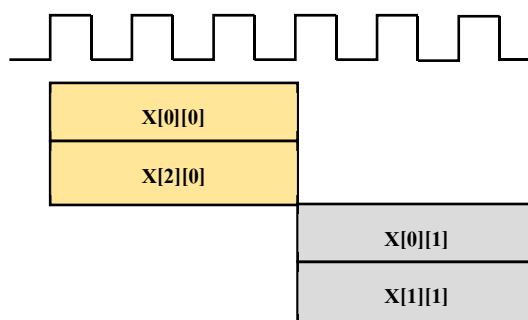


ภาพประกอบ 3.23 การทำไปป์ไลน์รูปสำหรับขั้นตอนการหาขอบภาพและการหาเส้นตรงในภาพประกอบ 3.21

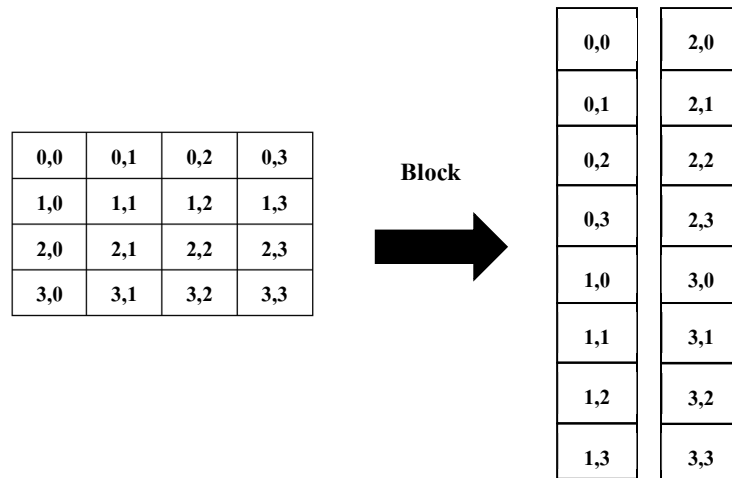
3.4.3 การแบ่งอาร์เรย์สำหรับการคลี่รูปและการไปป์ไลน์รูป

การแบ่งอาร์เรย์เป็นวิธีการเพิ่มประสิทธิภาพการทำงานวิธีการหนึ่งที่สามารถช่วยเพิ่มความเร็วในการประมวลผลหรือลดระยะเวลาหรือลดจำนวนทรัพยากรที่ใช้บนอุปกรณ์ฮาร์ดแวร์ได้ วิธีการนี้ยังเป็นวิธีการที่ช่วยส่งเสริมการทำงานของวิธีการไปป์ไลน์รูปและการคลี่รูปอีกด้วย เช่น ตัวแปรอาร์เรย์ 2 มิติ ขนาด 4x4 หากมีการคลี่รูปและมีลำดับการเข้าถึงอติเมนต์ของอาร์เรย์เป็นดังภาพประกอบ 3.24 การทำการแบ่งอาร์เรย์แบบบล็อกที่มีมิติ 1 ด้วยค่า factor เท่ากับ 2 เป็นวิธีที่เหมาะสมและเพียงพอสำหรับการทำการคลี่รูปภาพประกอบ 3.25 แล้ว

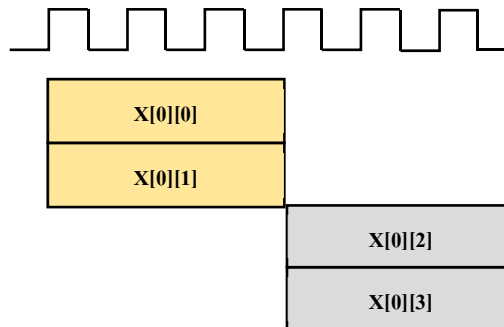
หากการทำการคลี่รูปโดยมีลำดับการเข้าถึงอติเมนต์ของอาร์เรย์เป็นดังภาพประกอบ 3.26 การแบ่งอาร์เรย์แบบวงกลมที่มีมิติที่ 2 ด้วย factor เท่ากับ 2 เหมาะสมกับการทำการคลี่รูปดังภาพประกอบ 3.27



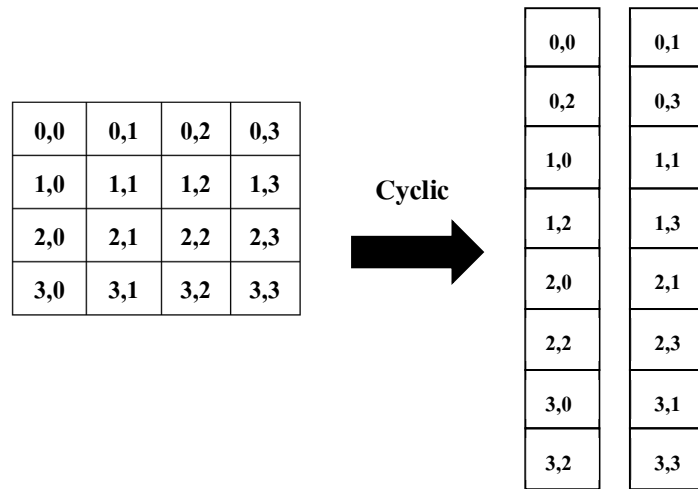
ภาพประกอบ 3.24 ตัวอย่างการคลี่รูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2x2 ที่ลูปชั้นนอก



ภาพประกอบ 3.25 การแบ่งอาร์เรย์แบบบล็อกในมิติที่ 2



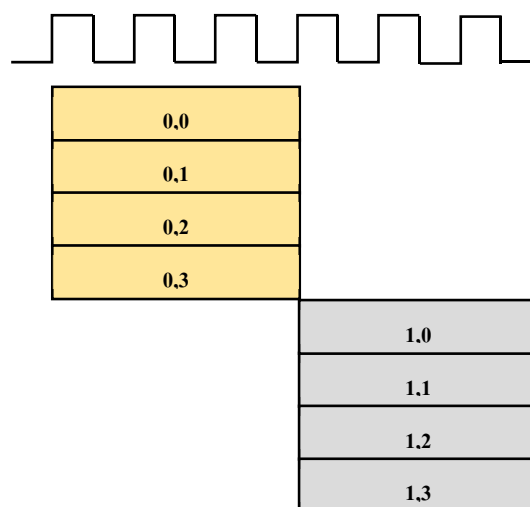
ภาพประกอบ 3.26 ตัวอย่างการคลี่ลูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2×2 ที่ลูปชั้นใน



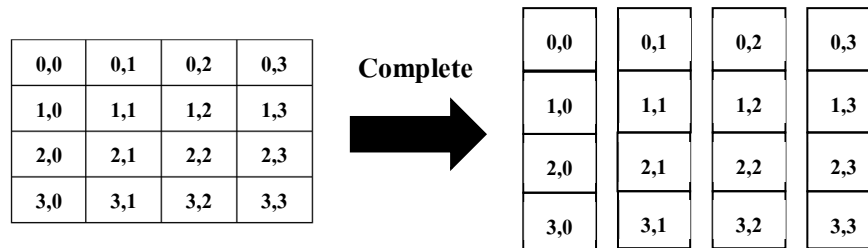
ภาพประกอบ 3.27 การแบ่งอาร์เรย์แบบวงกลมในมิติที่ 2

หากการทำการคลี่รูปโดยมีลำดับการเข้าถึงอีลีเมนต์ของอาร์เรย์เป็นดังภาพประกอบ 3.28 การแบ่งอาร์เรย์แบบสมบูร์นที่มีมิติที่ 2 ด้วย factor เท่ากับ 2 เหมาะสมกับการทำการคลี่รูปดังภาพประกอบ 3.29

ในกรณีของตัวแปรอาร์เรย์ $\text{hough1}[\text{rho}][\text{theta}]$ และ $\text{hough2}[\text{rho}][\text{theta}]$ สำหรับการทำการระบบตรวจจับเลนถนน ลำดับการเข้าถึงแต่ละอีลีเมนต์ของอาร์เรย์ทั้งสองจะขึ้นอยู่กับลำดับของมิติที่ 2 หรือค่า theta ของอาร์เรย์เป็นหลักโดยแต่ละรอบอีลีเมนต์จะเปลี่ยนไปเป็น $\text{theta}+1$ เมื่อทำ



ภาพประกอบ 3.28 ตัวอย่างการคลี่รูปสำหรับการวนลูปซ้อนลูปของตัวแปรอาร์เรย์ขนาด 2×2 ที่ลูปชั้นในพร้อมกัน 4 อีลีเมนต์



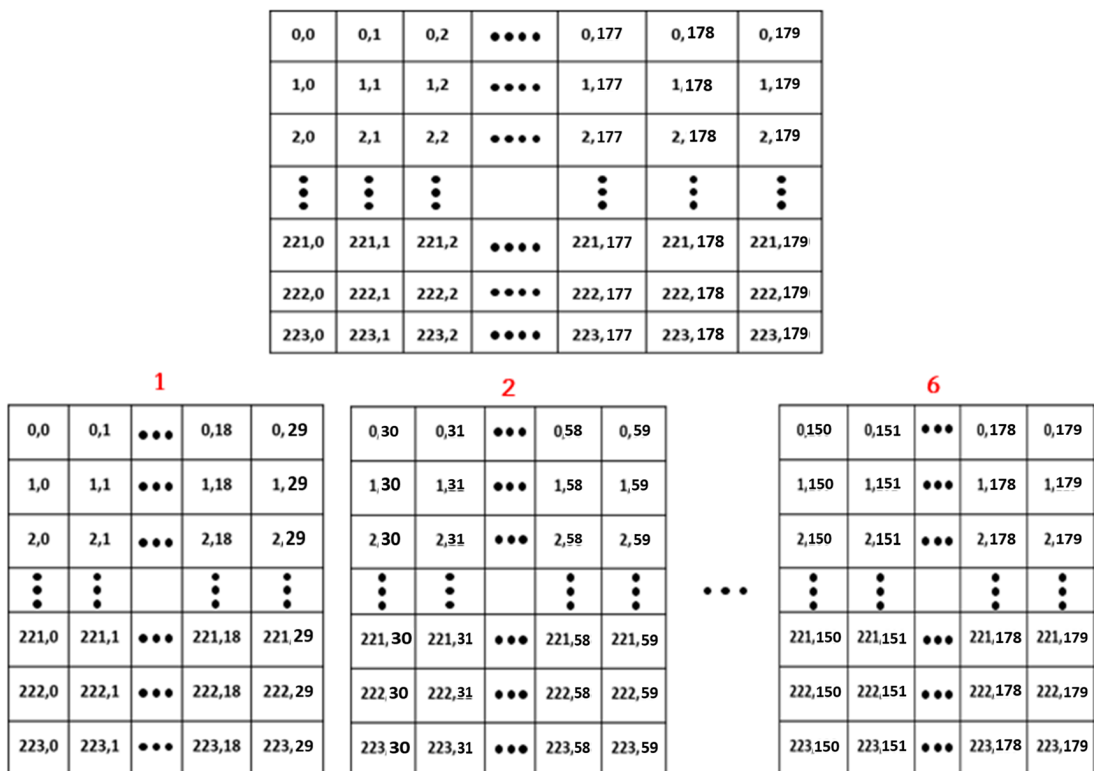
ภาพประกอบ 3.29 การแบ่งอาร์เรย์แบบสมมาตรในมิติที่ 2

การคลี่รูปที่ค่า Factor มากกว่าสองหรือเท่ากับ θ หรือการทำไปป์ไลน์รูปที่ $\Pi=1$ อีลีเมนต์ของ $\text{hough1}[x][\theta]$, $\text{hough1}[x][\theta+1]$, $\text{hough2}[x][\theta]$ และ $\text{hough2}[x][\theta+1]$ จะถูกเข้าถึงพร้อมกัน ดังนั้นการทำการแบ่งอาร์เรย์ควรเป็นแบบสมมาตรที่มิติที่ 2 ของอาร์เรย์จะทำให้การคลี่รูปและการไปป์ไลน์รูปทำงานได้เต็มประสิทธิภาพสูงสุด แต่เนื่องจากข้อจำกัดของทรัพยากรบนอุปกรณ์การออกแบบการคลี่รูปและการไปป์ไลน์รูปควบคู่ไปกับการแบ่งอาร์เรย์แบบสมมาตรจึงไม่สามารถทำได้ เนื่องจากเมื่อคำนวณจำนวน Block RAM ที่ใช้สำหรับรองรับอาร์เรย์ $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ เมื่อมีการแบ่งอาร์เรย์แบบสมมาตรตามสมการ (3.1) โดยชนิดของข้อมูลเป็น unsigned int 8 บิต ขนาดของอาร์เรย์เท่ากับ 224×180 ซึ่งแต่ละกลุ่มย่อยของอาร์เรย์จะมีการใช้จำนวน Block RAM เท่ากับ 16 ตัว ดังนั้นจำนวน Block RAM ที่ต้องการใช้ทั้งหมดจึงมีค่าเท่ากับ 360 ตัว โดยบนอุปกรณ์จริงมีจำนวน Block RAM รองรับเพียง 120 ตัว

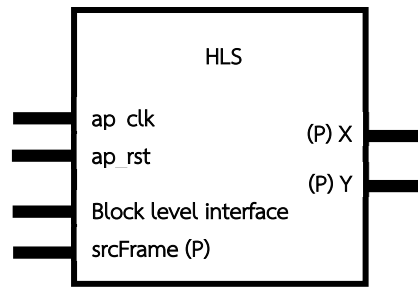
ในวิทยานิพนธ์นี้จะมีการปรับใช้การแบ่งอาร์เรย์แบบบล็อกที่ค่า Factor เท่ากับ 6 ในมิติที่ 2 กับตัวแปรอาร์เรย์ $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ ดังในภาพประกอบ 3.30 จะเห็นได้ว่าเดิมตัวแปรอาร์เรย์ที่ถูกจัดสรรใน Block RAM เดียวกันจะถูกแบ่งจัดสรรย่อยออกเป็น 6 Block RAM ขนาดเล็ก โดยในแต่ละกลุ่มย่อยจะมีค่า memory depth เท่ากับ 8,192 จำนวน Block RAM ที่ใช้มีค่าเท่ากับ 4 ตัว ดังนั้นที่ 6 กลุ่มย่อยใช้จำนวน Block RAM ทั้งหมดเท่ากับ 24 ตัว

3.4.4 การจัดการอินเทอร์เฟส HLS สำหรับอาร์เรย์

การออกแบบระดับวงจรหรือ Register Transfer Level (RTL) ถูกสร้างโดย Xilinx Vivado HLS ซึ่งแบบ RTL จะประกอบไปด้วยพอร์ตขาเข้าและออก ซึ่งการออกแบบ RTL ของการหาขอบภาพและการหาเส้นบนภาพประกอบไปด้วยพอร์ตขาเข้า 1 พอร์ต โดยมีชนิดของข้อมูลเป็นอาร์เรย์ขนาด 1 มิติ สำหรับรับข้อมูลภาพและพอร์ตขาออก 2 พอร์ต ชนิดของข้อมูลเป็นอาร์เรย์ขนาด 1 มิติ เช่นกัน สำหรับส่งข้อมูลของตำแหน่งเส้นบนภาพเพื่อใช้คำนวณมุมต่อไปดังภาพประกอบ 3.31 ซึ่งในการออกแบบอินเทอร์เฟส HLS จำเป็นต้องมีการคำนึงถึงลักษณะหรือชนิดของพอร์ตที่เหมาะสมเพื่อช่วยให้ระบบสามารถส่งผ่านข้อมูลไปและกลับได้ถูกต้องและ



ภาพประกอบ 3.30 การแบ่งอาร์เรย์สำหรับตัวแปร hough1[rho][theta] และ hough2[rho][theta]



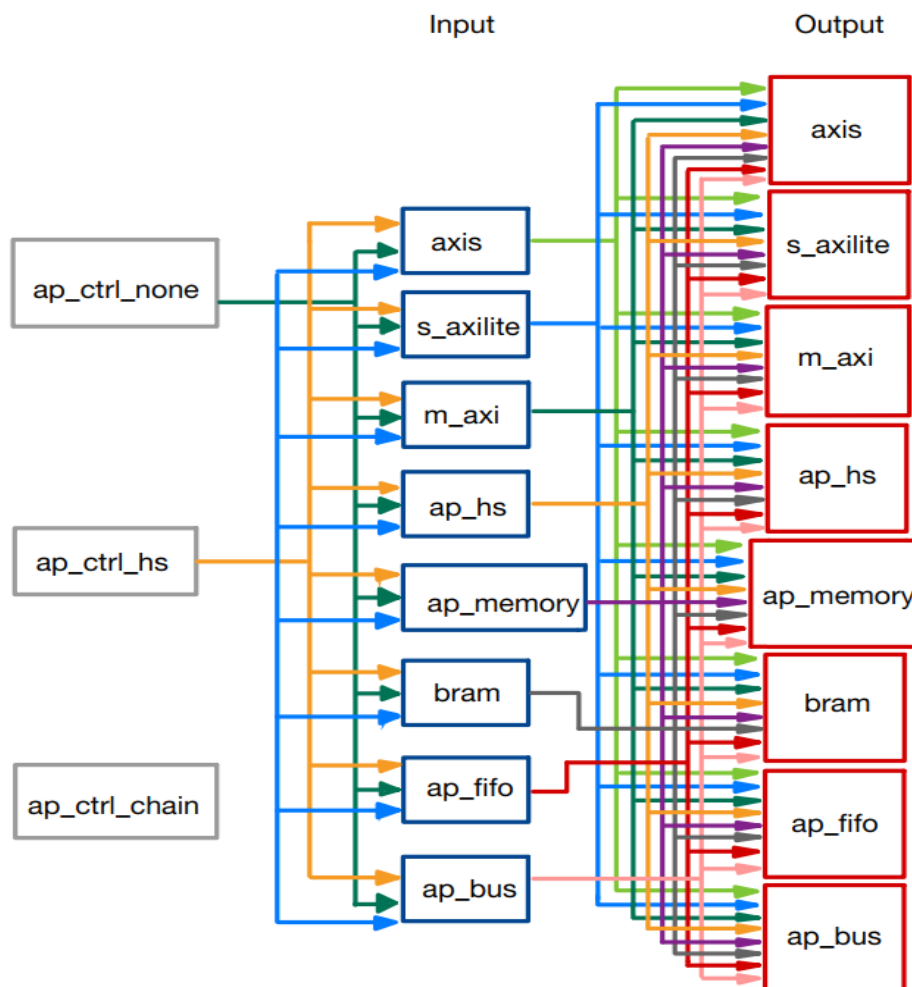
ภาพประกอบ 3.31 IP block

รวดเร็วที่สุด จากคู่มือการใช้งานดังตารางที่ 2 - 1 ชนิดของอินเทอร์เฟซ Block level ที่รองรับลักษณะของอาร์เรย์มี ap_ctrl_none, ap_ctrl_hs และ ap_ctrl_chain ส่วนลักษณะของอินเทอร์เฟซ Port level ที่มีรองรับสำหรับอาร์เรย์มี axis, s_axilite, m_axi, ap_hs, ap_memory, bram, ap_fifo และ ap_bus สำหรับใช้รับและส่งข้อมูลระหว่างการทำงานของ PS และ PL

เนื่องจากอินเทอร์เฟซ Block level และอินเทอร์เฟซ Port level ที่รองรับอาร์เรย์อาทิวเมนต์มีหลากหลายชนิดด้วยกันจึงต้องมีการจับแต่ละคู่ระหว่างอินเทอร์เฟซ Block level และอินเทอร์เฟซ Port level นอกจากการจับคู่ระหว่างอินเทอร์เฟซ Block level และอินเทอร์เฟซ Port level แล้วยังต้องมีการจับคู่ระหว่างอินเทอร์เฟซ Port level สำหรับอาร์เรย์อาทิวเมนต์ขาเข้าและอินเทอร์เฟซ Port level สำหรับอาร์เรย์อาทิวเมนต์ขาออก เนื่องจากอินเทอร์เฟซ HLS แต่ละชนิดให้ประสิทธิภาพการทำงานที่แตกต่างกันออกไปดังภาพประกอบ 3.32

3.5 สรุป

จากการออกแบบระบบตรวจจับเลนถนนพบว่าการคำนวณมุมของเส้นบนถนนสามารถคำนวณได้จากตำแหน่งเริ่มต้นและสุดท้ายของเส้นตรงที่ได้จากการทำ HT ซึ่งสามารถนำมุมที่ได้ไปควบคุมต่อไป โดยการทำให้ระบบการตรวจจับเลนถนน สามารถลดระยะเวลาที่ใช้ในการประมวลผลโดยย้ายขั้นตอนของการหาขอบภาพและหาเส้นตรงให้ประมวลผลในส่วนของ PL ซึ่งยังสามารถทำการเพิ่มประสิทธิภาพการทำงานให้ดีขึ้นได้ดังวิธีดังนี้ วิธีการปรับขนาดของอาร์เรย์เป็นวิธีการหลักสำหรับการลดจำนวนทรัพยากร ซึ่งเหมาะสมกับการดำเนินการบนอุปกรณ์ที่มีขีดจำกัดเรื่องทรัพยากรสูง วิธีการทำการคลี่ลูปเป็นวิธีการที่เหมาะสมกับลูปซ้อนลูปแบบอิสระต่อกัน การทำงานสามารถทำงานพร้อมกันที่รอบนาฬิกาเดียวกันได้ ส่วนวิธีการทำไปป์ไลน์ลูปเป็นวิธีที่เหมาะสมกับลูปซ้อนลูปแบบไม่อิสระต่อกัน การทำงานจะไม่ได้เริ่มทำงานพร้อมกันที่รอบ



ภาพประกอบ 3.32 การจับคู่อินเทอร์เฟส HLS

นาฬิกาเดียวกันแต่สามารถเริ่มทำงานก่อนขั้นตอนการทำงานก่อนหน้าเสร็จสมบูรณ์ ซึ่งทั้งสองวิธีการทั้งการคลี่ลูปและการไปป์ไลน์ลูปจะทำงานได้เต็มประสิทธิภาพจำเป็นต้องทำงานควบคู่ไปกับการทำการแบ่งอาร์เรย์เพื่อหลีกเลี่ยงปัญหาคอขวดที่อาจจะเกิดจากการคลี่ลูปและการทำไปป์ไลน์ลูปโดยค่า Factor ต่างๆ ในการคลี่ลูป การทำไปป์ไลน์ลูปและการแบ่งอาร์เรย์มีขีดจำกัดในเรื่องของทรัพยากรเป็นหลัก สำหรับอินเทอร์เฟซ Block level สำหรับอาร์เรย์สามารถใช้ได้ทั้ง ap_ctrl_none, ap_ctrl_hs และ ap_ctrl_chain ส่วนอินเทอร์เฟซ Port level พบว่าที่ชนิดข้อมูลเป็นอาร์เรย์ 1 มิติ ขาเข้าเหมาะสมกับชนิดของอินเทอร์เฟซ Port level ชนิด axis และ ap_hs และชนิดข้อมูลเป็นอาร์เรย์ 1 มิติ ทั้ง 2 พอร์ตที่เป็นข้อมูลขาออกเหมาะสมกับอินเทอร์เฟซ Port level ชนิด ap_memory และ bram เป็นคู่ตามลำดับ

บทที่ 4

ผลการทดลองและการวิเคราะห์

บทนี้นำเสนอผลการทดลองและการวิเคราะห์ผลการทดลอง โดยงานวิจัยชิ้นนี้เป็นการออกแบบและพัฒนาการทำระบบประมวลผลตรวจจับเลนถนนบนบอร์ดประมวลผลรุ่น Zybo z7-10 ตระกูล Zynq-7000 โดยใช้เทคนิคการเพิ่มประสิทธิภาพระบบที่ระดับสูง HLS งานวิจัยนี้ได้ทำการศึกษาและวิเคราะห์วิธีการหาขอบที่เหมาะสมกับการตรวจจับเลนถนนและวิธีการปรับใช้เทคนิคการเพิ่มประสิทธิภาพทั้งวิธีการกำหนดขนาดของอาร์เรย์ การคลี่ลู่ การไปป์ไลน์ลู่ การแบ่งอาร์เรย์และรวมถึงการจัดการอินเทอร์เฟซ HLS กับระบบตรวจจับเลนถนนที่ช่วยลดระยะเวลาในการประมวลผลมากที่สุด ในบทนี้จะนำเสนอรายละเอียดผลการทดลองและการวิเคราะห์ในแต่ละหัวข้อดังนี้

(4.1) ผลการทดลองและวิเคราะห์ระยะเวลาในการประมวลผลแต่ละขั้นตอนสำหรับการทำ ระบบตรวจจับเลนถนน

(4.2) ผลการทดลองและวิเคราะห์การใช้วิธีการหาขอบภาพต่างๆกับการทำระบบตรวจจับเลนถนน

(4.3) ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 ในการใช้เทคนิคการเพิ่มประสิทธิภาพระบบที่ระดับสูง HLS สำหรับ HA

(4.4) ผลการจำลองประสิทธิภาพเพิ่มเติมบน xczu9eg

4.1 ผลการทดลองและวิเคราะห์ระยะเวลาในการประมวลผลแต่ละขั้นตอนสำหรับระบบตรวจจับเลนถนน

จากขั้นตอนการออกแบบระบบตรวจจับเลนถนนซึ่งประกอบไปด้วย 4 ขั้นตอนคือ ขั้นตอนการเตรียมภาพ, ขั้นตอนการหาขอบภาพ, ขั้นตอนการหาเส้นตรงและขั้นตอนการคำนวณมุมของเส้นตรง บนอุปกรณ์ Intel® Core™ i7-7500U CPU ที่ความถี่ 2.70 GHz ที่ขนาดของภาพที่ป้อนเข้าระบบเท่ากับ 720×1280 พิกเซล ที่อัตราความถี่ของภาพที่เข้ามาใน 1 วินาทีหรือเฟรมเรทมีค่าเท่ากับ 24 FPS รายละเอียดการทำงานของแต่ละกระบวนการจะแสดงในรูปแบบของเวลาที่ใช้ซึ่งกระบวนการที่ใช้ในการเวลาในการประมวลผลมากที่สุดจะถูกนำไปวิเคราะห์และพัฒนาต่อไป จากผลการทดลองพบว่าขั้นตอนของการหาขอบภาพและขั้นตอนของการหาเส้นบนภาพใช้เวลาในการประมวลผลนานที่สุดดังตารางที่ 4 - 1

4.2 ผลการทดลองและวิเคราะห์การใช้วิธีการหาขอบภาพต่างๆของระบบตรวจจับเลนถนน

หลังจากขั้นตอนการทำโปรไฟล์พบว่าขั้นตอนการหาขอบภาพเป็นขั้นตอนหนึ่งที่ต้องจะดำเนินการบนส่วนฮาร์ดแวร์เพื่อเร่งความเร็ว ในการทดลองนี้จึงทำการเปรียบเทียบหาวิธีการของการหาขอบภาพที่เหมาะสมกับการตรวจจับเลนถนนมากที่สุดโดยจะมีการเปรียบเทียบระยะเวลาที่ใช้ในการประมวลผลสำหรับแต่ละวิธีการของการหาขอบภาพและค่าความถูกต้องของการตรวจจับเลนถนนเมื่อใช้วิธีการหาขอบภาพแต่ละวิธีการ

การทดสอบจะมีการใช้ไฟล์วิดีโอเป็นอินพุต ซึ่งสถานการณ์ที่ใช้ทดสอบจะประกอบไปด้วย 1. ขนระถยนต์วิ่งทางตรง 2. ขนระถยนต์วิ่งเข้าโค้งทางซ้าย และ 3. ขนระถยนต์วิ่งเข้าโค้งทางขวา โดยค่าความถูกต้องของการตรวจจับเลนถนนจะมีการเปรียบเทียบมุมที่วัดได้จากภาพถนนจริงเปรียบเทียบกับมุมที่วัดได้จากการใช้วิธีการตรวจจับเลนถนนดังภาพประกอบ 4. 1 โดยมุม θ_1 และ θ_3 เป็นมุมที่ได้จากวิธีการตรวจจับเลนถนน θ_2 และ θ_4 เป็นมุมที่ได้จากการวัดจริงจากภาพ ซึ่งในการทดลองนี้ค่าความแตกต่างของมุมบนภาพจริงกับมุมที่ได้จากการตรวจจับนั้นสามารถแตกต่างได้ ± 5 องศา ดังนั้นจากผลการทดลองดังตารางที่ 4 - 2 พบว่าวิธีการหาขอบภาพของ Robert เหมาะสมสำหรับการหาขอบภาพในการตรวจจับเลนถนนมากที่สุด เนื่องจากใช้ระยะเวลาในการประมวลผลน้อยที่สุดและให้ค่าความถูกต้องเป็นอันดับที่สองรองจากวิธีการหาขอบภาพของ Canny โดยค่าความถูกต้องแตกต่างกันเพียง 0.83 เปอร์เซนต์เท่านั้น

4.3 ผลการทดลองและวิเคราะห์ระยะเวลาในการใช้เทคนิคการเพิ่มประสิทธิภาพระบบที่ระดับสูง HLS สำหรับ HA

หลังทำการเลือกวิธีการที่เหมาะสมกับการหาขอบภาพแล้ว การพัฒนาการทำการตรวจจับเลนถนนถูกพัฒนาต่อโดยการดำเนินการบน Zybo z7-10 ขั้นตอนการหาขอบภาพและขั้นตอนการหาเส้นตรงจะถูกทำเป็น HA ในส่วนของ PL ส่วนขั้นตอนอื่นๆ จะถูกดำเนินการบนส่วน

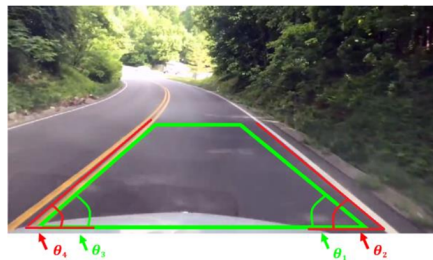
ตารางที่ 4 - 1 การเปรียบเทียบระยะเวลาในการประมวลผลของแต่ละกระบวนการ

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

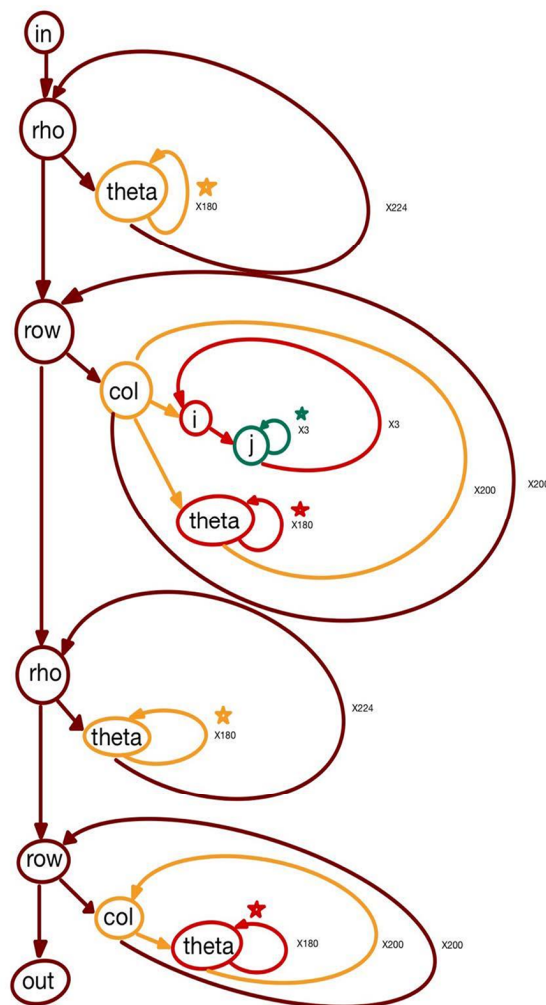
ของ PS เพื่อลดระยะเวลาในการประมวลผลของทั้งระบบ โดยในการทดลองจะเป็นการนำเอากระบวนการเพิ่มประสิทธิภาพดังหัวข้อ 3.4 มาปรับใช้กับอัลกอริทึมสำหรับทำ HA

4.3.1 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 ในการพิจารณาถึง การปรับขนาดของอาร์เรย์

จากผลการทดลองในตารางที่ 4 - 1 และตารางที่ 4 - 2 ขั้นตอนการหาขอบภาพและการหาเส้นบนภาพเป็นขั้นตอนที่ต้องมีการดำเนินการต่อบนอุปกรณ์ฮาร์ดแวร์โดยใช้วิธีการหาขอบภาพของ Robert สำหรับการหาขอบภาพและวิธีการ HT สำหรับการหาเส้นบนภาพ อย่างไรก็ตามทรัพยากรที่ใช้บนอุปกรณ์ฮาร์ดแวร์สำหรับการหาขอบภาพและการหาเส้นบนภาพนั้นเกินกว่าทรัพยากรที่มีรองรับบนอุปกรณ์ ดังนั้นจึงต้องทำการพิจารณากระบวนการของการปรับขนาดของอาร์เรย์ทั้งการพิจารณาถึงขนาดของ memory depth และ data width ที่จะส่งผลต่อการใช้หน่วยความจำ (Block RAM) เป็นหลักเพื่อลดและใช้จำนวนทรัพยากรให้คุ้มค่าและเพียงพอกับทรัพยากรบนฮาร์ดแวร์มากที่สุด โดยจำนวนของทรัพยากรที่ใช้ลดลงโดยเฉพาะ BRAM ดังตารางที่ 4 - 3



ภาพประกอบ 4. 1 การทดสอบความถูกต้องระบบตรวจจับเลนถนน



ภาพประกอบ 4. 2 ระดับรูปซ้อนรูปสำหรับทดสอบการคลี่รูปและการ ไปป์ไลน์รูป บน Zybo z7-10

เนื่องจากจำนวน memory depth และ data width ที่ถูกทำการปรับเปลี่ยนลดลงเป็นจำนวนมากทำให้จำนวน Block RAM ลดลงค่อนข้างมาก รวมไปถึงระยะเวลาที่ใช้ในการประมวลผลเช่นกัน

4.3.2 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 สำหรับขั้นตอนการวิเคราะห์รูป

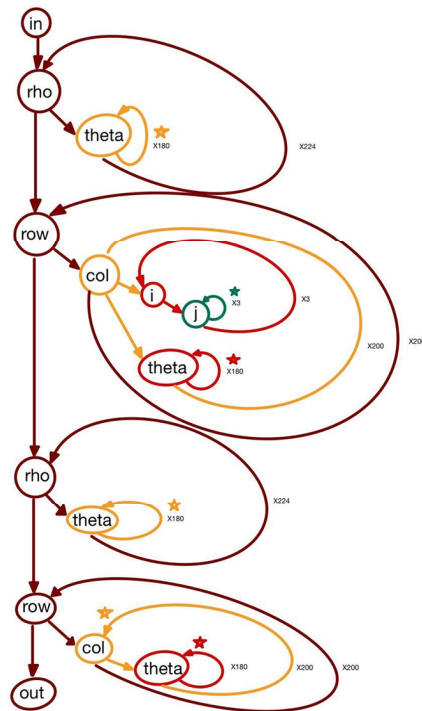
จากขั้นตอนการทำการวิเคราะห์รูปในกระบวนการเพิ่มประสิทธิภาพบน Vivado High-Level Synthesis จึงมีการทำการทดลอง โดยการปรับใช้วิธีการของการคลี่รูปและวิธีการไปป์ไลน์รูปกับอัลกอริทึมต่อจากการกำหนดขนาดของอาร์เรย์โดยในการทดลองนี้จะเลือก

ตารางที่ 4 - 3 การเปรียบเทียบระยะเวลาและจำนวนทรัพยากรที่ใช้ก่อนและหลังการทำการปรับขนาดของอาร์เรย์

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

ปรับใช้เทคนิคการคลี่ลู่และการไปป์ไลน์ลู่กับลู่ชั้นในสุดของแต่ละลู่ซ้อนลู่ดังภาพประกอบ 4. 2 (เครื่องหมาย * บ่งบอกถึงระดับชั้นของลู่ที่มีการคลี่ลู่หรือการไปป์ไลน์ลู่) เนื่องจากข้อจำกัดของทรัพยากรที่มีอยู่บน Zybo z7-10 โดยลู่ซ้อนลู่ที่ทดสอบมีด้วยกัน 4 ลู่หลัก โดยลู่ซ้อนลู่ที่ 1 และ 3 มีลักษณะการวนลู่ 2 ชั้น ลู่ซ้อนลู่ที่ 4 มีลักษณะการวนลู่ 3 ชั้น และลู่ซ้อนลู่ที่ 2 จะประกอบไปด้วย 2 ลู่ซ้อนลู่ย่อย ซึ่งลู่ซ้อนลู่ย่อยที่ 1 มีลักษณะการวนลู่ 3 ชั้น และลู่ซ้อนลู่ย่อยที่ 2 มีลักษณะการวนลู่ 2 ชั้นด้วยกัน

ในการทดลองนี้จะเป็นการเลือกพิจารณาวิธีการคลี่ลู่และไปป์ไลน์ลู่เพื่อปรับใช้ให้เหมาะสมกับการทำงานสำหรับแต่ละลู่ซ้อนลู่โดยจะมีการคำนึงถึงจำนวนเวลาซึ่งมีหน่วยเป็นจำนวนรอบนาฬิกาใช้ในการประมวลผลร่วมกับจำนวนทรัพยากรที่ใช้สำหรับทำ HA โดยทรัพยากรที่แสดงในผลการทดลองจะประกอบไปด้วย 1. DSP (ทรัพยากรบน FPGA ที่ช่วยในการคำนวณทางคณิตศาสตร์ เหมาะสำหรับการคำนวณ Digital signal processing: DSP โดยภายในจะประกอบไปด้วยวงจรวก คูณ และ Accumulator) 2. BRAM (BRAM หรือ Block RAM เป็นหน่วยความจำชนิดหนึ่ง) 3. LUT (LUT หรือ Look-up table เป็นทรัพยากรที่ใช้สำหรับสร้างฟังก์ชันการทำงานต่างๆบน CLBs) และ 4. FF (FF หรือ Flip-Flop มีหน้าที่รักษาหรือเปลี่ยนแปลงสถานะของเอาต์พุต)



ภาพประกอบ 4. 3 ระดับลูปซ้อนสำหรับทดสอบการคลี่ลูปและการไปป์ไลน์ลูปบน xczu9eg

ในกรณีที่พิจารณาถึงระยะเวลาที่ใช้ในการประมวลผลน้อยที่สุดดังตารางที่ 4 - 4 พบว่าการคลี่ลูปเป็นวิธีการที่เหมาะสมกับการทำงานในลูปซ้อนลูปลำดับที่ 1 3 และ 4 ส่วนการไปป์ไลน์ลูปเหมาะสำหรับการทำงานของลูปซ้อนลูปลำดับที่ 2 (ทั้งลูปซ้อนลูปย่อยของการหาขอบและการหาเส้นของภาพ) ถึงแม้ว่าการคลี่ลูปจะสามารถลดระยะเวลาประมวลผลได้ดีกว่าการไปป์ไลน์ลูปในลูปซ้อนลูปลำดับที่ 3 และ 4 แต่เมื่อพิจารณาถึงจำนวน CLBs ที่ต้องใช้พบว่าเกินกว่าจำนวนที่มีรองรับบนอุปกรณ์จริง ดังนั้นการไปป์ไลน์ลูปจึงมีความเหมาะสมที่จะทำงานร่วมในลูปซ้อนลูปลำดับที่ 3 และ 4 โดยวิธีการที่เสนอเป็นการรวมกันของวิธีการเพิ่มประสิทธิภาพที่เหมาะสมต่อลูปซ้อนลูปแต่ละลำดับลงในลูปซ้อนลูปนั้นๆ พบว่าที่ความถี่นาฬิกา 100 MHz จำนวนเวลาดลดลงจาก 103,709,185 รอบนาฬิกาเป็น 15,789,018 รอบนาฬิกาหรือลดลงประมาณ 6.57 ครั้ง ทั้งวิธีการคลี่ลูปและการไปป์ไลน์ลูปเป็นวิธีการเพิ่มประสิทธิภาพโดยจะคำนึงถึงการเพิ่มจำนวนความสามารถในการทำงาน (Throughput) เป็นหลัก ในกรณีของตัวแปร $hough1[\rho][\theta]$, $hough2[\rho][\theta]$ และอีกสองอาร์เรย์คือ $\sin[\theta]$ และ $\cos[\theta]$ ซึ่งหากตัวแปรอาร์เรย์ข้างต้นนี้ถูกเก็บอยู่บนหน่วยความจำบนฮาร์ดแวร์ตัวเดียวกันจะทำให้การทำงานของ

การคลี่รูปและการไปป์ไลน์รูปไม่สามารถทำงานได้เต็มประสิทธิภาพเนื่องจากขีดจำกัดของจำนวนครั้งในการเข้าถึงข้อมูลบนหน่วยความจำบนฮาร์ดแวร์ต่อ 1 รอบนาฬิกา

4.3.3 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 สำหรับวิธีการแบ่งอาร์เรย์

เนื่องจากตัวแปรอาร์เรย์ $\text{hough1}[\rho][\theta]$, $\text{hough2}[\rho][\theta]$ และอีกสองตัวแปรอาร์เรย์คือ $\sin[\theta]$ และ $\cos[\theta]$ ซึ่งเป็นตัวแปรที่ถูกกำหนดค่าคงที่ที่ถูกเก็บอยู่ในหน่วยความจำตัวเดียวกัน ทำให้การทำงานของกรคลี่รูปและการไปป์ไลน์รูปนั้นไม่สามารถทำงานได้เต็มประสิทธิภาพเนื่องจากขีดจำกัดของการเข้าถึงหน่วยความจำในแต่ละรอบนาฬิกาจึงทำการทดลองการทำการแบ่งอาร์เรย์โดยในการทดลองนี้จะทำการทดลองการแบ่งอาร์เรย์กับหน่วยความจำ Block RAM ซึ่งเก็บค่าของตัวแปร $\text{hough1}[\rho][\theta]$, $\text{hough2}[\rho][\theta]$, $\sin[\theta]$

ตารางที่ 4 - 4 การเปรียบเทียบวิธีการคลี่รูปและการไปป์ไลน์รูปในแต่ละรูป

Process		Method	Resource (%)				Latency (clock cycles)
			DSP	BRAM	CLB	FF	
Proposed method (1)			12	30	9	3	103,709,185
1 st		Unrolling (F=120)	12	30	27	3	98,847,177
		Pipelining (II=1)	12	30	10	3	98,860,618
2 nd	Edge detection	Unrolling (F=120)	13	30	9	3	97,891,440
		Pipelining (II=1)	13	30	10	3	98,012,241
	Line detection	Unrolling (F=120)	-	-	-	-	-
		Pipelining (II=1)	13	30	10	3	46,056,556
Edge + Line detection	Combination	15	30	10	3	45,046,530	
3 rd		Unrolling (F=120)	-	-	-	-	-
		Pipelining (II=1)	12	30	15	6	98,457,432
4 th		Unrolling (F=120)	-	-	-	-	-
		Pipelining (II=1)	12	30	11	3	74,828,791
Proposed method (2)			15	30	34	6	15,789,018

และ $\cos[\theta]$ โดยตัวแปร $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ จะมีการทดสอบการแบ่งอาร์เรย์แบบบล็อกและวงกลมที่มีมิติที่ 2 เท่านั้น เนื่องจากลำดับการเข้าถึงตัวแปรทั้งสองตัวนี้ขึ้นอยู่กับมิติที่ 2 คือค่า θ เป็นหลัก รวมไปถึงจำนวนอิลิเมนต์ทั้งมิติที่ 1 และมิติที่ 2 ไม่เป็นเลขฐานสอง การทำการแบ่งอาร์เรย์แบบสมมาตรส่งผลให้เกิดการใช้หน่วยความจำ Block RAM อย่างไม่คุ้มค่า และมากเกินไปจนมีรองรับบนฮาร์ดแวร์ ส่วนการทำการแบ่งอาร์เรย์สำหรับตัวแปรอาร์เรย์ $\sin[\theta]$ และ $\cos[\theta]$ จะมีการทดสอบทั้งแบบบล็อก วงกลมและสมมาตร

ผลการทดลองการทำการแบ่งอาร์เรย์เป็นไปตามตารางที่ 4 - 5 โดยทั้งตัวแปรอาร์เรย์ $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ เหมาะสมกับการทำการแบ่งอาร์เรย์ชนิดบล็อกที่มีมิติที่ 2 และค่า Factor มีค่าเท่ากับ 6 ผลจากการทดลองพบว่าทำการแบ่งอาร์เรย์สามารถลดจำนวนเวลาที่ใช้จาก 15,789,018 รอบนาฬิกาเหลือ 15,777,837 รอบนาฬิกาหรือประมาณ 1.001 เท่า ในความเป็นจริงการทำการแบ่งอาร์เรย์ชนิดสมมาตรสำหรับตัวแปรอาร์เรย์ $\text{hough1}[\rho][\theta]$ และ $\text{hough2}[\rho][\theta]$ สามารถทำได้และระยะเวลาในการทำงานของกระบวนการจะลดลง แต่ทรัพยากรที่ใช้มีจำนวนมาก หากมีการพัฒนาบนอุปกรณ์ขนาดใหญ่จะสามารถทำให้ระบบมีการทำงานที่รวดเร็วยิ่งขึ้นได้ แต่สำหรับตัวแปรอาร์เรย์ $\cos[\theta]$ และ $\sin[\theta]$ สามารถใช้วิธีการแบ่งอาร์เรย์ชนิดสมมาตรได้เนื่องจากอาร์เรย์มีจำนวนอิลิเมนต์ไม่มากนักสามารถทำงานบนอุปกรณ์ฮาร์ดแวร์ได้เพียงพอ ผลจากการทดลองเพิ่มการแบ่งอาร์เรย์กับทั้งสองตัวแปรอาร์เรย์พบว่าจำนวนเวลาที่ใช้ลดลงจาก 15,789,018 รอบนาฬิกาเหลือ 15,697,034 รอบนาฬิกา

4.3.4 ผลการทดลองและวิเคราะห์ระยะเวลาและทรัพยากรบน Zybo z7-10 ในการใช้อินเทอร์เฟส HLS

การทำ HA บน Zybo z7-10 มีการใช้ Vivado HLS ซึ่งบล็อก IP ที่เกิดจากการทำ HA จะมีการใช้อินเทอร์เฟส HLS ในการส่งและรับข้อมูลกับบล็อก IP ตัวอื่นๆ โดยบล็อก IP ที่สร้างจะประกอบไปด้วย 2 พอร์ตหลักคือ 1. อินเทอร์เฟส Block level และ 2. อินเทอร์เฟส Port level ในการทดลองนี้มีการสร้างบล็อก IP โดยอินพุตมีลักษณะเป็นอาร์เรย์ 1 มิติ ขนาด 8 บิต 200×200 อิลิเมนต์ และเอาต์พุตเป็นอาร์เรย์ 1 มิติ ขนาด 8 บิต 2 อิลิเมนต์ ซึ่งอินเทอร์เฟส Block level ที่สามารถรองรับชนิดของข้อมูลทั้งขาเข้าและออกประเภทอาร์เรย์ได้ดังตารางที่ 2 - 1 มี ap_ctrl_none , ap_ctrl_hs และ ap_ctrl_chain ส่วนอินเทอร์เฟส Port level ที่สามารถรองรับชนิดของข้อมูลทั้งขาเข้าและออกประเภทอาร์เรย์ได้มี axis , s_axilite , m_axi , ap_hs , ap_memory , bram , ap_fifo และ ap_bus ซึ่งในการทดลองนี้จะทำการจับคู่ทุกความเป็นไปได้ของอินเทอร์เฟส Block level และอินเทอร์เฟส Port level ดังภาพประกอบ 3.32 โดยผลการทดลองการจับคู่การใช้ อินเทอร์เฟส Block level และอินเทอร์เฟส Port level แสดงใน ตารางที่ 4 - 6 ตารางที่ 4 - 7 ตารางที่

4 - 8 ตารางที่ 4 - 9 ตารางที่ 4 - 10 และตารางที่ 4 - 11 และการเปรียบเทียบระยะเวลาการประมวลผลระหว่างระบบเมื่อมีการจัดการ อินเทอร์เน็ต HLS และค่าเริ่มต้นของอินเทอร์เน็ต HLS ถูกแสดงดังตารางที่ 4 - 12 ซึ่งจะมีการเปรียบเทียบกับ อินเทอร์เน็ต HLS เดิมที่ระบบมีการสร้างให้พบว่าการใช้อินเทอร์เน็ต Block level ให้ประสิทธิภาพเท่ากันทุกชนิด ในขณะที่คู่ของอินเทอร์เน็ต Port level สำหรับชนิดอินพุตและเอาต์พุตที่ส่งผลให้ประสิทธิภาพการทำงานของระบบดีที่สุดคือ axis-ap_memory, axis-bram, ap_hs-ap_memory และ ap_hs-bram

จากตารางที่ 4 - 13 เป็นการแสดงการเปรียบเทียบวิธีการเพิ่มประสิทธิภาพการทำงานในแต่ละวิธีการในหน่วยเฟรมต่อวินาที (FPS) โดยวิธีการที่นำเสนอที่ 1 (proposed 1) เกิดจากการทำการปรับขนาดของอาร์เรย์ วิธีการที่นำเสนอที่ 2 (proposed 2) เกิดจากการปรับใช้การคลี่ลูปและการไปป์ไลน์ลูปในอัลกอริทึม, วิธีการนำเสนอที่ 3 (proposed 3) เกิดจากการปรับใช้การแบ่งอาร์เรย์เพิ่มเติมต่อจากวิธีการนำเสนอที่ 2 เพื่อส่งเสริมการทำงานของกรคลี่ลูปและการไปป์ไลน์ลูปและวิธีการนำเสนอที่ 4 (proposed 4) เป็นการจัดการการใช้อินเทอร์เน็ต HLS ต่อจากวิธีการนำเสนอที่ 3 ให้เหมาะสมกับการใช้งาน โดยผลที่ได้จากการทำกระบวนการเพิ่มประสิทธิภาพของระบบตรวจจับเลนถนนอยู่ที่ประมาณ 6 เฟรม/วินาที ซึ่งไม่เพียงพอต่อการใช้งานจริงจึงมีการทำการทดลองเพิ่มเติมในหัวข้อ 4.3.5 เพื่อคูแวนโน้มการทำงานในระบบที่อุปกรณ์ขนาดใหญ่ขึ้น

4.3.5 ผลการจำลองประสิทธิภาพเพิ่มเติมบน xczu9eg

ในการทดลองนี้เป็นการจำลองวิธีการเพิ่มประสิทธิภาพบนของอัลกอริทึมเดิม แต่จะมีการปรับระดับของลูปที่มีการเพิ่มวิธีการคลี่ลูปหรือการไปป์ไลน์ลูปให้มีระดับที่สูงขึ้นดังภาพประกอบ 4. 3 เนื่องจากเป็นขั้นตอนที่สามารถปรับให้เหมาะสมกับอุปกรณ์นั้นได้ บน HLS เพื่อคูแวนโน้มของประสิทธิภาพการทำงานบนอุปกรณ์ที่มีทรัพยากรจำนวนเยอะขึ้นได้โดยที่วิธีการของการปรับขนาดของอาร์เรย์ การแบ่งอาร์เรย์และอินเทอร์เน็ต HLS ยังคงเหมือนเดิม ผลการทดลองที่ได้ถูกแสดงในตารางที่ 4 - 14 พบว่าประสิทธิภาพการประมวลผลบน xczu9eg เร็วกว่า Zybo z7-10 อยู่ 1.48 เท่า

ตารางที่ 4 - 5 การเปรียบเทียบระยะเวลาและทรัพยากรในการทำแบ่งอาร์เรย์

Method		Resource (%)				Latency (clock cycles)
		DSP	BRAM	CLB	FF	
Proposed method (2)		15	30	34	6	15,789,018
hough1 & hough2	Block type, Factor 2, dimension 2	15	30	27	6	15,782,316
	Block type, Factor 6, dimension 2	17	44	24	8	15,777,837
	Block type, Factor 12, dimension 2	17	44	26	9	15,803,597
	Cyclic type, Factor 2, dimension 2	13	30	389	55	23,414,035
	Cyclic type, Factor 6, dimension 2	15	44	387	56	23,409,564
	Cyclic type, Factor 12, dimension 2	13	30	389	55	23,414,035
sine, cosine	Complete	17	42	39	8	15,697,034
	Block type, Factor 2	13	29	397	59	13,429,011
	Block type, Factor 6	13	23	398	60	13,388,610
	Cyclic type, Factor 2	17	42	25	8	15,737,436
	Cyclic type, Factor 6	18	42	27	9	15,697,044
Proposed method (3)		17	42	39	8	15,697,034

ตารางที่ 4 - 6 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_none	axis (AXI4 interface)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40
	s_axilite (AXI4 interface)	axis (AXI4 interface)	16,017,421	69	15	8	39
		s_axilite (AXI4 interface)	20,537,435	72	16	9	41
		m_axi (AXI4 interface)	20,537,440	72	16	11	47
		ap_hs (wire handshake)	16,017,421	69	15	8	39
		ap_memory (memory interface)	15,737,435	69	16	8	40
		bram (memory interface)	15,737,435	69	16	8	40
		ap_fifo (memory interface)	16,017,421	69	15	8	39
		ap_bus	20,537,435	69	16	9	41
	m_axi (AXI4 interface)	axis (AXI4 interface)	16,300,228	44	15	10	43
		s_axilite (AXI4 interface)	20,820,242	47	16	10	45
		m_axi (AXI4 interface)	20,820,247	47	16	13	52
		ap_hs (wire handshake)	16,300,228	44	15	10	43
		ap_memory (memory interface)	16,020,242	44	16	10	44
		bram (memory interface)	16,020,242	44	16	10	44
		ap_fifo (memory interface)	16,300,228	44	15	10	43
		ap_bus	20,820,242	44	16	10	44
	ap_hs (wire handshake)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40

ตารางที่ 4 - 7 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_none	ap_memory (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	bram (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	ap_fifo (memory interface)	axis (AXI4 interface)	15,977,020	42	13	8	39
		s_axilite (AXI4 interface)	20,497,034	45	15	9	41
		m_axi (AXI4 interface)	20,497,039	45	15	11	47
		ap_hs (wire handshake)	15,977,020	42	13	8	39
		ap_memory (memory interface)	15,697,034	42	15	8	40
		bram (memory interface)	15,697,034	42	15	8	40
		ap_fifo (memory interface)	15,977,020	42	13	8	39
		ap_bus	20,497,034	42	15	8	40
	ap_bus	axis (AXI4 interface)	16,098,223	42	15	8	39
		s_axilite (AXI4 interface)	20,618,237	45	16	9	41
		m_axi (AXI4 interface)	20,618,242	45	16	11	47
		ap_hs (wire handshake)	16,098,223	42	15	8	39
		ap_memory (memory interface)	15,818,237	42	16	8	40
		bram (memory interface)	15,818,237	42	16	8	40
		ap_fifo (memory interface)	16,098,223	42	15	8	39
		ap_bus	20,618,237	42	16	8	40

ตารางที่ 4 - 8 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_hs	ap_memory (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	bram (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	ap_fifo (memory interface)	axis (AXI4 interface)	15,977,020	42	13	8	39
		s_axilite (AXI4 interface)	20,497,034	45	15	9	41
		m_axi (AXI4 interface)	20,497,039	45	15	11	47
		ap_hs (wire handshake)	15,977,020	42	13	8	39
		ap_memory (memory interface)	15,697,034	42	15	8	40
		bram (memory interface)	15,697,034	42	15	8	40
		ap_fifo (memory interface)	15,977,020	42	13	8	39
		ap_bus	20,497,034	42	15	8	40
	ap_bus	axis (AXI4 interface)	16,098,223	42	15	8	39
		s_axilite (AXI4 interface)	20,618,237	45	16	9	41
		m_axi (AXI4 interface)	20,618,242	45	16	11	47
		ap_hs (wire handshake)	16,098,223	42	15	8	39
ap_memory (memory interface)		15,818,237	42	16	8	40	
bram (memory interface)		15,818,237	42	16	8	40	
ap_fifo (memory interface)		16,098,223	42	15	8	39	
ap_bus		20,618,237	42	16	8	40	

ตารางที่ 4 - 9 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_hs	axis (AXI4 interface)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40
	s_axilite (AXI4 interface)	axis (AXI4 interface)	16,017,421	69	15	8	39
		s_axilite (AXI4 interface)	20,537,435	72	16	9	41
		m_axi (AXI4 interface)	20,537,440	72	16	11	47
		ap_hs (wire handshake)	16,017,421	69	15	8	39
		ap_memory (memory interface)	15,737,435	69	16	8	40
		bram (memory interface)	15,737,435	69	16	8	40
		ap_fifo (memory interface)	16,017,421	69	15	8	39
		ap_bus	20,537,435	69	16	9	41
	m_axi (AXI4 interface)	axis (AXI4 interface)	16,300,228	44	15	10	43
		s_axilite (AXI4 interface)	20,820,242	47	16	10	45
		m_axi (AXI4 interface)	20,820,247	47	16	13	52
		ap_hs (wire handshake)	16,300,228	44	15	10	43
		ap_memory (memory interface)	16,020,242	44	16	10	44
		bram (memory interface)	16,020,242	44	16	10	44
		ap_fifo (memory interface)	16,300,228	44	15	10	43
		ap_bus	20,820,242	44	16	10	44
	ap_hs (wire handshake)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40

ตารางที่ 4 - 10 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_chain	axis (AXI4 interface)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40
	s_axilite (AXI4 interface)	axis (AXI4 interface)	16,017,421	69	15	8	39
		s_axilite (AXI4 interface)	20,537,435	72	16	9	41
		m_axi (AXI4 interface)	20,537,440	72	16	11	47
		ap_hs (wire handshake)	16,017,421	69	15	8	39
		ap_memory (memory interface)	15,737,435	69	16	8	40
		bram (memory interface)	15,737,435	69	16	8	40
		ap_fifo (memory interface)	16,017,421	69	15	8	39
		ap_bus	20,537,435	69	16	9	41
	m_axi (AXI4 interface)	axis (AXI4 interface)	16,300,228	44	15	10	43
		s_axilite (AXI4 interface)	20,820,242	47	16	10	45
		m_axi (AXI4 interface)	20,820,247	47	16	13	52
		ap_hs (wire handshake)	16,300,228	44	15	10	43
		ap_memory (memory interface)	16,020,242	44	16	10	44
		bram (memory interface)	16,020,242	44	16	10	44
		ap_fifo (memory interface)	16,300,228	44	15	10	43
		ap_bus	20,820,242	44	16	10	44
	ap_hs (wire handshake)	axis (AXI4 interface)	15,936,619	42	13	8	39
		s_axilite (AXI4 interface)	20,456,633	45	15	9	41
		m_axi (AXI4 interface)	20,456,638	45	15	11	47
		ap_hs (wire handshake)	15,936,619	42	13	8	39
		ap_memory (memory interface)	15,656,633	42	15	8	40
		bram (memory interface)	15,656,633	42	15	8	40
		ap_fifo (memory interface)	15,936,619	42	13	8	39
		ap_bus	20,456,633	42	15	8	40

ตารางที่ 4 - 11 ผลการทดลองการเลือกใช้อินเทอร์เฟซ HLS

Block Level Interface	Port Level Interface		Latency (clock cycle)	Resource (%)			
	Input	Output		BRAM	DSP	FF	LUT
ap_ctrl_chain	ap_memory (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	bram (memory interface)	axis (AXI4 interface)	16,017,421	42	15	8	39
		s_axilite (AXI4 interface)	20,537,435	45	16	9	41
		m_axi (AXI4 interface)	20,537,440	45	16	11	47
		ap_hs (wire handshake)	16,017,421	42	15	8	39
		ap_memory (memory interface)	15,737,435	42	16	8	40
		bram (memory interface)	15,737,435	42	16	8	40
		ap_fifo (memory interface)	16,017,421	42	15	8	39
		ap_bus	20,537,435	42	16	8	40
	ap_fifo (memory interface)	axis (AXI4 interface)	15,977,020	42	13	8	39
		s_axilite (AXI4 interface)	20,497,034	45	15	9	41
		m_axi (AXI4 interface)	20,497,039	45	15	11	47
		ap_hs (wire handshake)	15,977,020	42	13	8	39
		ap_memory (memory interface)	15,697,034	42	15	8	40
		bram (memory interface)	15,697,034	42	15	8	40
		ap_fifo (memory interface)	15,977,020	42	13	8	39
		ap_bus	20,497,034	42	15	8	40
	ap_bus	axis (AXI4 interface)	16,098,223	42	15	8	39
		s_axilite (AXI4 interface)	20,618,237	45	16	9	41
		m_axi (AXI4 interface)	20,618,242	45	16	11	47
		ap_hs (wire handshake)	16,098,223	42	15	8	39
		ap_memory (memory interface)	15,818,237	42	16	8	40
		bram (memory interface)	15,818,237	42	16	8	40
		ap_fifo (memory interface)	16,098,223	42	15	8	39
		ap_bus	20,618,237	42	16	8	40

ตารางที่ 4 - 12 การเปรียบเทียบทรัพยากรและระยะเวลาในการจัดการอินเทอร์เฟซ HLS

Optimization	Resource (%)				Latency (clock cycles)
	DSP	BRAM	CLB	FF	
Default	17	42	39	8	15,697,034
การจัดการ HLS interface	16	42	39	8	15,656,633
Proposed method (4)	16	42	39	8	15,656,633

ตารางที่ 4 - 13 การเปรียบเทียบประสิทธิภาพการทำงานในแต่ละวิธีการในหน่วย FPS

Method	Latency (clock cycles)	Processing Performance (FPS)
Default	15,847,183,592	0.006
Proposed (1)	103,709,185	0.964
Proposed (2)	15,789,018	6.334
Proposed (3)	15,697,034	6.371
Proposed (4)	15,656,633	6.387

ตารางที่ 4 - 14 ผลการเปรียบเทียบระยะเวลาและทรัพยากรบน Zybo z7-10 และ xczu9eg

Optimization	Resource				Latency (clock cycles) clk 100 MHz	Processing Performance (FPS)
	DSP	BRAM	CLB	FF		
Zybo z7-10 (proposed method (4))	13	51	7,292	3,250	15,656,633	6.387
Xcзу9eg-ffvc900-2lv-e-EVAL	217	51	86,301	33,996	10,548,288	9.480

บทที่ 5

บทสรุป

บทนี้กล่าวถึงบทสรุปและข้อเสนอแนะหลังจากที่ได้ทำการศึกษาวิธีการออกแบบระบบตรวจจับเลนถนนและวิธีการเพิ่มประสิทธิภาพบนอุปกรณ์เอฟพีจีเอและทำการออกแบบและทดสอบวิธีการเพื่อประสิทธิภาพบนอุปกรณ์เอฟพีจีเอ (Zybo z7-10) ที่ระดับ HLS โดยประยุกต์ใช้กับระบบตรวจจับเลนถนนที่ได้ทำการออกแบบ

5.1 สรุป

วิทยานิพนธ์ฉบับนี้ได้นำเสนอวิธีการเพิ่มประสิทธิภาพบนอุปกรณ์เอฟพีจีเอที่ระดับ HLS สำหรับระบบตรวจจับเลนถนน โดยมีการออกแบบระบบตรวจจับเลนถนนและลำดับการเพิ่มประสิทธิภาพบนเอฟพีจีเอที่ระดับ HLS ให้เหมาะสมกับอัลกอริทึมซึ่งสามารถสรุปได้ดังนี้

สำหรับการออกแบบระบบตรวจจับเลนถนนนั้นมีขั้นตอนการทำงานทั้งหมด 4 ขั้นตอนด้วยกันคือ 1. ขั้นตอนการเตรียมภาพ 2. ขั้นตอนการหาขอบภาพ 3. ขั้นตอนการหาเส้นตรงบนภาพ และ 4. ขั้นตอนการคำนวณมุมของเส้นตรงบนภาพ โดยในขั้นตอนการหาขอบภาพจะมีการใช้วิธีการหาขอบภาพของ Robert ซึ่งเป็นวิธีการที่ให้ผลลัพธ์ในการตรวจจับเลนถนนดีที่สุด ขั้นตอนการหาเส้นตรงบนภาพจะใช้วิธีการของ HT และทำการคำนวณมุมของเส้นตรงที่ได้จากวิธีการ HT สำหรับการทำงานเส้นทางโค้งข้างหน้าต่อไป

สำหรับการออกแบบวิธีการเพิ่มประสิทธิภาพการทำงานบนอุปกรณ์เอฟพีจีเอที่ระดับ HLS กับระบบตรวจจับเลนถนนซึ่งมีลักษณะของอัลกอริทึมเป็นลูปซ้อนลูปที่ค่อนข้างซับซ้อน โดยวิธีการเพิ่มประสิทธิภาพมีขั้นตอนทั้งหมด 4 ขั้นตอนด้วยกันคือ 1. ขั้นตอนการกำหนดขนาดของอาร์เรย์ ในขั้นตอนนี้เป็นขั้นตอนที่ทำให้เกิดการใช้ทรัพยากรบนเอฟพีจีเออย่างคุ้มค่าที่สุดที่สุด 2. ขั้นตอนการวิเคราะห์ลูป ขั้นตอนนี้มีการเพิ่มวิธีการการคลี่ลูปและการไปป์ไลน์ลูปตามลักษณะการทำงานของลูปซึ่งช่วยเพิ่มความเร็วในการประมวลผลของระบบ โดยค่าตัวแปรสำหรับขั้นตอนนี้สามารถเปลี่ยนแปลงได้ขึ้นอยู่กับอุปกรณ์เอฟพีจีเอที่ดำเนินการ 3. ขั้นตอนการแบ่งอาร์เรย์ ขั้นตอนนี้เป็นขั้นตอนเพิ่มความเร็วในการประมวลผลซึ่งเป็นขั้นตอนที่ส่งเสริมการทำงานของการคลี่ลูปและการไปป์ไลน์ลูปและ 4. ขั้นตอนการจัดการอินเทอร์เฟส HLS ขั้นตอนนี้เป็น

ขั้นตอนที่ช่วยเพิ่มความเร็วในการประมวลผลเช่นกัน โดยผู้ใช้งานสามารถเลือกให้อินเทอร์เฟซ HLS ได้ตามลักษณะของข้อมูล

ดังนั้นสามารถสรุปได้ว่าการออกแบบระบบตรวจจับเลนถนนสามารถทำนายลักษณะเส้นทางโค้งข้างหน้าได้และวิธีการเพิ่มประสิทธิภาพบนอุปกรณ์เอพฟิเจอีที่ระดับ HLS กับระบบตรวจจับเลนถนนสามารถนำไปประยุกต์กับอัลกอริทึมที่มีลักษณะเป็นลูปซ้อนลูปที่ซับซ้อนได้ ทำให้การทำงานของระบบสามารถทำงานได้อย่างมีประสิทธิภาพมากขึ้น

5.2 ปัญหา

5.2.1 ระบบตรวจจับเลนถนนไม่สามารถตรวจจับได้เมื่อแสงสว่างบนถนนมีการเปลี่ยนแปลงเฉียบพลัน

5.2.2 การทดสอบบนอุปกรณ์เอพฟิเจอีใช้เวลานาน

5.2.3 วิธีการเพิ่มประสิทธิภาพบนอุปกรณ์เอพฟิเจอีไม่สามารถใช้ร่วมกับไลบรารี OpenCV ได้

5.3 ข้อเสนอแนะ

5.3.1 ระบบตรวจจับเลนถนนสามารถทำการปรับปรุงให้สามารถตรวจจับเลนถนนได้ถูกต้องมากขึ้น

5.3.2 วิธีการเพิ่มประสิทธิภาพสามารถทำการพัฒนาและปรับเปลี่ยนเพิ่มเติมได้เมื่อมีการดำเนินการบนอุปกรณ์ขนาดใหญ่ขึ้น

บรรณานุกรม

- [1] Fu-ming Xiao, Dong-sheng Li, Gao-ming Du, Yu-kun Song, Duo-li Zhang and Ming-lun Gao, “Design of AXI bus based MPSoC on FPGA,” Proceedings of 3rd International Conference on Anti-counterfeiting, Security and Identification in Communication, Aug 2009.
- [2] João Silva, Valery Sklyarov and Iouliia Skliarova, “Comparison of On-chip Communications in Zynq-7000 All Programmable Systems-on-Chip,” Proceedings of IEEE Embedded Systems Letters, pp. 31-34, Feb 2015.
- [3] Valery Sklyarov, Iouliia Skliarova, João Silva and Alexander Sudnitson, “Analysis and Comparison of Attainable Hardware Acceleration in All Programmable Systems-on-Chip,” Proceedings of Euromicro Conference on Digital System Design, Aug 2015.
- [4] Mehdi Feniche and Tomader Mazri, “Lane Detection and Tracking for Intelligent Vehicles: A Survey,” Proceedings of International Conference of Computer Science and Renewable Energies, Jul 2019.
- [5] Tin Trung Duong, Cuong Cao Pham, Tai Huu-Phuong Tran, Tien Phuoc Nguyen and Jae Wook Jeon, “Near Real-time Ego-Lane Detection in Highway and Urban Streets,” Proceedings of IEEE International Conference on Consumer Electronics-Asia, Oct 2016.
- [6] Deok-Kwon Lee, Ju-Seok Shin, Je-Han Jung, Sang-Jun Park, Se-Jin Oh and In-Soo Lee, “Real-time Lane Detection and Tracking System using Simple Filter and Kalman Filter,” Proceedings of 9th International Conference on Ubiquitous and Future Networks, Jul 2017.
- [7] Panadda Solod, Kiattisak Sengchuai, Apidet Booranawong, Pakpoom Hoyingcharoen, Surachate Chumpol, Masami Ikura and Nattha Jindapeth, “Model Based Design Approach for Road Lane Tracking,” Proceedings of Asia Pacific Conference on Robot IoT System Development and Platform 2018, Oct 2018.

- [8] Piyatap Promrit and Wannarat Suntiamorntut, “Design and Development of Lane Detection Based on FPGA,” Proceedings of 14th International Joint Conference on Computer Science and Software Engineering, Jul 2017.
- [9] Seokha Hwang and Youngjoo Lee, “FPGA-based Real-time Lane Detection for Advanced Driver Assistance Systems,” Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, Oct 2016.
- [10] Chanon Khongprasongsiri, Pinit Kumhom, Watcharapan Suwansantisuk, Teerasak Chotikawanid, Surachate Chumpol and Masami Ikura, “A Hardware Implementation for Real-time Lane Detection using High-Level Synthesis,” Proceedings of International Workshop on Advanced Image Technology, Jan 2018.
- [11] Xianwen Wei, Zhaojin Zhang, Zongjun Chai and Wei Feng, “Research on Lane Detection and Tracking Algorithm Based on Improved Hough Transform,” Proceedings of International Conference of Intelligent Robotic and Control Engineering, Aug 2018.
- [12] Digilent, SDSoC platforms for Digilent Zynq boards, [ออนไลน์], เข้าถึงได้จาก : https://github.com/Digilent/SDSoC-platforms?_ga=2.148487038.873870348.1596554937-668227463.1582683709

ภาคผนวก 1

Asia Pacific Conference on Robot IoT System Development and Platform 2018

Model Based Design Approach for Road Lane Tracking

PANADDA SOLOD^{†1} KIATTISAK SENGCHUAI^{†1} APIDET BOORANAWONG^{†1}
 PAKPOOM HOYINGCHAROEN^{†1} SURACHATE CHUMPOL^{‡2}
 MASAMI IKURA^{†2} NATTHA JINDAPETCH^{†1}

Abstract. This paper present the optimal method of edge detection applied to road lane tracking has been studied by Model Based Design (MBD) approach. We focus on the processing time and the accuracy when using different edge detection methods including Prewitt, Robert, Sobel and Canny edge detection. The video input file has size 720x1280 pixels and frame rate 24 fps. All processes were run on Intel® Core™ i7-7500U CPU @ 2.70 GHz. From the experiment result, Canny edge detection took longest time to do processing but gained the highest accuracy. Meanwhile Robert edge detection had a high accuracy similar to Canny edge detection and took least time for processing. Robert edge detection is an alternative method that is suitable for applying to road lane tracking.

Keywords: Road lane tracking, Edge Detection, Hough Transform, Model Based Design (MBD).

1. Introduction

Nowadays Advanced Driver-Assistance Systems (ADAS) are developed to enhance the system of the vehicle safety and better driving. Road lane tracking is one of ADAS that can be used to avoid accident on road. The main issue of doing road lane detection is the speed and accuracy.

Algorithms for road lane tracking have been proposed [1] [2] and [3]. Two important steps for detecting road lane are edge detection and line detection. Mostly, Hough line transform is used for line detection but there are many methods to do edge detection such as Prewitt, Sobel, Robert and Canny edge detection [4], [5]. The complexity of each edge detection method is different. Therefore, the accuracy of the result and the processing time of doing lane detection with using different edge detection methods must be different.

In this paper we present the accuracy and the processing time of doing road lane tracking with different edge detection methods using Model Base Design (MBD).

2. Theory

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

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

(a)

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

(b)

Figure 1 Gradient value of Prewitt Edge Detection

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

(a)

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

(b)

Figure 2 Gradient value of Sobel Edge Detection

1	0
0	-1

(a)

0	1
-1	0

(b)

Figure 3 Gradient value of Robert Edge Detection

Meanwhile Canny Edge Detection also uses the gradient method, but more complexity of step than the others. 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 formula in (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.

$$\theta = \tan^{-1}(G_x/G_y) \tag{1}$$

3. Methodology

The methodology for doing road lane tracking can be separated into four steps as shown in Figure 4. The first step is pre-processing. In pre-processing step includes doing Region of Interest (ROI), converting RGB image to gray scale image, converting greyscale image to binary image and reducing noise (median filtering). The second step is edge detection. There are many methods for edge detection. In this paper, we focus on

^{†1} Department of Electrical Engineering, Prince of Songkla University.
^{‡2} Toyota Tsusho Nexty Electronics (Thailand) co., Ltd.

Prewitt, Robert, Sobel and Canny edge detection. The third step is line detection by using Hough transform. The final step is decision and drawing line on the road lane.

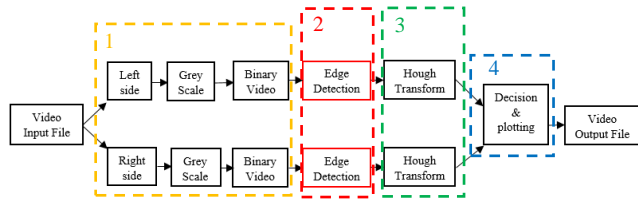


Figure 4. Block diagram of road lane tracking System

4. Experiment and Results

In the experiment, we compared the processing time of road lane tracking with different edge detection methods, which are Prewitt, Robert, Sobel and Canny edge detection by running the road lane tracking method as shown in Figure 4 on Intel® Core™ i7-7500U CPU @ 2.70 GHz. The total time of input file video was 60 second with 24 fps. From Figure 5, we compared the accuracy in each method of edge detection by comparing the angle of the actual road lane (θ_2 and θ_4) with the angle of the road lane tracking (θ_1 and θ_3) belong to the step in Figure 4.

For correct condition of the actual angle and the tracking angle, we allow the different between the actual angle and the tracking angle can vary by ± 5 degrees. In case of road curve in Figure 6, the actual angle (θ_2) is equal to 32 degrees and the tracking angle (θ_1) is equal to 37 degrees, but the tracking area still correct.

From Table 1, we found that Robert edge detection took least time compared to Prewitt, Sobel and Canny edge detection whereas Canny edge detection took longest time to do processing but gained the highest accuracy.

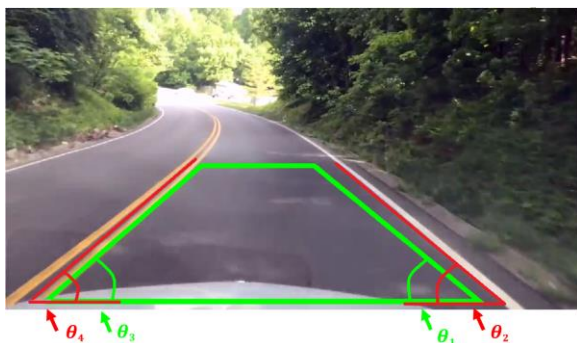


Figure 5. Comparison of actual angle with tracking angle

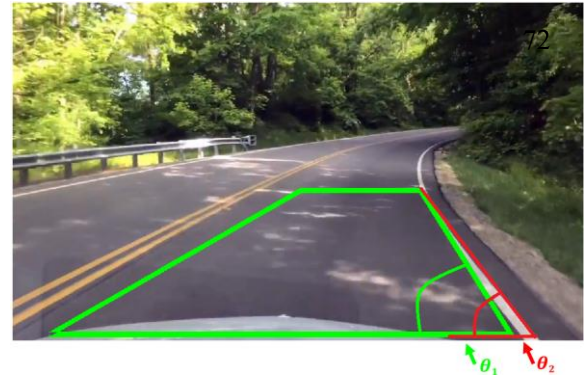


Figure 6. Comparison of actual angle with tracking angle

Table 1 Processing time and accuracy of edge detection methods

Edge detection method	Processing Time (s)	Accuracy (%)
Prewitt	180.23	86.50
Robert	174.17	89.17
Sobel	183.03	85.00
Canny	237.63	90.00

5. Conclusion

This paper has presented the comparison of each edge detection method applied to road lane tracking. We found that Robert edge detection took least time 174.17 second compared to Prewitt, Sobel and Canny edge detection and Canny edge detection took longest time 237.63 second to do processing but gained 90 percent of the highest accuracy. Even Robert edge detection does not have the highest accuracy, but the accuracy is the most similar to Canny edge detection and consumed the lowest processing time. Therefore, Robert edge detection is an alternative edge detection method for doing road lane tracking.

Reference

- [1] W. Phueakjeen, N. Jindapetch, L. Kuburat, and N. Suvanvorn, "A study of the edge detection for road lane," in The 8th Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference 2011, 2011, pp. 995–998.
- [2] P. Promrit and W. Suntiamorntut, "Design and development of lane detection based on FPGA," in 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017, pp. 1–4.
- [3] S. Dong and C. Peng, "A lane detection method based on track management approach," in 2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014, pp. 1–6.
- [4] "Canny edge detector". https://en.wikipedia.org/wiki/Canny_edge_detector, (accessed 2018-09-28).
- [5] "Hough transform". https://en.wikipedia.org/wiki/Hough_transform, (accessed 2018-08-29)

ภาคผนวก 2

2019 4th IEEE International Circuits and Systems Symposium (ICSS)

Memory Optimization for Accelerating Hough Transform on FPGA using High Level Synthesis

74

Panadda Solod
Department of Electrical
Engineering, Prince of
Songkla University
Songkhla, Thailand
panadda.solod@gmail.com

Nattha Jindapetch
Department of Electrical
Engineering, Prince of
Songkla University
Songkhla, Thailand
nattha.s@psu.ac.th

Kiattisak Sengchuai
Department of Electrical
Engineering, Prince of
Songkla University
Songkhla, Thailand
ak.kiattisak@hotmail.com

Apidet Booranawong
Department of Electrical
Engineering, Prince of
Songkla University
Songkhla, Thailand
bapidet@eng.psu.ac.th

Pakpoom Hoyingcharoen
Department of Electrical
Engineering, Prince of
Songkla University
Songkhla, Thailand
hpakpoom@eng.psu.ac.th

Surachate Chumpol
Toyota Tsusho Nexty
Electronics (Thailand)co.,
ltd
Bangkok, Thailand
surachate@th.nexty-
ele.com

Masami Ikura
Toyota Tsusho Nexty
Electronics (Thailand)co.,
ltd
Bangkok, Thailand
ikura@th.nexty-ele.com

Abstract—In this work, we present memory optimization in Hough transform on hardware of programmable logic (PL) part of FPGAs by using array partitioning block, cyclic and complete types at different dimension and different factor on the Xilinx Vivado High Level Synthesis tool. Loop analysis was presented to extract nested loops and suitably manage the array. The experimental results show that adding array partitioning complete type at dimension 2 can be the best type to improve speed of operating time and reduce the number of resources on Zybo ZC7010-1.

Keywords—Array Partitioning, High Level Synthesis (HLS), FPGA, Hough transform

I. INTRODUCTION (HEADING 1)

HLS (High Level Synthesis) is an effective tool to accelerate image processing on FPGA by enabling C/C++ languages and optimization techniques such as loop pipelining and loop unrolling. However, most image processing applications consist of nested loop and need a large amount of memory to store data, especially line detection and circle detection. Therefore, a more precise memory optimization is necessary to reduce operating time and resource usage.

There are many methods for doing line detection such as Hough transform, convolution based technique and dot plot [6]. Hough transform is one of famed methods in straight line detection. It is a feature extraction, which got high accuracy but takes long time for processing and needs a lot of memory. Hough transform improved the performance of operating time by probabilistic Hough transform [2] and was reduced resource usage by Hough transform utilizing improved voting scheme [3]. Directional wavelet transform base on Hough transform and morphology base on Hough transform are proposed in [5] to reduce both of operating time and resource usage. All of these works improved performance on part of processing system.

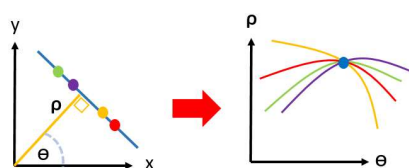


Fig. 1 Hough Transform

The alternative to develop Hough transform algorithm to get the higher speed is developing on hardware of programmable logic (PL) part of FPGAs. However, the biggest issue of developing on FPGAs is limitation of resource usage. One more issue of using FPGAs is hardware description language (HDL) such as Verilog language and VHDL because at present the developers are not familiar with HDL. Moreover, developing on FPGAs without analysis will cost a lot of resources. In some case, the speed of operating time will not increase as well. In [4], parallel accessing of multiple dimension array was demonstrated and applied to convolution technique of edge detection. In [1], parallel data access via data reuse is applied to image processing application to increase speed of operation time and reduce resource overhead.

As above mentioned, we found that Hough transform is one of famed method for applying to lane detection but process of Hough Transform is quite complicated, therefore it takes long time to operate and needs large memory to store data due to some data cannot reuse.

In this work, we aim to develop Hough transform method in PL part on Zybo ZC7010-1 with C language on Xilinx Vivado High Level Synthesis (HLS), which is expedient for developing by parallel computing to increase speed of

```
// Initialize Hough array
for (rho=0; rho<2048; rho++)
  for (theta=0; theta<41; theta++)
    hough[rho][theta] = 0;

// Counting hough[rho][theta]
for (row=0; row<rows; row++)
  for (col=0; col<cols; col++)
    if (in_img[row][col] == 255)
      for (theta=70; theta<120; theta++) {
        rho = row*cos(theta) + col*sin(theta);
        if (rho>0 && rho<2048) {
          hough[rho][theta]++;
        }
      }
}

// Straight line decision
for (row=0; row<nrows; row++)
  for (col=0; col<ncols; col++) {
    out_img[row][col] = 0;
    for (theta=-90; theta<180; theta++) {
      rho = row*cos(theta) + col*sin(theta);
      if (rho>0 && rho<2048 && hough[rho][theta]>thresh)
        out_img[x][y] = 255;
    }
  }
}
```

Fig.2 C language for Hough transform

operating time and memory management to reduce the number of memory compared to traditional algorithms to divisibly implement with the device.

II. THEORY

A. Hough transform

Hough transform is a feature extraction to identify lines, circles, ellipses, etc. This technique finds imperfect instances of objects within a certain class of shapes by a voting procedure. In general, the straight line in (1) can be represent by two parameters (slope, intercept) as a point (c, m).

$$y = mx + c \quad (1)$$

$$\rho = x \cos \theta + y \sin \theta \quad (2)$$

$$hough[\rho][\theta] = hough[\rho][\theta] + 1 \quad (3)$$

The polar form of a line is represented as (2). From Fig. 1 ρ represents the perpendicular distance of the line from the origin in pixels and θ is angle measured between ρ and horizontal axis. Every point of a straight line (blue line in Fig.1) on x-y space corresponds to each curve on θ - ρ space. In the same straight line, every corresponding curve on θ - ρ space will have the same intersection point. Thus, the number of intersection curves represents to length of the straight line.

From (1), (2) and (3) Hough transform can developed on C language following to steps in Fig. 2. To figure out the intersection point on θ - ρ space. $hough[\rho][\theta]$ in (3) is defined to represent 2-dimension array, which collects intersection times in each θ and ρ . Initially, $hough[\rho][\theta]$ is set to zero for every θ and ρ to store the number of times of the intersection point. Then, the value of ρ in every pixel is calculated by (2) to indicate position of array for every θ . Therefore, a 3-layer nested loop (row, column and θ) was created. After intersection point counting has finished, the straight line is decided by comparing value of $hough[\rho][\theta]$ with threshold value.

B. Array Partitioning

Array partition is an area optimization technique from Xilinx Vivado HLS. It separates an array into multiple smaller arrays and stores into multiple banks. There are three types of array partitioning shown in Fig. 3. The first type is block type, the original array is split into equally sized block of contiguous elements, the second type is cyclic, the original array is split into equally sized block interleaving the elements and the last type is complete, the original array is split into individual elements.

C. Pipelining

Pipelining is performance optimization technique. It reduces the initiation interval of a function or loop. Fig. 4 is an example of adding loop pipelining. Inside loop, there are 3 operations without adding loop pipelining. The first operation in the second round can operate until last operation in the first round finished, whereas in case of adding loop pipelining with initiation interval equal to one, the first operation in the second round can operate in next clock cycle.

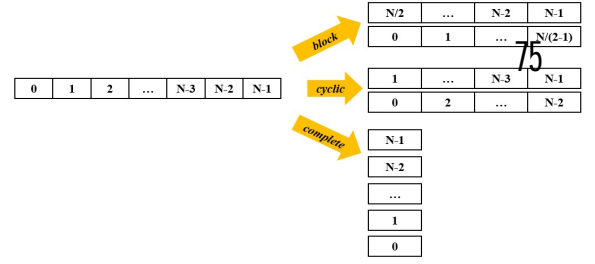


Fig. 3 Array Partitioning

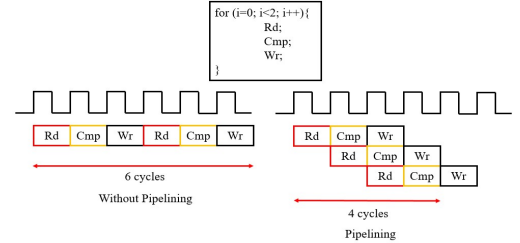


Fig. 4 Loop pipelining

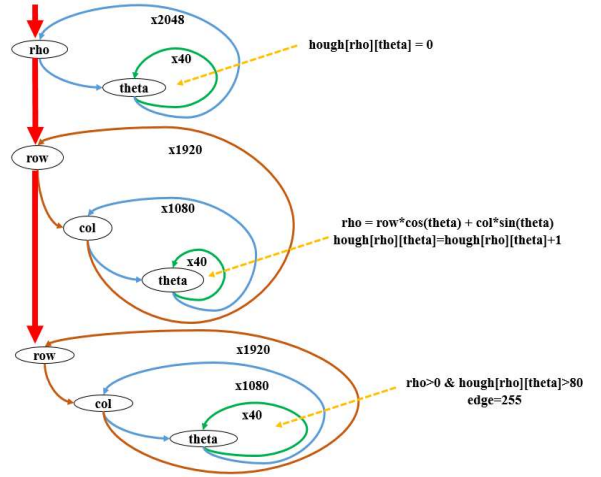


Fig. 5 Hough transform nested loop.

III. MEMORY OPTIMIZATION IN HIGH LEVEL SYNTHESIS

Vivado HLS accelerates design implementation by enabling C/C++ languages and allows optimization technique such as array partitioning, loop pipelining or loop unrolling etc. In this topic, a loop analysis is presented to analyze nested loop and suitably manage the array. This array is divisibly handled with size of block RAM.

A. Loop analysis

From the theory of Hough transform in C language in Fig. 2, we can summarize line detection using Hough transform following to Fig. 5 into three main steps. First step is started with 2 layers nested loop to initialize value of 2-dimension array with 16 bits ($hough[\rho][\theta]$) equal to zero to collect data. The next step is counting number of time of intersection in θ and ρ space for every pixel and the final step is straight line decision. From all steps of Hough transform in C language, we found that every step consists of nested loop, whatever 2-layer or 3-layer of nested loops, which cause speed reduction of operating time. In this work, we particularly consider in the second step due to (3) decreases

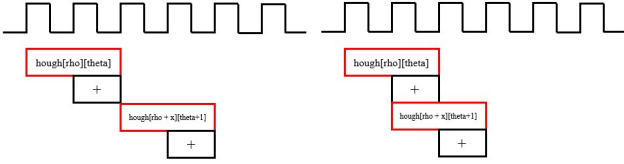


Fig. 6 Adding pipelining

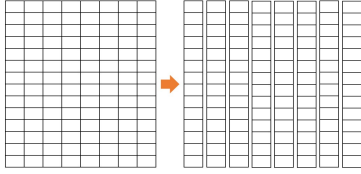


Fig. 7 Array partitioning of hough [rho] [theta]

speed of operating time. Fig.6 on left hand side demonstrates the traditional array access scheduling of (3), which can be reduce the speed of scheduling by adding loop pipelining with initiation interval (II) equal to one. On right hand side of Fig. 6 demonstrate the array access scheduling after adding loop pipelining. The total number of clock cycle in each loop is reduced. However, in case of array hough [rho] [theta] is stored in one bank of block RAM, bottleneck will be occurred because each block RAM can be read only one element at that time. Therefore, the second element (hough [rho + x] [theta + 1]) of array hough [rho] [theta] have to wait until the first element (hough [rho] [theta]) finish reading. From the bottleneck issue, we found that this issue can resolve by adding array partitioning technique to split array hough [rho] [theta] into individual banks. By splitting theta, 41 banks of block RAM are created according to the number of theta.

B. Array sizing

Array sizing must be first point to consider. Analyze and compute size of array to fit the requirement of array sizes with block memory storage capability to reduce over allocation number of block RAM by consider the number of block RAM with (4). For example, we consider to 1-dimension array 16 bits with size is equal to 2050 ($y [2050]$). Memory depth of this array is equal to 2050. From the condition of memory depth, this array needs sufficient memory depth is equal to 4096 (2^{12}), data width is equal to 16 bits and block RAM size is equal to 18K bits, according to (4) we need to allocate 4 (2^2) block RAM 18K to store an array. In case of reduce array size from 2050 to 2048, number of block RAM is reduced to 2 (2^1).

In this work we apply line detection using Hough transform to do lane detection with image size 1920x1080 the actual value of ρ is equal to 2203 and θ should equal to 180.

Here, we reduce ρ from 2203 to 2048 for block RAM 18K fit allocation and reduce θ from 180 to 41 because lane detection does not need to calculate from 0° to 180° .

$$\text{block RAM} = \frac{\text{memory depth} \times \text{data width}}{\text{block RAM size}} \quad (4)$$

C. Array Factor and Dimension

Configuration of array factor and dimension of array partitioning is also important to consider to increase speed of operating time and reduce the number of block RAM. Factor is representing to the number of bank (smaller array) of block RAM that is split from 1 bank of block RAM. Dimension is

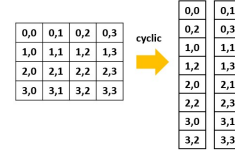


Fig. 8 Cyclic type with 2 factor at dimension 2

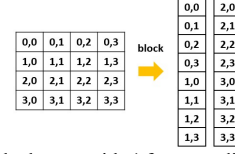


Fig. 9 Block type with 4 factor at dimension 2

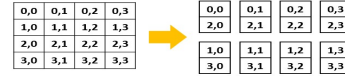


Fig. 10 Combination between complete type at dimension 2 and block type with 2 factor at dimension 1

representing which dimension of multi-dimension array to partition. Dimension is specified as an integer from 0 to N, for array with N dimension. The value of factor will affect to array partition block type and cyclic type only. For example, Fig. 8 is a 2-dimension array with size 4x4. This array is added array partitioning with 2 factor at dimension 2, 1 bank of block RAM is split into 2 banks of block RAM and sequence of each element is according to the figure. Fig. 9 has the same condition of factor and dimension as Fig. 7. The different of Fig. 8 and Fig. 9 is type of array partitioning, Fig. 9 is an array partitioning block type. The sequence of each element in array will different. Fig. 10 is a combination between array partitioning complete type and block type at different dimension. This figure presents adding array partitioning complete type at dimension 2 and array partitioning block type at dimension 1.

Factor and dimension of array partitioning is flexible. Both values can vary depend on size of array and requirements of memory access sequence.

D. Array Partitioning for Loop Pipelining

Array partitioning is an optimization technique, which can reduce the number of block RAM resources. Since some case array needs memory capacity less than allocated due to configure mode of block RAM.

Xilinx Vivado HLS configures mode of memory depth of 2^n , n is the number of bit and the number of memory allocation (block RAM) is equal to 2^n , n is the number of bit.

In this part, array partitioning complete type at dimension 2 is added to reduce number of block RAM resources following to Fig. 7. The original array on left side store in 1 bank of memory. After array partitioning has been added, this array is split into individual banks. Thus, 46 banks are created belong to number of theta with array width equal to 2048.

The number of block RAM resources before adding array partitioning memory depth is equal to 83,968 (2048×41). Therefore, memory depth allocated is 132,072 (2^{17}) due to configure of memory depth mode and data width is equal to 16, and capacity of block RAM should equal or more than 2,113,152 bits. The number of block RAM 18K equal to 256 due to the number of memory allocation mode. After array partitioning has been added, the memory depth in each bank is equal to 2048 with 16 memory width, and block RAM

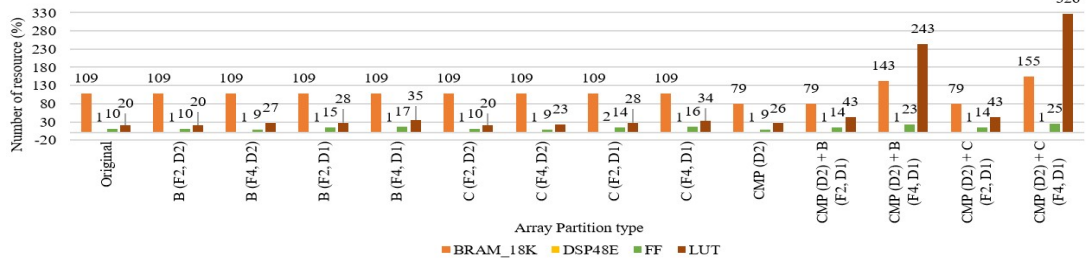


Fig. 11 Comparison of memory management

TABLE I. COMPARISON OF OPERATING TIME

Array Partition type	Operating time	
	Clock cycles	Times
Original	1,080,992,562	1.0000
Block (F2, D2)	1,081,086,770	1.0001
Block (F4, D2)	1,267,893,132	1.1729
Block (F2, D1)	995,987,918	0.9214
Block (F4, D1)	1,091,373,518	1.0096
Cyclic (F2, D2)	1,080,992,562	1.0000
Cyclic (F4, D2)	1,159,859,535	1.0730
Cyclic (F2, D1)	1,091,373,518	1.0096
Cyclic (F4, D1)	1,091,373,518	1.0096
Complete (D2)	894,234,428	0.8272
Complete (D2) + Block (F2, D1)	904,617,433	0.8368
Complete (D2) + Block (F4, D1)	1,000,003,033	0.9251
Complete (D2) + Cyclic (F2, D1)	904,617,433	0.8368
Complete (D2) + Cyclic (F4, D1)	1,000,003,033	0.9251

capacity should equal or more than 32,768 bits. Therefore, the number of block RAM 18K equal to 2 for each bank, and the total number of block RAM is 92.

Array partitioning techniques not only reduce number of block RAM, but the bottleneck of from loop analysis can also be resolve as well because first element and second element are store in different bank of block RAM resource, thus both element can be access in the same cycle.

IV. EXPERIMENT & DISCUSSION

In this experiment, we compared the number of resources usage and the operating time in case of adding array partitioning techniques with block and cyclic type with 2 factor at dimension 2, block and cyclic type with 4 factor at dimension 2, block and cyclic type 2 factor at dimension 1, block and cyclic type with 4 factor at dimension 1, complete type at dimension 2, combination between complete type at dimension 2 and block type 2 and 4 factor at dimension 1 and combination between complete type at dimension 2 and cyclic type 2 and 4 factor at dimension 1.

In case of array partitioning block and cyclic type with factor equal to 2, array stored in 1 bank of memory will be the same way, array partitioning block and cyclic type with factor equal to 4, array stored in 1 bank of memory will be split into 4 banks.

In case of combination of array partitioning complete type at dimension 2 and block or cyclic type at dimension 1 with 4 factor, the array are divided into 4 groups in each group is divided into 46 banks.

Fig. 11 shows the comparison of resource usage for doing Hough transform in each type of array partitioning. The result is present in percentage from total available resource on device. From the result, we found that array partitioning

complete type at dimension 2, combination of complete type at dimension 2 with block type 2 factor at dimension 1 and combination of complete type at dimension 2 with cyclic type 2 factor at dimension 1 can reduce number of block RAM from 109% to 79%, which be able to implement on the device.

From the comparison of operating time of each type of array partitioning in TABLE I, we found that the case of adding array partitioning complete type spent least operating time, which 0.8272 times faster than the original at clock frequency 100 MHz. This shows that adding array partitioning complete type at dimension 2 can be the best type because not only the least resource usage but also the fastest operating time for doing line detection using Hough transform.

CONCLUSION

This paper has presented memory management using array partitioning block, cyclic and complete type with different factor at different dimension on Xilinx Vivado High Level Synthesis to an array of Hough transform, which needs a large memory to store data of array but resources are limited on FPGA. Array partitioning is an optimization technique to manage memory to reduce resources and increase speed of operating time. In the experiment, array partitioning complete type at dimension 2 was the best array partitioning type to reduce not only percentage of block RAM usage from 109% to 79% but also latency, which spent 0.8272 times faster than the original at clock frequency 100 MHz.

REFERENCES

- [1] J. Su, F. Yang, X. Zeng, D. Zhou, and J. Chen, "Efficient Memory Partitioning for Parallel Data Access in FPGA via Data Reuse," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1674–1687, Oct. 2017.)
- [2] H. Ye, G. Shang, L. Wang, and M. Zheng, "A New Method Based on Hough Transform for Quick Line and Circle Detection," in 2015 8th International Conference on Biomedical Engineering and Informatics (BMEI), 2015, pp. 52–56.
- [3] H. Chang, J. Wang, and L. Wang, "A Hough Transform Based Line Detection Method Utilizing Improved Voting Scheme," in Proceedings of the 29th Chinese Control Conference, 2010, pp. 2857–2860.
- [4] P. Li, Y. Wang, P. Zhang, G. Luo, T. Wang, and J. Cong, "Memory Partitioning and Scheduling Co-Optimization in Behavioral Synthesis," in 2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2012, pp. 488–495.
- [5] D. Huiying and H. Tingting, "Transmission Line Extraction Method Based on Hough Transform," in The 27th Chinese Control and Decision Conference (2015 CCDC), 2015, pp. 4892–4895.
- [6] P. Promrit and W. Suntiarnortut, "Design and Development of Lane Detection Based on FPGA," in 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017, pp. 1–4.

ภาคผนวก 3

Pertanika Journal of Science & Technology (JST)

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

1 79

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

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

2

80

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

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

*Department of Electrical Engineering, Faculty of Engineering, Prince of
Songkla University, Hat Yai, Songkhla 90112, Thailand*

Toyota Tsusho Nexty Electronics (Thailand) co, Ltd Bangkok, Thailand

*Corresponding author

List of Tables:

Table 1.

Table 2.

Table 3.

Table 4.

Table 5.

Table 6.

List of Figures:

Figure 1.

Figure 2.

Figure 3.

Figure 4.

Figure 5.

Figure 6.

Figure 7.

Figure 8.

Figure 9.

Figure 10.

Figure 11.

Figure 12.

Figure 13.

Figure 14.

Figure 15.

Figure 16.

Figure 17.

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

ABSTRACT

In this work, road lane detection is proposed to reciprocate the requirements of Lane Keeping Assistant System (LKAS) and Lane Departure Warning System (LDWS), which are the position of lane line on the image and the tolerance to the unexpected road lane, especially curve lane. The angle calculation is proposed to realize the curve's direction. The speed and memory usage of an algorithm is improved as well by adding the High-Level Synthesis (HLS) optimization techniques. Array sizing, loop unrolling, loop pipelining, array partitioning, and HLS interface management are respectively applied according to the limitation of resources and the speed of operation time using HLS development on Xilinx Zynq-7000 family (Zybo ZC7010-1). From the experimental results, the proposed method reaches 6.66 times earlier than the primitive method at clock frequency 100 MHz.

INTRODUCTION

Advanced Driving Assistant Systems (ADAS) are enhance self-driving in autonomous driving cars. Lane Keeping Assistant System (LKAS) or Lane Departure Warning System (LDWS), which is a part of ADAS, consists of several parameters to operate. The lane line position is the most important parameter for controlling and warning in autonomous cars.

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 curves, and strange lane shape lane was reviewed in (Feniche & Mazri, 2019). The typical model for lane detection consists of five steps: image cleaning, feature detection, model application, tracking integration, and coordinates translation. For image cleansing, gradient conversion is generated from the RGB image to get the obvious boundary of lane detection and also a gradient separation between road and lane based on linear discriminant. The case of shadows is resolved by intensive normalization between brightness and shadow areas. Both of Canny edge operator and Sobel filter is used to be edge detection and Hough transform (HT) is commonly used to be line detection in the feature extraction step. Although Hough transform is commonly used for line detection in road lane detection, it still cannot represent the curve of the line. Thus in (Duong, Pham, Tran, Nguyen, & Jeon, 2016), Bird's eye view should be added to improve the curve line detection. However, the generic model is quite complicated to implement and the process operation time is long.

The real-time lane detection proposed in (Lee, Shin, Jung, Park, Oh, & Lee, 2017) was implemented in a high-performance device (IMX6Q) to decrease the operation time, which is relatively high cost by using a simple filter for pre-processing and Kalman filter for tracking. The approximate speed was 15 frames per second (FPS). An alternative development of the real-time road lane detection to get higher speed and lower cost is developing on hardware devices such as Zynq-7000 family, which is a combination

technology between FPGA as a Programmable Logic (PL) part and microcontroller (ARM-cortex A9) as a Processing System (PS). In (Promrit, & Suntiamorntut, 2017) implementation of real-time lane detection on a hardware device (Zynq-7000 family) has been implemented by applying statistic and blob detection in pre-processing to get a clear image. The remedial edge detection is proposed in (Hwang, & Lee, 2016) to reduce the computational complexity of the traditional Canny-Hough lane detection algorithm to get real-time lane detection on an FPGA platform (Terasic's DE1-SoC board). In (Khongprasongsiri, Kumhom, Suwansantisuk, Chotikawanid., Chumpol, & Ikura, 2018), the pipelining technique is applied to resolve the bottleneck of HT by using High-Level Synthesis (HLS) on Xilinx's Zynq 7000 chip XC7045. However, the optimization techniques, which are available on the hardware devices have not been completely analyzed yet.

As above mentioned, we found that road lane detection, which is one of the important parameters of ADAS, needs the position of lane line for forward controlling. Curve lane should be recognized without adding the complicated method that increases the operation time.

In this study, the road lane detection technique is developed to represent the line position on the image with curve recognition without Bird's eye view added to avoid the complexity of Zybo ZC7010-1. With the sequential flow, the first considering is started with array sizing for limitation of the resource. Loop unrolling and loop pipelining will be the second considering to increase operation speed. The third considering is array partitioning that prevents the bottleneck of the previous step and the HLS interface will be the last step to increase the operation speed by suitable selection interface. The development is expedited by using C language based on the Xilinx Vivado HLS, which generates the parallel accelerators on PL part to enhance the speed of processing. HLS optimization based resource and memory management are

also proposed to decrease the number of memories compared to conventional techniques which can be implemented adequately based on Zybo ZC7010-1.

THEORY

A. Edge detection

There are many methods to perform edge detection. Prewitt, Sobel, Robert, and Canny edge detection 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 to detect edge has different gradient values as shown in Figure 1, Figure 2, and Figure 3.

Meanwhile, canny edge detection also uses the gradient method, but more

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

complexity of step than the others. 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 formula in [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.

B. Hough transform

For line and circles identification, Hough transform is one of a feature extraction to process by using a voting procedure to detect incomplete

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

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

instances of objects of a particular type of shape. Generally, according to [2], the straight line is capable to represent the linear equation with two important parameters: slope (m) and intercept (c). [3] represents the polar form of a line.

From Figure 4, 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 . At the position on the blue straight line on x-y plane indicates to each curve on θ - ρ plane. On 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.

C. Pipelining

Pipelining is a method for rapidity optimization, which decreases the latency of the process by the initiation interval (II) decreasing for a function or loop. Figure 5 is a sample of the pipelining technique applies to for-loop, which contains 3 operations. The left-hand side of Figure 5 is an operating sequence of the process without pipelining is added. 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 for-loop with II equal to 1. In the second clock cycle, the first operation in the consecutive iteration can be immediately operated according to the right-hand side of Figure 5.

D. Unrolling

Unrolling is a performance optimization technique as well. It allows some or all of iteration can be concomitant. Figure 6 is an example of applying loop unrolling. Without unrolling, 6 iterations are needed to process. Only 2 iterations are needed in case of using unrolling.

E. 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 type. In the case of block type, a large array is divided into balanced blocks whose array elements

are consecutively arranged. In the case of cyclic type, a large array is divided into a balanced block and interleaved with elements. In the case of complete type, an array is divided into single elements.

F. 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 types of port on RTL design, which includes clock and reset port, block-level interface protocols, and port-level interface protocols are created by Xilinx Vivado. Typically, the 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 is complete. 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.

OVERALL FRAMEWORK

A framework of lane tracking proposed in this paper is shown in Figure 8. There are 4 main steps, 1. Pre-processing 2. Edge detection 3. Line detection and 4. Angle calculation. Edge detection and line detection are implemented on PL part and others are implemented on PS part to increase the operation speed. Pre-processing is expanded in Figure 9 that includes cropping image, 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, Sengchuai, Booranawong, Hoyingcharoen, Chumpol, Ikura, & Jindapeth, 2018), Robert edge detection is selected to perform edge detection and Hough line transform

is applied to the line detection. The positions of (x_1, y_1) and (x_2, y_2) are obtained from line detection step 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 90° and θ_2 must be less than 90° . The second case is the left curve as shown in Figure 11, in which θ_1 and θ_2 must be less than 90° . The last case is the right curve as shown in Figure 12, in which θ_1 and θ_2 must be over 90° .

OPTIMIZATION IN HIGH LEVEL SYNTHESIS

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 High-Level Synthesis on Zynq-7000, which allows C and C++ languages and authorizes efficient method as loop unrolling, loop pipelining, or array partitioning, etc. The proposed 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.

A. 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 [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 [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 0°-360° to 30°-150°. Thus the second dimension is reduced from 360 to 120, which is sufficient for road lane detection.

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

B. 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 15. 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 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. 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 15, 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 13, 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.

$$\begin{aligned} hough_{(1,2)}[rho][theta] & \quad [5] \\ & = hough_{(1,2)}[rho][theta] + 1 \end{aligned}$$

The second minor nested loop is counting times of intersection in θ and ρ space according to the Hough line transform. [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 17.

`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 $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ 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 pipelining must be selected instead of loop unrolling because of the limitation of resources on the device.

C. 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, $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ 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 ($\text{hough1}[\rho+x][\theta+1]$ and $\text{hough2}[\rho+x][\theta+1]$) of array $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ cannot be accessed until the first element ($\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$) is successfully accessed. Due to the bottleneck issue, loop unrolling in Figure 16 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 $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ in Figure 13 . 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.

D. 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.

EXPERIMENT AND DISCUSSION

This experiment aims to increase the speed of lane detection by adding 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.

A. Process profiling

In the beginning, the profile of the operation time of each process in 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.

B. 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, $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ is resized from 2203×120 to 224×120 according to HT theory, which in ρ 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 for 1,710% to 57%.

C. Loop unrolling and loop pipelining results

In case of considering to least latency, loop unrolling is sufficient to add along with first, third, and last nested loop of Figure 15. 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 $\text{hough1}[\rho][\theta]$, $\text{hough2}[\rho][\theta]$ and two more arrays, $\sin[\theta]$ and $\cos[\theta]$, 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.

D. Array partitioning results

Since $\text{hough1}[\rho][\theta]$, $\text{hough2}[\rho][\theta]$ 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 $\text{hough1}[\rho][\theta]$, $\text{hough2}[\rho][\theta]$, $\sin[\theta]$ and $\cos[\theta]$ compare to the first proposed method. Both $\text{hough1}[\rho][\theta]$ and $\text{hough2}[\rho][\theta]$ 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.

E. 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 14, 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, which 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`.

CONCLUSION

This paper has presented a 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 proposed method can reduce 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.

The proposed method in this work can reach higher performance by implementing on the High-performance devices. The optimization can further gain the benefit on the devices with sufficient resources.

REFERENCES

- Duong, T. T., Pham, C. C., Tran, T. H.-P., Nguyen, T. P., & Jeon, J. W. (2016). Near Real-Time Ego-Lane Detection in Highway and Urban Streets, *Proceedings of Consumer Electronics-Asia (ICCE-Asia)* (pp. 1–4).
- Feniche, M., & Mazri, T. (2019). Lane Detection and Tracking for Intelligent Vehicles: A Survey, *Proceedings of 2019 International Conference of Computer Science and Renewable Energies (ICCSRE)* (pp. 1–4).
- Hwang, S., & Lee, Y., (2016). FPGA-Based Real-Time Lane Detection for Advanced Driver Assistance Systems, *Proceedings of 2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)* (pp. 218–219).
- 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, *Proceedings of 2018 International Workshop on Advanced Image Technology (IWAIT)* (pp. 1–4)
- 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, *Proceedings of 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)* (pp. 275–277).
- Promrit, P., & Suntiamorntut, W., (2017). Design and Development of Lane Detection Based on FPGA, *Proceedings of 2017 14th International Joint*

Conference on Computer Science and Software Engineering (JCSSE)

(pp. 1–4).

Solod, P., Sengchuai, K., Booranawong, A., Hoyingcharoen, P., Chumpol, S.,
Ikura, M., & Jindapeth, N., (2018). Model Based Design Approach for
Road Lane Tracking. In *Asia Pacific Conference on Robot IoT System
Development and Platform 2018 (APRIS2018)* (pp. 17–18).

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

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

Table 2

Comparison of default and array sizing

Process	Method	Resource (%)				Latency (clock cycle)
		<i>DSP</i>	<i>BRAM</i>	<i>CLB</i>	<i>FF</i>	
All	Default	12	30	9	3	103,709,185
1 st	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
2 nd	Pipelining	13	30	10	3	46,056,556
Edge + Line detection	Combination	15	30	10	3	45,046,530
3 rd	Unrolling	12	30	177	79	98,457,416
	Pipelining	12	30	15	6	98,457,432
4 th	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

Table 3

Comparison of loop unrolling and loop pipelining in each loop

Optimization	Resource (%)				Latency (clock cycle)	
	<i>DSP</i>	<i>BRAM</i>	<i>CLB</i>	<i>FF</i>		
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
Complete type	17	42	39	8	15,697,034	
Block type, F 2	18	42	27	9	15,777,845	
sin & cos	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	

Table 4

Comparison of array partitioning

Optimization	Resource				Latency (clock cycle)
	<i>DSP</i>	<i>BRA</i> <i>M</i>	<i>CL</i> <i>B</i>	<i>FF</i>	
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 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

Table 6

Shows the summary of the proposed optimization techniques compared the default method

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

(a)

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

(b)

Figure 1. Gradient value of Prewitt Edge Detection

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

(a)

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

(b)

Figure 2. Gradient value of Sobel Edge Detection

1	0
0	-1

(a)

0	1
-1	0

(b)

Figure 3. Gradient value of Robert Edge Detection

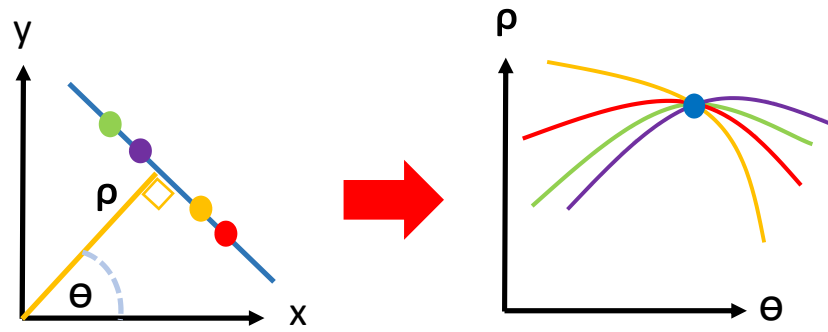


Figure 4. Hough transform

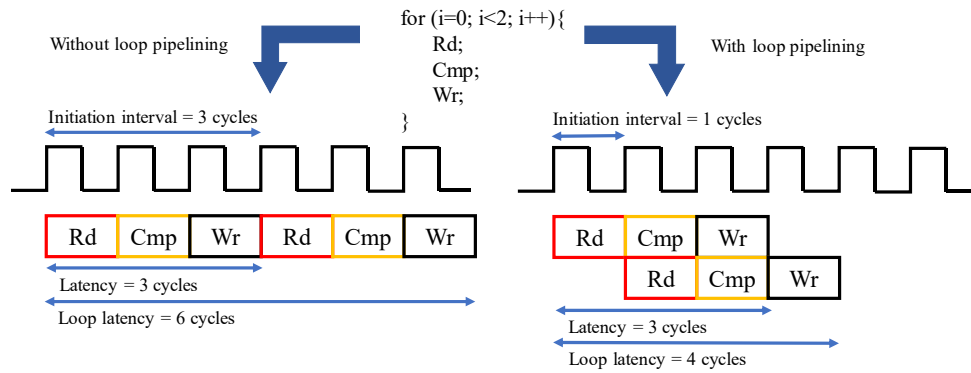


Figure 5. Loop pipelining

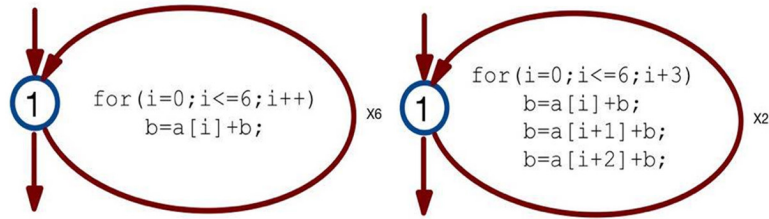


Figure 6. Loop unrolling

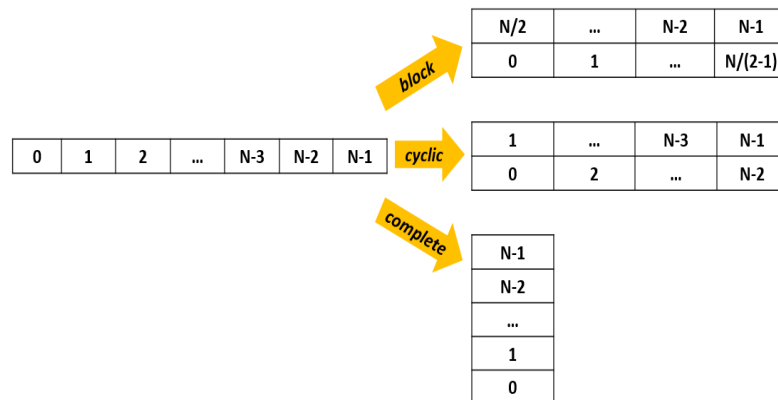


Figure 7. Array partitioning

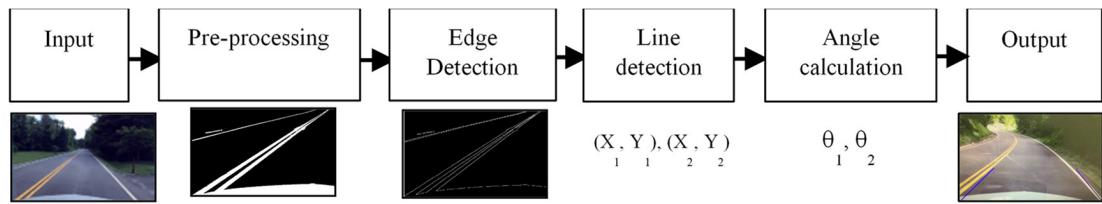


Figure 8. Road lane detection framework

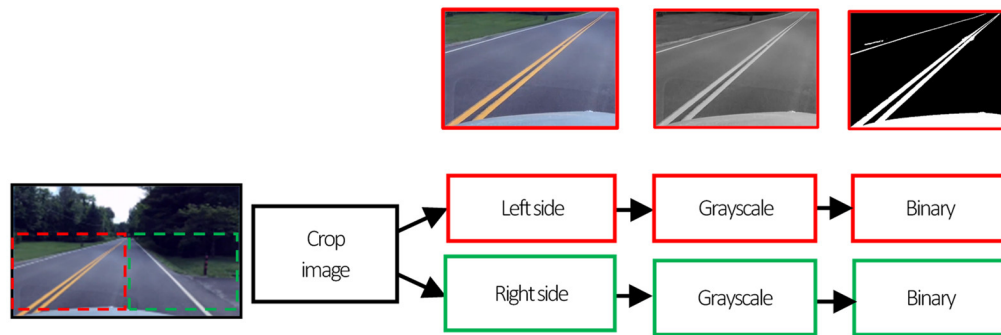


Figure 9. Pre-processing

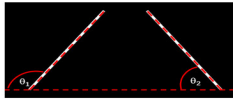


Figure 10. Straight
lane

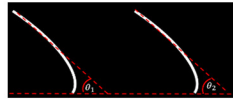


Figure 11. Left curve

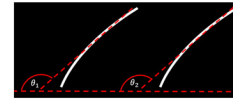


Figure 12. Right
curve

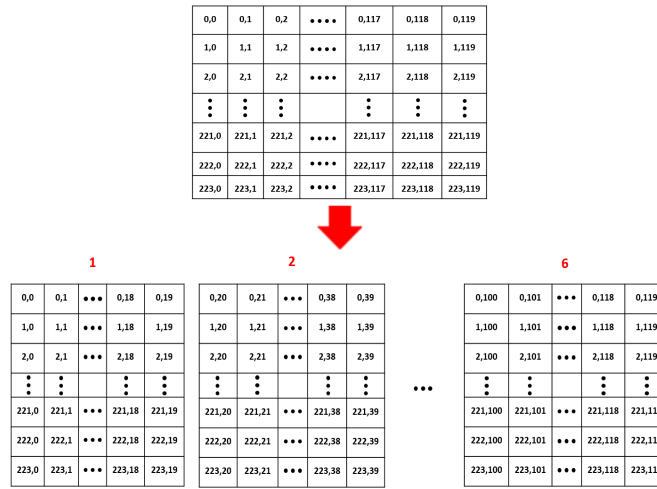


Figure 13. Array partitioning of hough1[rho][theta] and hough2[rho][theta]

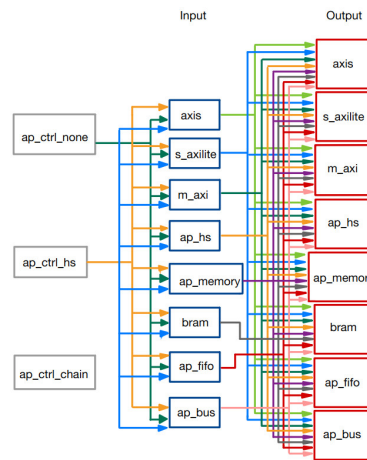


Figure 14. HLS interface management

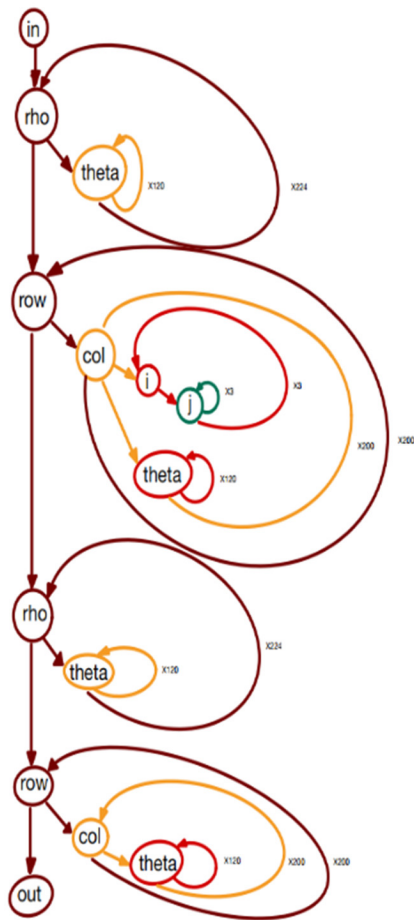


Figure 15. Loops and C language of edge and line detection

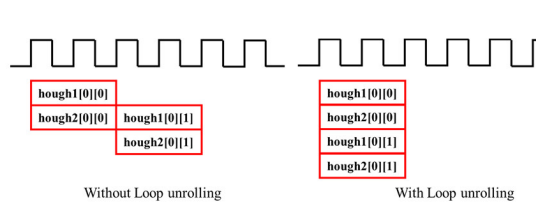


Figure 16. Loop unrolling of in 1st nested loop

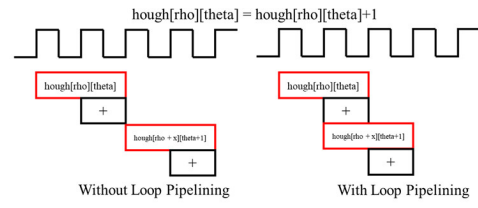


Figure 17. Loop pipelining in 2nd minor nested loop

ประวัติผู้เขียน

ชื่อ ชื่อสกุล นางสาวปนัดดา โสฬส

รหัสประจำตัวนักศึกษา 6010120066

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิศวกรรมศาสตรบัณฑิต (วิศวกรรมเมคาทรอนิกส์)	มหาวิทยาลัยสงขลานครินทร์	2559

ทุนการศึกษาที่ได้รับ (ที่ได้รับระหว่างการศึกษา)

1. ทุนศิษย์ก้นกุฎิ คณะวิศวกรรมศาสตร์
2. ทุนอุดหนุนการวิจัยเพื่อวิทยานิพนธ์ มหาวิทยาลัยสงขลานครินทร์

การตีพิมพ์เผยแพร่ผลงาน

Panadda Solod, Kiattisak Sengchuai, Apidet Booranawong, Pakpoom Hoyingcharoen, Surachate Chumpol, Masami Ikura and Nattha Jindapeth, “Model Based Design Approach for Road Lane Tracking,” Proceedings of Asia Pacific Conference on Robot IoT System Development and Platform 2018, Oct 2018.

Panadda Solod, Kiattisak Sengchuai, Apidet Booranawong, Pakpoom Hoyingcharoen, Surachate Chumpol, Masami Ikura and Nattha Jindapeth, “Memory Optimization for Accelerating Hough Transform on FPGA using High Level Synthesis,” Proceedings of IEEE International Circuits and Systems Symposium, pp. 1 – 4, Sep 2019.

Panadda Solod, Kiattisak Sengchuai, Apidet Booranawong, Pakpoom Hoyingcharoen, Surachate Chumpol, Masami Ikura and Nattha Jindapeth, “Optimizations of Road Lane Detection using High Level Synthesis Development on Zynq-7000” Pertanika Journal of Science & Technology. (Submitted)