# การออกแบบและพัฒนาฮาร์ดแวร์เร่งความเร็วของการเรียนรู้เชิงลึกด้วยเอฟพีจีเอ

โดย

นายธนพล ทองคำ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ แผน ก แบบ ก 2

ภาควิชาวิศวกรรมไฟฟ้า

มหาวิทยาลัยศิลปากร

ปีการศึกษา 2567

# การออกแบบและพัฒนาฮาร์ดแวร์เร่งความเร็วของการเรียนรู้เชิงลึกด้วยเอฟพีจีเอ

โดย
นายธนพล ทองคำ


วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ แผน ก แบบ ก 2
ภาควิชาวิศวกรรมไฟฟ้า
มหาวิทยาลัยศิลปากร
ปีการศึกษา 2567

# DISIGN AND DEVELOPMENT OF HARDWARE ACCELERATORS FOR DEEP LEARNING USING FPGA

By

MR. Thanapol THONGKHAM

A Thesis Submitted in Partial Fulfillment of the Requirements

for Master of Engineering (ELECTRICAL AND COMPUTER ENGINEERING)

Department of ELECTRICAL ENGINEERING

Academic Year 2024

| | |
|---|---|
| หัวข้อ | การออกแบบและพัฒนาฮาร์ดแวร์เร่งความเร็วของการเรียนรู้เชิงลึกด้วยเอฟพีจีเอ |
| โดย | นายธนพล ทองคำ |
| สาขาวิชา | วิศวกรรมไฟฟ้าและคอมพิวเตอร์ แผน ก แบบ ก 2 |
| อาจารย์ที่ปรึกษาหลัก | ผู้ช่วยศาสตราจารย์ ดร. ยุทธนา เจวจินดา |

คณะวิศวกรรมศาสตร์และเทคโนโลยีอุตสาหกรรม มหาวิทยาลัยศิลปากร ได้รับพิจารณาอนุมัติให้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์และ
(ผู้ช่วยศาสตราจารย์ ดร. อรุณศรี ลีจีรจำเนียร) เทคโนโลยีอุตสาหกรรม

พิจารณาเห็นชอบโดย

.....ประธานกรรมการ
(รองศาสตราจารย์ ดร. ชูเกียรติ สอดศรี)

.....อาจารย์ที่ปรึกษาหลัก
(ผู้ช่วยศาสตราจารย์ ดร. ยุทธนา เจวจินดา)

.....ผู้ทรงคุณวุฒิภายใน
(ผู้ช่วยศาสตราจารย์ ดร. จิรัฏฐ์ เหมือนชู)

.....ผู้ทรงคุณวุฒิภายนอก
(รองศาสตราจารย์ ดร. ยรรยงค์ พันธ์สวัสดิ์)

640920045 : วิศวกรรมไฟฟ้าและคอมพิวเตอร์ แผน ก แบบ ก 2
คำสำคัญ : เอฟพีจีเอ, โครงข่ายประสาทเทียมคอนโวลูชัน, การคอนโวลูชันด้วยวิโนกราด, ระบบเลขแบบซีเอสดี

นาย ธนพล ทองคำ: การออกแบบและพัฒนาฮาร์ดแวร์เร่งความเร็วของการเรียนรู้เชิงลึกด้วยเอฟพีจีเอ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก : ผู้ช่วยศาสตราจารย์ ดร. ยุทธนา เจวจินดา

ในปัจจุบัน การเรียนรู้เชิงลึก (Deep Learning) ได้รับความนิยมและนำไปประยุกต์ใช้งานในหลากหลายสาขา เช่น การจำแนกภาพ การตรวจจับวัตถุ และระบบปัญญาประดิษฐ์ต่าง ๆ ซึ่งกระบวนการเรียนรู้ดังกล่าวมีความซับซ้อนสูงและต้องใช้พลังการประมวลผลจำนวนมาก ทำให้เกิดข้อจำกัดในการนำไปใช้งานบนระบบฝังตัวที่มีทรัพยากรจำกัด งานวิจัยนี้จึงมุ่งเน้นการออกแบบและพัฒนาสถาปัตยกรรมฮาร์ดแวร์เร่งความเร็วสำหรับการเรียนรู้เชิงลึกด้วยอุปกรณ์ FPGA ซึ่งมีข้อดีคือสามารถประมวลผลแบบขนาน ใช้พลังงานต่ำ ขนาดเล็ก และปรับแต่งได้ตามความต้องการ

โดยได้ศึกษาการออกแบบตัวเร่งความเร็วสำหรับโครงข่ายประสาทเทียมคอนโวลูชัน (Convolutional Neural Network: CNN) ซึ่งเลือกใช้อัลกอริทึม Winograd convolution และระบบตัวเลข Canonical Signed Digit (CSD) เพื่อเพิ่มประสิทธิภาพด้านความเร็วและลดการใช้ทรัพยากร พร้อมทั้งพัฒนาต้นแบบฮาร์ดแวร์บนบอร์ด Zybo Z7-20 FPGA และเปรียบเทียบประสิทธิภาพกับวิธีมาตรฐาน การออกแบบสถาปัตยกรรมฮาร์ดแวร์ดังกล่าวถูกทดสอบและประเมินประสิทธิภาพทั้งด้านความเร็วและการใช้ทรัพยากร พบว่าสามารถลดจำนวนการคูณและเพิ่มความเร็วในการประมวลผลได้อย่างมีนัยสำคัญ

ผลจากงานวิจัยนี้ทำให้ได้ต้นแบบตัวเร่งความเร็วเชิงลึกบน FPGA ซึ่งเหมาะสมสำหรับระบบฝังตัวต้นทุนต่ำ อีกทั้งยังสามารถปรับเปลี่ยนโครงสร้างและพารามิเตอร์ได้ตามต้องการ เพื่อรองรับการประยุกต์ใช้งานด้านการมองเห็นของหุ่นยนต์เคลื่อนที่หรือระบบสมองกลฝังตัวอื่น ๆ ในอนาคต

640920045 : Major (ELECTRICAL AND COMPUTER ENGINEERING)
Keyword : FPGA, Convolution neural network, Winograd convolution, CSD number system

MR. Thanapol THONGKHAM : Disign and Development of Hardware Accelerators for Deep Learning using FPGA Thesis advisor : Assistant Professor Yutana Jewajinda

At present, Deep Learning has gained widespread popularity and is applied in various fields such as image classification, object detection, and artificial intelligence systems. However, the high computational complexity and intensive processing power required in Deep Learning pose limitations for its implementation on embedded systems with limited resources. This research focuses on the design and development of hardware accelerators for Deep Learning using FPGA devices, which offer advantages in parallel processing, low power consumption, compact size, and high flexibility in customization.

This study investigates the design of hardware accelerators for Convolutional Neural Networks (CNNs), employing the Winograd convolution algorithm and the Canonical Signed Digit (CSD) number system to enhance computational speed and reduce hardware resource usage. A hardware prototype was implemented on a Zybo Z7-20 FPGA board and its performance was evaluated against standard processing approaches. The proposed hardware architectures demonstrated significant improvements in processing speed by reducing multiplication operations while optimizing resource utilization.

The research results yielded a prototype Deep Learning accelerator on FPGA suitable for cost-effective embedded systems, with the flexibility to adjust architecture and parameters for various applications. It is particularly well-suited for vision-based systems in mobile robotics and other embedded artificial intelligence applications.

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีจากความกรุณาของผู้ช่วยศาสตราจารย์ ดร. ยุทธนา เจวจินดา ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ที่คอยช่วยเหลือ ให้แนวคิด และให้คำปรึกษาตลอดการทำวิจัยที่ผ่านมา รวมทั้งขอขอบคุณรองศาสตราจารย์ ดร.ชูเกียรติ สอดศรี ประธานกรรมการสอบวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร.จิรัฏฐ์ เหมือนชู กรรมการการสอบวิทยานิพนธ์ และรองศาสตราจารย์ ดร.ยรรยงค์ พันธ์สวัสดิ์ ผู้ทรงคุณวุฒิภายนอก ที่กรุณาให้ข้อเสนอแนะแก่การทำวิทยานิพนธ์ฉบับนี้ให้มีความถูกต้องสมบูรณ์มากขึ้น

ผู้วิจัยขอกราบขอบพระคุณในความกรุณาของอาจารย์ที่ปรึกษา ผู้ช่วยศาตราจารย์ ดร.ยุทธนา เจวจินดา และหน่วยงานภาควิชาวิศวกรรมไฟฟ้า ที่ให้ความรู้ และการสนับสนุนไม่ว่าเป็นด้านทุนสำหรับการศึกษาเล่าเรียน และอุปกรณ์ต่างๆที่ใช้ในงานวิจัย

ธนพล ทองคำ

# สารบัญ

หน้า

# สารบัญตาราง

**หน้า**



# สารบัญรูปภาพ

หน้า

# บทที่ 1
# บทนำ

## 1.1 ความเป็นมา และความสำคัญ

ในปัจจุบันการเรียนรู้เชิงลึก (Deep learning) ประสบความสำเร็จในการประยุกต์การใช้งานในด้านต่างๆ เช่น Image classification [1-5] ,Object detection [6] และอื่นๆ การเรียนรู้เชิงลึกเป็นส่วนหนึ่งของการเรียนรู้ของเครื่อง (Machine learning) ที่มีพื้นฐานจากโครงข่ายประสาทเทียม (Artificial neural network) การเรียนรู้เชิงลึกมี โครงสร้างที่หลากหลายและแตกต่างกันจำนวนมาก เช่น Deep neural network (DNNs), Recurrent neural networks (RNNs), Convolutional neural networks (CNNs) และอื่นๆ

การเรียนรู้เชิงลึกเป็นวิธีการที่ต้องการใช้พลังการคำนวณมหาศาลจึงทำให้การประยุกต์ใช้งานของการเรียนรู้เชิงลึกในปัจจุบันทำงานอยู่บนอุปกรณ์ที่มีความสามารถในการคำนวณความเร็วสูงเช่น GPU หรือคอมพิวเตอร์แบบขนานและแบบกระจาย FPGA เป็นทางเลือกนึงที่มีความสามารถในการคำนวณแบบขนาน มีขนาดเล็ก อีกทั้งยังสามารถออกแบบเป็นวงจรดิจิตอลที่มีการใช้พลังงานที่ต่ำได้ และมีความยืดหยุ่นในการออกแบบสูง จึงเป็นอุปกรณ์ที่มีความเหมาะสมในการนำมาประยุกต์ใช้งานการเร่งความเร็วการเรียนรู้เชิงลึกบนระบบฝังตัวได้เป็นอย่างดี

การออกแบบตัวเร่งความเร็วสำหรับการเรียนรู้เชิงลึกด้วยเอฟพีจีเอ นับเป็นหัวข้อของงานวิจัยที่ได้รับความสนใจต่อเนื่องจนถึงปัจจุบัน [7-17] โดยสถาปัตยกรรมของตัวเร่งความเร็วสำหรับการเรียนรู้เชิงลึกที่ใช้หลักการนำอัลกอริทึมของการเรียนรู้เชิงลึกไปทำการออกแบบในรูปแบบสถาปัตยกรรมฮาร์ดแวร์โดยตรงบนเอฟพีจีเอถูกนำเสนอในงานวิจัย [8, 12, 13, 15] ซึ่งมีข้อดีที่สามารถออกแบบได้ง่ายและทรัพยากรต่างๆจะถูกใช้ได้อย่างมีประสิทธิภาพเนื่องจากการที่สถาปัตยกรรมของตัวเร่งความเร็วถูกออกแบบมาสำหรับอัลกอริทึมนั้นโดยเฉพาะ แต่มีข้อเสียคือการปรับเปลี่ยนอัลกอริทึมจะเป็นสิ่งที่ทำได้ยาก ซึ่งสำหรับการเรียนรู้เชิงลึกนั้นเป็นหัวข้อที่ถูกทำวิจัยอย่างมากและต่อเนื่องส่งผลให้อัลกอริทึมของการเรียนรู้เชิงลึกมีการเปลี่ยนแปลงไปสู่อัลกอริทึมที่ดีกว่าตลอดเวลา ซึ่งแนวคิดที่ทำให้สถาปัตยกรรมของตัวเร่งความเร็วสำหรับการเรียนรู้เชิงลึกบนเอฟพีจีเอนั้นสามารถปรับเปลี่ยนอัลกอริทึมและพารามิเตอร์ต่าง ๆในการทำงานให้เหมาะสมสำหรับแต่ละการประยุกต์ใช้งานได้ถูกนำเสนอใน [10, 11, 14, 18] โดยการทำงานจะใช้

การแปลงอัลกอริทึมของการเรียนรู้เชิงลึกที่ต้องการมาเป็นชุดคำสั่ง(instruction) สำหรับการตั้งค่าการทำงาน (configuration) ของตัวเร่งความเร็วให้เป็นไปตามแต่ละอัลกอริทึมซึ่งมีลักษณะการทำงานเช่นเดียวกันกับตัวประมวลผลกลางของคอมพิวเตอร์ทั่ว ๆไป ซึ่งวิธีการนี้ทำให้สามารถใช้ตัวเร่งความเร็วเดิมในการประยุกต์ใช้งานที่แตกต่างกันได้ งานวิจัยที่เน้นการพัฒนาตัวเร่งในระบบเอฟพีจีเอฝังตัวแสดงใน [9] การใช้งานเอฟพีจีสำหรับตัวเร่งความเร็วทำให้สามารถลดการใช้งานพลังงานแสดงใน [11-13] แนวทางการสร้างกรอบการออกแบบตัวเร่งความเร็วโดยใช้เอฟพีจีนำเสนอใน [19, 20] อีกแนวทางหนึ่งที่ได้รับความสนใจคือการเน้นใช้ทรัพยากรในตัวเอฟพีจีมาช่วยทำงานเร่งความเร็วซึ่งถูกนำเสนอใน [21] อีกแนวทางหนึ่งสำหรับการแปลงโดยตรงจากอัลกอริทึมโดยใช้ภาษาชั้นสูงนำเสนอใน [22] ซึ่งวิธีการนี้จะได้สถาปัตยกรรมฮาร์ดแวร์ที่มีประสิทธิภาพไม่สูงมากแต่สะดวกออกแบบตัวเร่ง

งานวิจัยนี้จะมุ่งเน้นการออกแบบสถาปัตยกรรมของตัวเร่งความเร็วสำหรับการเรียนรู้เชิงลึกที่ใช้ทรัพยากรจำกัดบนตัวเอฟพีจีเอและเลือกสร้างตัวเร่งการเรียนรู้เชิงลึกจากวิธีใหม่ที่มีประสิทธิภาพสูงซึ่งเหมาะสมกับการสร้างในฮาร์ดแวร์หรือร่วมกันทำงานระหว่างฮาร์ดแวร์และซอฟต์แวร์บนระบบฝังตัวซึ่งใช้เอฟพีจีเอเป็นหลักสำหรับปัญหาการจำแนกวัตถุจากภาพเพื่อใช้งานการมองเห็นของหุ่นยนต์เคลื่อนที่ โดยงานวิจัยนำเสนอทางเลือกในการออกแบบตัวเร่งความเร็วที่ใช้ระบบเอฟพีจีเอแบบฝังตัวที่มีราคาไม่แพงเพื่อให้ได้ต้นแบบที่สามารถนำมาใช้งานได้จริงทั่วไปและสามารถปรับเปลี่ยนสถาปัตยกรรมสำหรับตัวเร่งความเร็วให้มีประสิทธิภาพการทำงานที่ดีขึ้นได้

**1.2 วัตถุประสงค์**

1. เพื่อศึกษาทฤษฎีของการเรียนรู้เชิงลึก
2. เพื่อศึกษาสถาปัตยกรรมฮาร์ดแวร์ของตัวเร่งความเร็วของการเรียนรู้เชิงลึก สำหรับใช้งานในระบบสมองกลฝังตัว
3. เพื่อออกแบบ และพัฒนาสถาปัตยกรรมใหม่ของตัวเร่งความเร็วของการเรียนรู้เชิงลึก
4. สร้างต้นแบบเพื่อประเมินประสิทธิภาพของสถาปัตยกรรมฮาร์ดแวร์ใหม่ที่นำเสนอในเอฟพีจีเอ

**1.3 สมมติฐานงานวิจัย**

จากการศึกษาพบว่าปัญหาที่เกี่ยวกับการโครงข่ายประสาทเทียมจะเป็นด้านประสิทธิภาพในการประมวลผลเนื่องจากในยุคปัจจุบันมีข้อมูลหลากหลายชนิด เช่น ข้อมูลเสียง ข้อมูลรูปภาพ และอื่น ๆ ซึ่งข้อมูลเหล่านี้จะมีความซับซ้อนที่สูงทำให้ก่อเกิดปัญหาด้านประสิทธิภาพด้านการประมวลผล ซึ่งมีงานวิจัยที่ใช้งาน FPGA เพื่อเร่งความเร็วโครงข่ายประสาทเทียม เนื่องจาก FPGA สามารถออกแบบฮาร์ดแวร์ได้เอง ซึ่งจะมีข้อได้เปรียบกว่าการใช้งาน CPU และ GPU ในการประมวลผล

โมเดลที่จะใช้ในการปรับปรุงประสิทธิภาพคือ CNN (Convolution neural network) ซึ่งภายใน CNN ประกอบด้วย Convolution layer มาประกอบกันกับ Layer ชนิดอื่น เช่น Pooling layer แล้วนำกลุ่ม Layer ดังกล่าวมาซ้อนต่อๆ กัน โดยอาจเปลี่ยน Hyperparameter บางอย่าง เช่นขนาดของ Filter layer (ซึ่งเป็นส่วนหนึ่งของ Convolution layer) และจำนวน Channel ของ layer วิธีการนำเอาส่วนต่างๆ มาประกอบกันนี้ เรียกว่าเป็นโครงสร้าง (Architecture) ของ CNN ซึ่งมีโมเดลหลายแบบ เช่น AlexNet [2], ZF Net [5], VGG Net [3], GoogLeNet [4], Microsoft ResNet [1] ซึ่งแต่ละโมเดลจะมีข้อดีข้อเสียที่แตกต่างกันจึงเหมาะสมตามแต่ละการประยุกต์ใช้งาน

ในงานวิจัยนี้จะมุ่งเน้นการออกแบบสถาปัตยกรรมของตัวเร่งความเร็วสำหรับการเรียนรู้เชิงลึกที่ใช้ทรัพยากรจำกัดบนตัวเอฟพีจีเอและเลือกสร้างตัวเร่งการเรียนรู้เชิงลึกจากวิธีใหม่ที่มีประสิทธิภาพสูงซึ่งเหมาะสมกับการสร้างในฮาร์ดแวร์หรือร่วมกันทำงานระหว่างฮาร์ดแวร์และซอฟต์แวร์บนระบบฝังตัวซึ่งใช้เอฟพีจีเอเป็นหลักสำหรับปัญหาการจำแนกวัตถุจากภาพเพื่อใช้งานการมองเห็นของหุ่นยนต์เคลื่อนที่ โดยงานวิจัยนำเสนอทางเลือกในการออกแบบตัวเร่งความเร็วที่ใช้ระบบเอฟพีจีเอแบบฝังตัวที่มีราคาไม่แพงเพื่อให้ได้ต้นแบบที่สามารถนำมาใช้งานได้จริงทั่วไปและสามารถปรับเปลี่ยนสถาปัตยกรรมสำหรับตัวเร่งความเร็วให้มีประสิทธิภาพการทำงานที่ดีขึ้นได้

**1.4 ขอบเขตของการวิจัย**

1. ศึกษางานวิจัยที่เกี่ยวข้องในการออกแบบ และสร้างตัวเร่งความเร็วของการเรียนรู้เชิงลึกในเอฟพีจีเอ
2. ออกแบบ และเสนอสถาปัตยกรรม และทำการจำลองบนเครื่องคอมพิวเตอร์
3. ทำการทดสอบ และประเมินประสิทธิภาพของต้นแบบที่สร้างขึ้นบนเอฟพีจีเอ และเปรียบเทียบกับงานวิจัยอื่นที่เกี่ยวข้อง

**1.5 ความจำกัดของการวิจัย**

1. ซอฟต์แวร์ที่ใช้การจำลองการทำงาน มีข้อจำกัดเนื่องจากเป็นชุดพัฒนาสำหรับนักศึกษา
2. ต้นแบบที่สร้างบนเอฟพีจีเอเป็นต้นแบบบนห้องปฏิบัติการ เนื่องจากชุดพัฒนาเอฟพีจีเอที่ใช้งานมีขนาดเล็ก และมีข้อจำกัดในเรื่องความเร็ว

**1.6 ประโยชน์ที่คาดว่าจะได้รับ**

1. ได้ต้นแบบระดับห้องปฏิบัติการของตัวเร่งความเร็วเชิงลึกบนเอฟพีจีเอ
2. สามารถนำสถาปัตยกรรมของตัวเร่งความเร็วเชิงลึก ไปใช้งานในระบบสมองกลฝังตัวได้

# บทที่ 2
# ทฤษฎีพื้นฐานและวรรณกรรมที่เกี่ยวข้อง

ในส่วนของการทบทวนวรรณกรรมจะทำการนำเสนอความรู้ทฤษฎีพื้นฐานและวรรณกรรมที่เกี่ยวข้อง กับตัวเร่งความเร็วการเรียนรู้เชิงลึกโดยเนื้อหาในส่วนนี้ประกอบด้วย การเรียนรู้เชิงลึก, โครงข่ายประสาทเทียมคอนโวลูชัน, การคอนโวลูชันด้วยอัลกอริทึมของวิโนกราด, การคำนวณแบบจุดทศนิยมคงที่, ระบบเลข Canonical Signed Digit, การควอนไทซ์ (Quantization), Field-Programmable Gate Array (FPGA) และ วรรณกรรมที่เกี่ยวข้อง

## 2.1 การเรียนรู้เชิงลึก

การเรียนรู้เชิงลึก หรือ Deep Learning [1-6] เป็นสาขาหนึ่งของปัญญาประดิษฐ์ (Artificial Intelligence: AI) และการเรียนรู้ของเครื่อง (Machine Learning) ที่ใช้โครงข่ายประสาทเทียมหลายชั้น (Deep Neural Networks: DNN) ในการเรียนรู้จากข้อมูลจำนวนมากและสร้างแบบจำลองที่ซับซ้อน โดย Deep Learning ได้รับความนิยมมากขึ้นเนื่องจากประสิทธิภาพสูงในการแก้ไขปัญหาต่าง ๆ เช่น การรู้จำภาพ การแปลภาษา การวิเคราะห์ข้อมูลเสียง และระบบแนะนำอัจฉริยะ

### องค์ประกอบของ Deep Learning

Deep Learning อาศัยโครงข่ายประสาทเทียมที่มีหลายชั้น ซึ่งประกอบด้วยส่วนสำคัญดังนี้:

- โหนด (Neuron): หน่วยพื้นฐานของโครงข่ายที่ได้รับอินพุตจากแหล่งข้อมูลหรือโหนดก่อนหน้า และผ่านฟังก์ชันกระตุ้น (Activation Function) เพื่อส่งต่อผลลัพธ์ไปยังโหนดถัดไป
- ค่าถ่วงน้ำหนัก (Weights): ค่าพารามิเตอร์ที่ใช้ในการกำหนดความสำคัญของสัญญาณที่ส่งผ่านโครงข่าย
- ฟังก์ชันกระตุ้น (Activation Function): เช่น Sigmoid, ReLU (Rectified Linear Unit), และ Tanh ใช้เพื่อเพิ่มความไม่เชิงเส้นให้กับโมเดล
- การถ่ายทอดข้อมูล (Forward Propagation): กระบวนการส่งสัญญาณจากอินพุตผ่านเครือข่ายไปยังเอาต์พุต
- การย้อนกลับปรับปรุงน้ำหนัก (Backpropagation): เทคนิคที่ใช้ร่วมกับอัลกอริธึมการปรับค่าพารามิเตอร์ เช่น Gradient Descent เพื่อปรับปรุงค่าถ่วงน้ำหนักให้เหมาะสมกับข้อมูลฝึกสอน

**ประเภทของโครงข่ายประสาทเทียมใน Deep Learning**

- Fully Connected Neural Networks (FCNN): โครงข่ายที่ทุกโหนดของแต่ละชั้นเชื่อมโยงกันกับทุกโหนดในชั้นถัดไป ใช้ในการจำแนกประเภทข้อมูลและปัญหาต่าง ๆ ที่ต้องการโมเดลที่มีโครงสร้างทั่วไป
- Convolutional Neural Networks (CNN): โครงข่ายที่ได้รับการออกแบบมาเพื่อประมวลผลข้อมูลภาพโดยเฉพาะ ใช้ Convolutional Layers และ Pooling Layers เพื่อลดขนาดของภาพและดึงคุณลักษณะที่สำคัญออกมา
- Recurrent Neural Networks (RNN): โครงข่ายที่เหมาะกับข้อมูลที่เป็นลำดับ (Sequential Data) เช่น ข้อความและเสียง พัฒนาไปเป็น Long Short-Term Memory (LSTM) และ Gated Recurrent Units (GRU) เพื่อแก้ปัญหาการลืมข้อมูลในลำดับยาว
- Transformer Models: โครงข่ายที่ใช้ Attention Mechanism ในการวิเคราะห์ลำดับข้อมูล เช่น BERT และ GPT ซึ่งเป็นพื้นฐานของโมเดลการประมวลผลภาษาธรรมชาติสมัยใหม่

**อัลกอริธึมการฝึกสอนและเทคนิคที่สำคัญ**

- Optimization Algorithms: เช่นStochastic Gradient Descent (SGD), Adam, และ RMSprop ใช้ในการปรับค่าพารามิเตอร์ของโมเดลให้เหมาะสม
- Loss Functions: เช่น Mean Squared Error (MSE) สำหรับปัญหาการพยากรณ์ และ Cross-Entropy Loss สำหรับปัญหาการจำแนกประเภท
- Regularization Methods: เช่น Dropout และ Batch Normalization ช่วยลดการเกิด Overfitting

**การประยุกต์ใช้ Deep Learning**

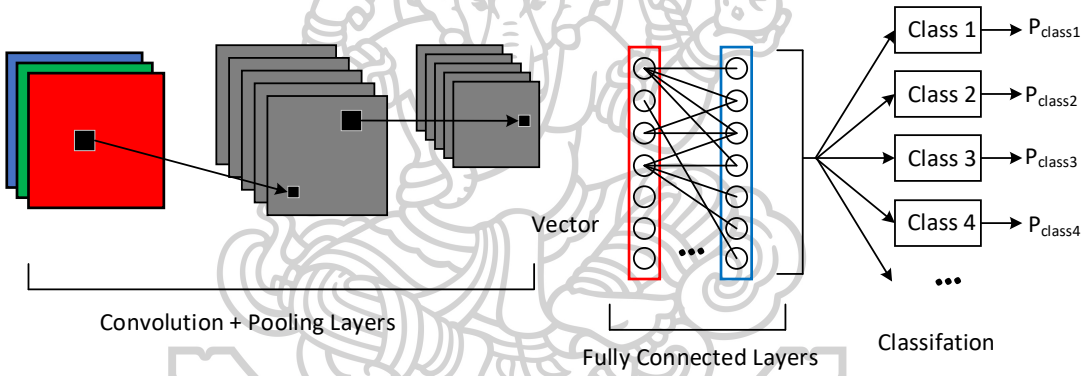
Deep Learning มีการนำไปใช้ในหลากหลายสาขา เช่น:

- การรู้จำภาพ (Image Recognition): ใช้ในระบบเฝ้าระวัง กล้องอัจฉริยะ และการวิเคราะห์ภาพทางการแพทย์
- การประมวลผลภาษาธรรมชาติ (Natural Language Processing - NLP): ใช้ในการแปลภาษา การสร้างข้อความอัตโนมัติ และการวิเคราะห์อารมณ์

- ระบบแนะนำ (Recommendation Systems): ใช้ในแพลตฟอร์มสตรีมมิงและอีคอมเมิร์ซ เช่น Netflix และ Amazon
- การขับขี่อัตโนมัติ (Autonomous Driving): ใช้ในการวิเคราะห์ภาพจากกล้องและเซนเซอร์ของรถยนต์ไร้คนขับ

## 2.2 โครงข่ายประสาทเทียมคอนโวลูชัน

Convolutional neural network (CNN) [23] เป็นโครงข่ายประสาทที่มีไอเดียในการแก้ปัญหาทางด้านรูปภาพ ซึ่งเป็นไอเดียที่ปฏิวัติวงการ Image Recognition การทำงานแบบคร่าวๆ มีรูปมาใช้ นำรูปมาเข้ากระบวนการ Convolution เพื่อเพิ่มรายละเอียด ให้กับรูป Input จากนั้นก็ผ่านกระบวนการ Polling (มีหลายแบบ Average, Max) จากนั้นก็นำ output ที่ได้ ส่งต่อไปให้กับโมเดล Neural Network เพื่อเรียนรู้ปรับ Weight




ภาพที่ 1: องค์ประกอบของ Convolution neural network

แนวคิดของ CNN นั้นค่อนข้างเป็นแนวคิดที่ดีมาก แต่สิ่งที่ซับซ้อนของมันคือระบบการคำนวณที่สอดคล้องกับ Concept ของมันเองและต้องมีคณิตศาสตร์มารองรับ โดยการคำนวณตามแนวคิดนี้ใช้หลักการเดียวกันกับ คอนโวลูชันเชิงพื้นที่ (Spatial Convolution) ในการทำงานด้าน Image Processing การคำนวณนี้จะเริ่มจากการกำหนดค่าใน ตัวกรอง (filter) หรือ เคอร์เนล (kernel) ที่ช่วยดึงคุณลักษณะที่ใช้ในการรู้จำวัตถุออก โดยปกติตัวกรอง/เคอร์เนลอันหนึ่งจะดึงคุณลักษณะที่สนใจออกมาได้หนึ่งอย่าง เราจึงจำเป็นต้องตัวกรองหลายตัวกรองด้วย เพื่อหาคุณลักษณะทางพื้นที่หลายอย่างประกอบกัน

**ลักษณะของ Filter**

สำหรับ Filter ของภาพดิจิทัลนั้น โดยปกติแล้วจะเป็นตารางสองมิติที่มีขนาดตามพื้นที่ย่อยๆที่จะพิจารณา ถ้าเราต้องการหาเส้นตรงทะแยงสีขาว ตัวกรองอาจจะอยู่ในลักษณะดังรูปที่ 2

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

ภาพที่ 2: ตัวกรอง 3x3 สำหรับหาเส้นทแยงสีขาว

ตำแหน่งตรงกลางที่มีกรอบสีฟ้าคือ Anchor ที่เอาไว้ทาบบนพิกเซลของภาพข้อมูลเข้า ตัวกรองจะถูกทาบลงในพิกเซลแรกของภาพข้อมูลเข้า จากนั้นจะถูกเลื่อนไปทาบบนพิกเซลอื่นในภาพทีละพิกเซลจนครบทุกพิกเซลในภาพ อาจจะไม่ทาบตัวกรองบนพิกเซลที่อยู่ใกล้กรอบภาพ เพราะตัวกรองจะล้นออกไปนอกภาพ เมื่อเลื่อนตัวกรองไปเรื่อยๆจนครบทุกพิกเซลที่สามารถเลื่อนได้ในภาพ สิ่งที่ได้นั้นจะเป็นสิ่งที่เรียกว่า ผังคุณลักษณะ (feature map) ดังรูปที่ 3

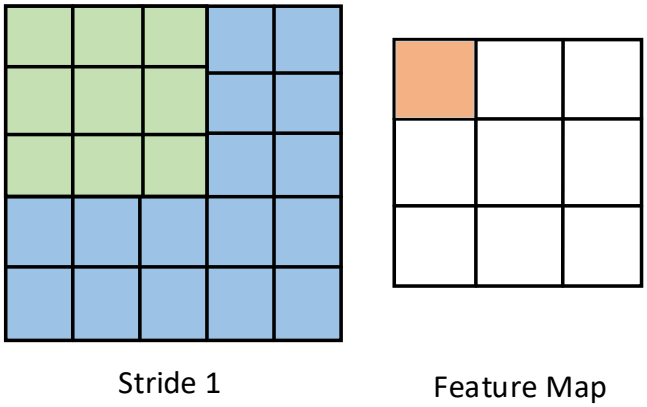
| | | | | |
|---|---|---|---|---|
| 1x1 | 1x0 | 1x1 | 0 | 0 |
| 0x0 | 1x1 | 1x0 | 1 | 0 |
| 0x1 | 0x0 | 1x1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

| | | |
|---|---|---|
| 4 | | |
| | | |
| | | |

ภาพที่ 3: การทำ Convolution

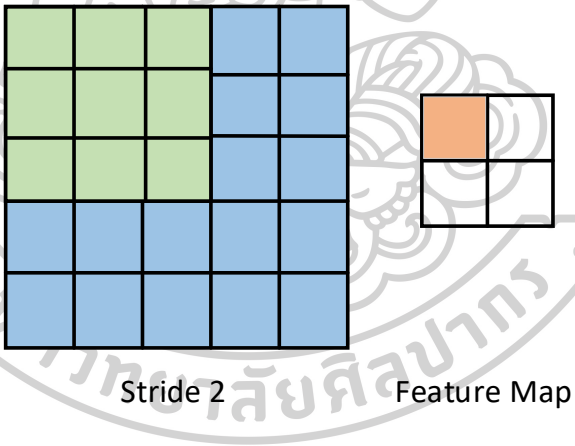
**Stride และ Padding**

Stride เป็นตัวกำหนดว่าเราจะเลื่อนตัวกรอง (filter) ไปด้วย Step เท่าไร (ตัวอย่างด้านล่างกำหนด Stride เท่ากับ 1)



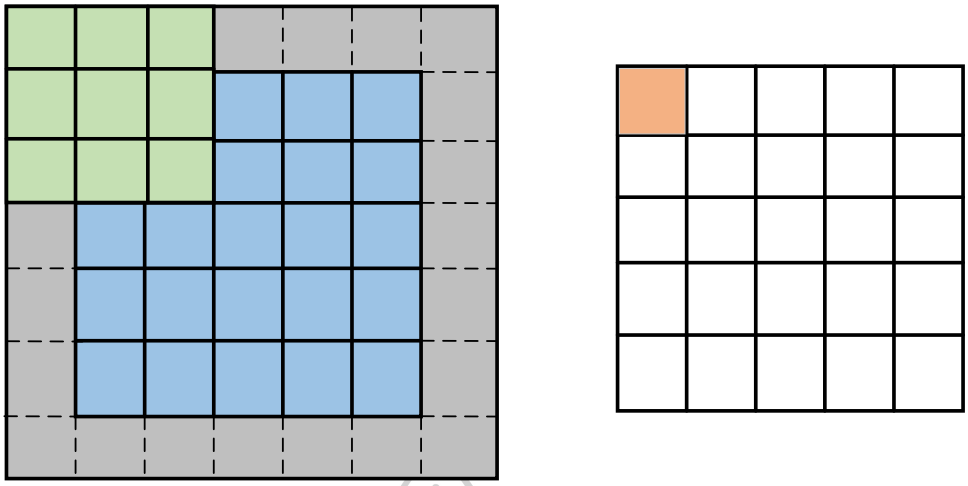

ภาพที่ 4: Stride เท่ากับ 1

สามารถกำหนดค่าของ Stride ให้มากขึ้นได้ ถ้าต้องการให้การคำนวนหาคุณลักษณะมีพื้นที่ทับซ้อนกันน้อยขึ้น แต่อย่างไรก็ตามการกำหนดค่าของ Stride ที่มากขึ้นจะทำให้เราได้ผังคุณลักษณะ (feature map) ที่มีขนาดเล็กลง




ภาพที่ 5: Stride เท่ากับ 2

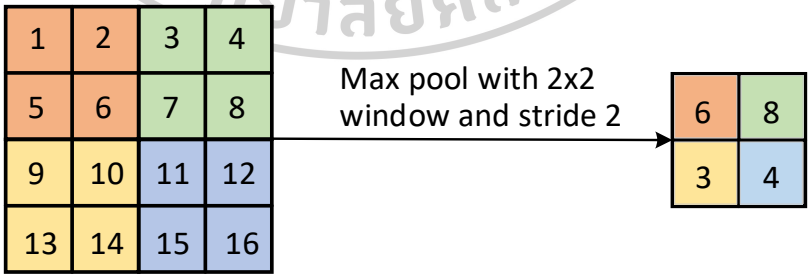
Padding จากรูปด้านล่างเราจะพื้นที่สีเทารอบๆ Input พื้นที่เหล่านี้เป็นพื้นที่ที่เรามักเติมเข้าไป โดยอาจจะเป็นเติม 0 หรือค่าต่างๆเข้าไป เพื่อให้เวลาในการทำ CNN นั้น Feature Map ที่ได้ยังคงมีขนาดเท่ากับ Input




ภาพที่ 6: การ Padding

**Pooling**

Pooling คือความสามารถในการย่อรูปแบบหนึ่ง ซึ่งมีสองประเภทหลักที่นิยมกันคือ max pooling และ mean pooling Max Pooling เป็นตัวกรองแบบหนึ่งที่หาค่าสูงสุดในบริเวณที่ตัวกรองทาบอยู่มาเป็นผลลัพธ์ โดยเราจะเตรียมตัวกรองในลักษณะเดียวกับการทำ Feature Extraction ของ CNN มาทาบบนข้อมูลแล้วเลือกค่าที่สูงที่สุดบนตัวกรองนั้นมาเป็นผลลัพธ์ใหม่ และจะเลื่อนตัวกรองไปตาม Stride ที่กำหนดไว้ โดยขนาดตัวกรองของการทำ max pooling จะนิยมเรียกกันว่า pool size




ภาพที่ 7: การทำ Max pooling

**2.3 การคอนโวลูชันด้วยอัลกอริทึมของวิโนกราด**

Winograd Convolution [24] เป็นอัลกอริทึมที่ออกแบบมาเพื่อเพิ่มประสิทธิภาพของการคำนวณ Convolution ซึ่งเป็นกระบวนการสำคัญในงานประมวลผลภาพ (Image Processing) และ

การเรียนรู้เชิงลึก (Deep Learning) โดยเฉพาะใน Convolution อัลกอริทึม Winograd ได้รับการตั้งชื่อตาม Shmuel Winograd ผู้พัฒนาวิธีการนี้ในปี 1980 เพื่อลดจำนวนการคูณเชิงเมทริกซ์ ซึ่งเป็นกระบวนการที่มีต้นทุนสูงในเชิงคำนวณ

Winograd อาศัยความสัมพันธ์ทางคณิตศาสตร์ ในการแปลง Convolution ให้เป็นกระบวนการที่ต้องใช้การคูณน้อยลง โดยเฉพาะอย่างยิ่งในการประมวลผลฟิลเตอร์ขนาดเล็ก เช่น 3x3 หรือ 5x5 ซึ่งเป็นขนาดฟิลเตอร์ที่นิยมใช้ใน CNNs

**หลักการทำงาน**

Winograd Algorithm เป็นอัลกอริทึมที่ออกแบบมาเพื่อเพิ่มประสิทธิภาพของกระบวนการ Convolution โดยลดจำนวนการคูณ (Multiplications) ซึ่งเป็นขั้นตอนที่มีต้นทุนสูงในแง่ของการประมวลผล ทำให้เหมาะสำหรับการใช้งานใน Deep Learning โดยเฉพาะการประมวลผลใน Convolutional Neural Networks (CNNs)

หลักการสำคัญของ Winograd คือการแปลง Convolution ให้เป็นปัญหาทางคณิตศาสตร์ในรูปของการคำนวณที่ใช้การบวกแทนการคูณ โดยมีสมการพื้นฐานดังนี้

$$Y=A^{T}\cdot[(G\cdot g\cdot G^{T})\odot(B^{T}\cdot d\cdot B)]\cdot A \qquad (1)$$

โดยที่ Y คือ ผลลัพธ์สุดท้ายของคอนโวลูชัน

g คือ ฟิลเตอร์ 3×3

d คือ อินพุต 4×4

G, B, และ A เป็นเมทริกซ์การแปลงที่ใช้ในขั้นตอนต่าง ๆ

โดยที่ ค่าคงที่ของเมทริกซ์ G, B, และ A คือ

$$G=\begin{bmatrix}1 & 0 & 0\\ 0.5 & 0.5 & 0.5\\ 0.5 & -0.5 & 0.5\\ 0 & 0 & 1\end{bmatrix}$$

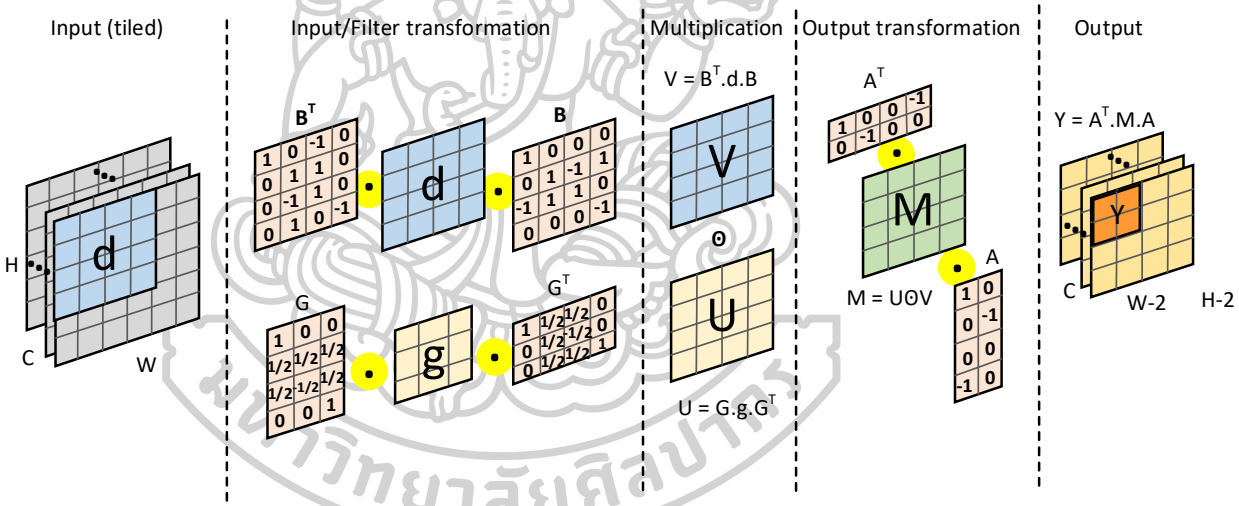
เมทริกซ์ G ใช้สำหรับแปลงฟิลเตอร์จาก spatial domain ไปสู่ Winograd domain โดยอาศัยหลักการของการลดขนาดของการคูณที่ไม่จำเป็นในกระบวนการคอนโวลูชัน

$$B = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

เมทริกซ์ B ใช้แปลงอินพุตจาก spatial domain ไปยัง Winograd domain ซึ่งจะช่วยลดการคูณระหว่างอินพุตและฟิลเตอร์

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

เมทริกซ์ **A** ใช้ในการแปลงกลับจาก Winograd domain กลับไปยัง spatial domain เพื่อให้ได้ผลลัพธ์ของการคอนโวลูชัน




ภาพที่ 8: ขั้นตอนการทำงานของ Winograd convolution

## 2.4 การคำนวณแบบจุดทศนิยมคงที่

หัวข้อนี้อธิบายวงจรการออกแบบทั่วไปในการพัฒนาแอปพลิเคชันประมวลผลสัญญาณ ขั้นตอนแรกของวงจรคือการกำหนดข้อกำหนดและสเปคของระบบ ข้อกำหนดเหล่านี้มักระบุอัตราการสุ่มตัวอย่าง (sampling rate) ตัวชี้วัดเชิงปริมาณของประสิทธิภาพระบบ และพารามิเตอร์เฉพาะของแอปพลิเคชัน ข้อกำหนดเหล่านี้เป็นข้อจำกัดที่ทำให้นักออกแบบต้องสำรวจทางเลือกและอัลกอริทึมต่างๆ เพื่อให้ตรงตามข้อกำหนดนั้นในรูปแบบที่ประหยัดที่สุด

การสำรวจอัลกอริทึมมักดำเนินการผ่าน MATLAB ซึ่งมีชุดเครื่องมือ ไลบรารี และฟังก์ชันมากมาย หลังจากการพัฒนาและวิเคราะห์อัลกอริทึมใน MATLAB แล้ว โค้ดมักจะถูกแปลงเป็นภาษาระดับสูง เช่น C/C++ หรือ C#

หัวข้อนี้จึงต้องให้ความสำคัญกับระบบเลข โดยจะอธิบายการแทนค่าตัวเลขที่มีเครื่องหมายในรูปแบบทูส์คอมพลีเมนต์ (Two's complement) ในการแทนค่ารูปแบบนี้ บิตที่มีค่าสูงสุด (MSB) จะมีค่าน้ำหนักเป็นลบ ส่วนบิตที่เหลือมีค่าน้ำหนักเป็นบวก แม้ว่าคณิตศาสตร์แบบทูส์คอมพลีเมนต์จะช่วยให้การบวกและลบง่ายขึ้น (เนื่องจากสามารถทำการลบได้โดยใช้การบวก) แต่ค่าน้ำหนักลบของบิตเครื่องหมายก็มีผลกระทบต่อการคูณและการเลื่อนบิต

เนื่องจาก MSB ของเลขที่มีเครื่องหมายมีน้ำหนักเป็นลบ การคูณจึงต้องมีการจัดการที่แตกต่างกันขึ้นอยู่กับประเภทของตัวถูกดำเนินการ ซึ่งสามารถแบ่งได้เป็น 4 กรณี ได้แก่ การคูณระหว่างตัวเลขไม่มีเครื่องหมายกับตัวเลขไม่มีเครื่องหมาย, ตัวเลขมีเครื่องหมายกับตัวเลขไม่มีเครื่องหมาย, ตัวเลขไม่มีเครื่องหมายกับตัวเลขมีเครื่องหมาย และตัวเลขมีเครื่องหมายกับตัวเลขมีเครื่องหมาย บทนี้จะอธิบายการคูณในรูปแบบทูส์คอมพลีเมนต์เพื่อแสดงความแตกต่างของแต่ละกรณี

นอกจากนี้ ยังมีการอธิบายการสเกลค่าของตัวเลขที่มีเครื่องหมาย เนื่องจากเป็นสิ่งที่จำเป็นบ่อยครั้งเมื่อนำอัลกอริทึมไปใช้งานในรูปแบบค่าคงที่ (fixed-point format)

2.4.1 ระบบเลขทศนิยมลอยตัว และเลขทศนิยมคงตัว

จากมุมมองการประมวลผลสัญญาณ อัลกอริธึมสามารถนำไปใช้ในรูปแบบของเลขจุดทศนิยมแบบคงที่ (fixed point) หรือจุดทศนิยมแบบลอยตัว (floating point) รูปแบบจุดทศนิยมแบบลอยตัวจะเก็บตัวเลขในลักษณะของมันทิสซา (mantissa) และเลขชี้กำลัง (exponent) ฮาร์ดแวร์ที่รองรับรูปแบบจุดทศนิยมแบบลอยตัวจะทำการคำนวณแต่ละครั้งและทำการปรับสเกลมันทิสซาและอัปเดตเลขชี้กำลังโดยอัตโนมัติเพื่อให้ผลลัพธ์พอดีกับจำนวนบิตที่กำหนดในลักษณะเฉพาะ การดำเนินการเหล่านี้ทำให้ฮาร์ดแวร์ที่รองรับจุดทศนิยมแบบลอยตัวมีต้นทุนที่สูงกว่าในแง่ของพื้นที่และพลังงานเมื่อเทียบกับฮาร์ดแวร์ที่รองรับจุดทศนิยมแบบคงที่

ในฮาร์ดแวร์ที่ใช้จุดทศนิยมแบบคงที่ (fixed point) หลังจากทำการคำนวณ ฮาร์ดแวร์จะไม่ติดตามตำแหน่งของจุดทศนิยม และจะปล่อยให้ความรับผิดชอบนี้ตกอยู่กับนักพัฒนา จุดทศนิยมจะ

ถูกกำหนดไว้ล่วงหน้าสำหรับแต่ละตัวแปร การกำหนดจุดทศนิยมทำให้ตัวแปรสามารถรับค่าได้ในช่วงที่กำหนดเท่านั้น เมื่อค่าของตัวแปรเกินขอบเขตนี้ ผลลัพธ์จากการคำนวณจะสูญหายหรือถูกทำลาย ซึ่งเรียกว่า overflow (การล้นของข้อมูล)

มีหลายวิธีในการจัดการกับการล้นข้อมูลในคณิตศาสตร์แบบจุดทศนิยมคงที่ การจัดการการล้นข้อมูลจะต้องทำการจำกัดผลลัพธ์ให้อยู่ในค่าบวกสูงสุดหรือค่าลบต่ำสุดที่สามารถกำหนดให้กับตัวแปรในรูปแบบจุดทศนิยมคงที่ ซึ่งจะทำให้ประสิทธิภาพหรือความแม่นยำลดลง นักพัฒนาสามารถกำหนดตำแหน่งของจุดทศนิยมสำหรับตัวแปรทั้งหมด เพื่อให้การจัดวางดังกล่าวป้องกันการล้นข้อมูลได้ วิธีนี้จะต้องการให้ผู้ออกแบบทำการทดสอบกับข้อมูลทุกชนิดและสังเกตช่วงค่าของตัวแปรทั้งหมดในกระบวนการจำลอง การรู้ช่วงค่าของตัวแปรทั้งหมดในอัลกอริธึมทำให้การกำหนดตำแหน่งจุดทศนิยมเพื่อหลีกเลี่ยงการล้นข้อมูลเป็นเรื่องง่ายมาก

การนำอัลกอริธึมการประมวลผลสัญญาณและการสื่อสารไปใช้งานบนโปรเซสเซอร์ที่ใช้จุดทศนิยมคงที่เป็นงานที่ตรงไปตรงมา เนื่องจากโปรเซสเซอร์แบบจุดทศนิยมคงที่มีการใช้พลังงานต่ำและราคาถูกกว่า โปรเซสเซอร์ที่ใช้จุดทศนิยมคงที่ (DSP) จึงพบได้บ่อยในแอปพลิเคชันฝังตัวต่างๆ ในขณะที่โปรเซสเซอร์ที่ใช้จุดทศนิยมแบบลอยตัว (floating point) มักใช้รูปแบบ 32 บิต โปรเซสเซอร์แบบจุดทศนิยมคงที่มักใช้รูปแบบ 16 บิต ซึ่งทำให้การออกแบบจุดทศนิยมคงที่ใช้หน่วยความจำน้อยกว่า หน่วยความจำบนชิปมักใช้พื้นที่ซิลิคอนมากที่สุด ซึ่งส่งผลให้ต้นทุนของระบบลดลง การออกแบบจุดทศนิยมคงที่จึงถูกใช้ในโซลูชันด้านมัลติมีเดียและการสื่อสารโทรคมนาคม

แม้ว่าตัวเลือกแบบจุดทศนิยมลอยตัวจะมีราคาแพงกว่า แต่ก็ให้ความแม่นยำมากกว่า จุดทศนิยมจะเคลื่อนที่ไปมาและคำนวณใหม่ทุกครั้งที่ทำการคำนวณ ดังนั้นฮาร์ดแวร์จะตรวจจับการล้นข้อมูลโดยอัตโนมัติและปรับตำแหน่งจุดทศนิยมโดยการเปลี่ยนแปลงเลขชี้กำลัง ซึ่งช่วยกำจัดข้อผิดพลาดจากการล้นข้อมูลและลดความไม่แม่นยำที่เกิดจากการปัดเศษที่ไม่จำเป็น

2.4.2 ระบบเลข 2’complement

ในระบบเลขฐานสองแบบทูส์คอมพลีเมนต์ (Two’s complement) [25] การแทนค่าตัวเลขที่มีเครื่องหมาย (signed number) ถูกกำหนดโดย $a = a_{N-1}a_{N-2}\ldots a_2a_1a_0$ โดยที่บิตที่มีค่าสูงสุด (MSB: Most Significant Bit) $a_{N-1}$ แทนเครื่องหมายของตัวเลข ถ้า a เป็นบวก (positive) บิตเครื่องหมาย $a_{N-1} = 0$ และบิตที่เหลือแสดงขนาดของตัวเลข (magnitude) รูปแบบการคำนวณของ a เมื่อ $a >= 0$ คือ

$$a = \sum_{i=0}^{N-2}(a_i 2^i) \qquad (2)$$

ดังนั้น การแทนค่าตัวเลขบวกในระบบทูส์คอมพลีเมนต์ที่มี N บิต จะเทียบเท่ากับค่าของตัวเลขที่ไม่มีเครื่องหมาย (unsigned) ที่มี (N-1) บิต ในระบบนี้ เลข 0 ถูกพิจารณาว่าเป็นเลขบวก และช่วงของค่าที่เป็นบวกที่สามารถแทนได้ใน N บิต คือ 0 ถึง ($2^{N-1}$ - 1)

สำหรับตัวเลขลบ ค่าบิต MSB $a_{N-1}$ = 1 จะมีน้ำหนักเชิงลบ ในขณะที่บิตที่เหลือมีน้ำหนักเป็นบวกนิพจน์ทั่วไปสำหรับตัวเลขลบในรูปแบบทูส์คอมพลีเมนต์คือ

$$a = -2^{N-1} + \sum_{i=0}^{N-2}(a_i 2^i) \text{ เมื่อ } a < 0 \qquad (3)$$

เมื่อรวมกรณีของตัวเลขบวกและลบเข้าด้วยกัน จะได้รูปแบบทั่วไปของเลขในทูส์คอมพลีเมนต์

$$a = -2^{N-1}a_{N-1} + \sum_{i=0}^{N-2}(a_i 2^i) \qquad (4)$$

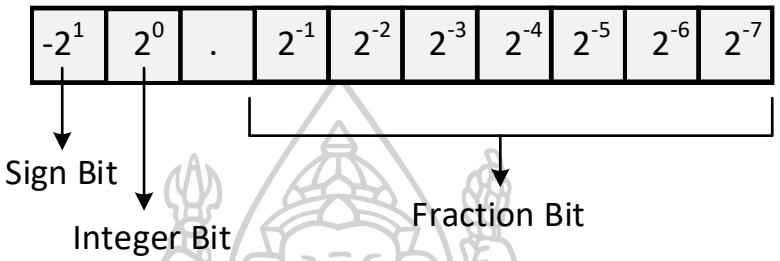
โดยค่า $a_{N-1}$ จะกำหนดว่าค่า a เป็นบวกหรือลบ

2.4.3 Qn.m format for fixed-point arithmetic

ระบบประมวลผลสัญญาณและระบบสื่อสารส่วนใหญ่มักถูกพัฒนาในขั้นต้นโดยใช้คณิตศาสตร์แบบจุดลอยตัวความแม่นยำสองเท่า (double-precision floating point arithmetic) ผ่านเครื่องมืออย่าง MATLAB ขณะดำเนินการพัฒนาอัลกอริทึม เป้าหมายหลักของนักพัฒนาคือการทำความเข้าใจและนำฟังก์ชันการทำงานของอัลกอริทึมมาใช้อย่างถูกต้อง จากนั้นโค้ด MATLAB จะถูกแปลงเป็นโค้ด C/C++ แบบจุดลอยตัว ซึ่งโดยทั่วไปแล้วโค้ด C++ จะทำงานได้เร็วกว่าโค้ด MATLAB มาก นอกจากนี้ กระบวนการแปลงโค้ดยังช่วยให้นักพัฒนามีความเข้าใจอัลกอริทึมมากขึ้น เนื่องจากอาจมีการใช้ฟังก์ชันต่างๆ จาก MATLAB toolboxes ความเข้าใจในอัลกอริทึมเหล่านี้เป็นสิ่งสำคัญอย่างยิ่งในการคำนวณตัวเลขในซอฟต์แวร์หรือฮาร์ดแวร์สำหรับอุปกรณ์ฝังตัว หลังจากที่ได้ประสิทธิภาพตามที่ต้องการจากอัลกอริทึมแบบจุดลอยตัว (floating point) แล้ว การนำไปใช้งานจะถูกแปลงเป็นรูปแบบจุดตรึง (fixed point) โดยตัวแปรและค่าคงที่ที่ใช้ในการจำลองแบบจุดลอยตัวจะถูกแปลงเป็นรูปแบบ Qn.m fixed point [25] ซึ่งเป็นระบบตัวเลขที่มีตำแหน่งจุดทศนิยมคงที่สำหรับแทนค่าตัวเลขแบบจุดลอยตัว

รูปแบบ Qn.m สำหรับตัวเลขที่มี N บิต จะกำหนดให้ n บิต อยู่ทางซ้ายของจุดทศนิยม (binary point) และ m บิต อยู่ทางขวาของจุดทศนิยม ในกรณีของตัวเลขที่มีเครื่องหมาย (signed numbers) MSB ใช้แทนเครื่องหมายของตัวเลข และมีค่าน้ำหนักเชิงลบ ตัวเลขแบบทูส์คอมพลีเมนต์ในรูปแบบ Qn.m สามารถเขียนเป็น $b = b_{n-1}b_{n-2}\ldots b_1b_0 \,.\, b_{-1}b_{-2}\ldots b_{-m}$ ซึ่งมีค่าจุดลอยตัวที่เทียบเท่ากับ

$$-b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + b_{-m}2^{-m}$$




ภาพที่ 9: รูปแบบการเขียน Qn.m

2.4.4 CSD number system

ระบบเลข CSD (Canonical Signed Digit) [26] เป็นระบบที่ใช้แทนเลขฐานสองโดยใช้สัญลักษณ์ -1, 0, และ 1 แทนการใช้ 0 หรือ 1 ในการเขียนเลขฐานสองแบบปกติ ซึ่งช่วยลดจำนวนการดำเนินการทางคณิตศาสตร์ เช่น การคูณหรือการบวก/ลบ เนื่องจากเลขที่อยู่ติดกันไม่สามารถเป็น 1 หรือ -1 พร้อมกันได้ ทำให้ประหยัดทรัพยากรและเพิ่มความเร็วในการคำนวณ เช่น ในการประมวลผลสัญญาณหรือการออกแบบวงจรดิจิตอล รายละเอียดจะอธิบายในหัวถัดไป

**2.5 ระบบเลขแบบ Canonical Signed Digit (CSD)**

Canonical Signed Digit (CSD) [26] เป็นเทคนิคที่ใช้ในการแทนค่าตัวเลขในระบบเลขฐานสองที่ช่วยลดจำนวนบิตที่เป็น 1 ซึ่งมีผลต่อประสิทธิภาพการคำนวณในฮาร์ดแวร์ดิจิตอล เช่น วงจรตรรกะและหน่วยประมวลผล โดย CSD ใช้หลักการแทนค่าตัวเลขในรูปแบบ (-1, 0, 1) แทนที่จะใช้เฉพาะ (0,1) ในระบบเลขฐานสองทั่วไป โดยแทนตัวเลข -1 คือ 10, 0 คือ 00 และ 1 คือ 01 ซึ่งช่วยลดจำนวนการดำเนินการที่ต้องใช้ในการคูณและบวกในฮาร์ดแวร์ การแทนค่าด้วย CSD ช่วยลดจำนวนบิตที่เป็น 1 ซึ่งทำให้ลดการใช้พลังงานและเพิ่มประสิทธิภาพของระบบ

CSD ถูกนำไปใช้ในหลายด้านของการออกแบบฮาร์ดแวร์ดิจิตอล เช่น FPGA และ ASIC เนื่องจากสามารถลดความซับซ้อนของวงจรดิจิตอลและลดต้นทุนในการผลิต ในระบบที่ต้องการ

คำนวณค่าคงที่ เช่น ตัวกรองดิจิตอล (Digital Filters) และตัวเร่งฮาร์ดแวร์ (Hardware Accelerators) การใช้ CSD ทำให้สามารถลดจำนวนทรัพยากรที่ใช้ในวงจรดิจิตอลได้ ซึ่งมีความสำคัญอย่างยิ่งต่อการออกแบบระบบที่ต้องการประสิทธิภาพสูงและการใช้พลังงานต่ำ

ด้วยข้อดีของ CSD ในการลดจำนวนบิตที่เป็น 1 และปรับปรุงประสิทธิภาพของวงจร ทำให้เป็นเทคนิคที่มีประโยชน์อย่างมากในการออกแบบฮาร์ดแวร์ดิจิตอล โดยเฉพาะในระบบที่ต้องการการคำนวณแบบคูณซ้ำ ๆ หรือการดำเนินการทางเลขคณิตที่มีความซับซ้อนสูง

**อัลกอริธึมในการแปลงจากเลขฐานสองเป็น CSD**

มีหลายอัลกอริธึมที่สามารถใช้ในการแปลงเลขฐานสองเป็น CSD วิธีการที่ง่ายและเหมาะสมสำหรับการคำนวณด้วยมือ คือการค้นหาลำดับของเลข 1 ที่อยู่ติดกันและตามด้วยเลข 0 จาก LSB (Least Significant Bit) ไปยัง MSB (Most Significant Bit) แล้วแทนที่ด้วยรูปแบบ CSD ที่เหมาะสม ตัวอย่างเช่น ลำดับ 0111 ในเลขฐานสอง สามารถแทนค่าเป็น $100\bar{1}$ ในระบบ CSD

ขั้นตอนนี้อาจต้องทำซ้ำหลายครั้ง เนื่องจากอาจมีลำดับของเลข 1 อื่น ๆ หรืออาจเกิดลำดับของเลข 1 ใหม่ขึ้นจากการแปลงก่อนหน้า

ตัวอย่างที่ 1: แปลง 01011101 เป็น CSD

1. ค้นหาลำดับ 0111 จาก LSB ไปยัง MSB ซึ่งสามารถแทนค่าเป็น $100\bar{1}$
2. แทนที่ 0111 ด้วย $100\bar{1}$ ทำให้ได้เลข $0110\bar{0}101$
3. ค้นหาใหม่ พบลำดับ 011 ซึ่งสามารถแทนค่าเป็น $10\bar{1}$
4. แทนที่ 011 ด้วย $10\bar{1}$ ได้ผลลัพธ์สุดท้ายเป็น $1\bar{0}10\bar{0}101$

ตัวอย่างที่ 2: แปลง 1010111 เป็น CSD

1. ค้นหาลำดับ 0111 และแทนที่ด้วย $100\bar{1}$ ได้เลข $101100\bar{1}$
2. ค้นหาใหม่ พบลำดับ 011 และแทนที่ด้วย $10\bar{1}$ ได้เลข $11\bar{0}100\bar{1}$
3. พบลำดับ 11 ซึ่งต้องการเพิ่ม 0 ทางซ้ายมือของ $11\bar{0}100\bar{1}$ เพื่อไม่ให้ค่าของเลขเปลี่ยนแปลง
4. แทนที่ 011 ด้วย $10\bar{1}$ ได้ผลลัพธ์สุดท้ายเป็น $10\bar{1}0\bar{1}00\bar{1}$

วิธีนี้ช่วยลดจำนวนบิตที่เป็น 1 ในการแทนค่าตัวเลข ส่งผลให้การคำนวณทางฮาร์ดแวร์มีประสิทธิภาพมากขึ้น โดยเฉพาะในการดำเนินการคูณที่สามารถแทนที่ด้วยการเลื่อนบิตและการบวก-ลบแทน

2.5.1 CSD Multiplication

อย่างที่ได้กล่าวไปแล้ว การคูณโดยใช้ CSD [27] นั้นเร็วกว่าการคูณแบบเลขฐานสองทั่วไป อย่างไรก็ตาม ก่อนที่จะใช้ CSD เราจำเป็นต้องแปลงตัวเลขที่ต้องการคูณจากเลขฐานสองไปเป็น CSD ก่อน ในกรณีที่ตัวคูณเป็นค่าคงที่ เราสามารถคำนวณค่า CSD ของตัวคูณและนำไปใช้งานได้โดยตรง

ตัวอย่างเช่น สมมติว่าเราออกแบบตัวกรอง FIR และจำเป็นต้องคูณด้วยค่าคงที่ 0.30410913 โดยทั่วไปเราจะใช้ตัวคูณสเกล (scaling factor) และปัดค่าตัวคูณให้เป็นค่าที่สามารถใช้งานได้ ตัวคูณสเกลนี้ถูกเลือกโดยพิจารณาจากขนาดสัมบูรณ์ของค่าสัมประสิทธิ์ที่ใหญ่ที่สุดและจำนวนบิตที่ใช้แทนค่า

สมมติว่าค่าสัมประสิทธิ์ที่ใหญ่ที่สุดของเราคือ 0.30410913 และเราต้องการใช้ตัวแทนเลข 16 บิต เราสามารถเลือกตัวคูณสเกลได้สูงสุดถึง $\frac{2^{15}-1}{0.30410913} \approx 107750$

ถ้าเลือกตัวคูณสเกลเป็น $10^5$ จะได้ค่าคงที่ คือ 30411 ซึ่งตัวเลขฐานสองคือ 0111011011001011 เมื่อแปลงเป็น CSD จะได้ $1000\bar{1}00\bar{1}0\bar{1}0\bar{1}0\bar{1}$

2.5.2 CSD Addition

CSD Addition จะมีความพิเศษ เนื่องจากการบวก หรือการลบตัวเลข CSD ไม่สามารถบวกลบกันได้โดยตรง จำเป็นต้องใช้อัลกอริทึมเข้ามาช่วย อัลกอริทึมที่ใช้คือ Online Bit Adder [28] ซึ่งอัลกอริทึมนี้มีความพิเศษ คือเป็นการคำนวณในส่วนที่เป็น MSB ก่อน แทนที่คำนวณในส่วน LSB ก่อน ทำให้อัลกอริทึมไม่มีการคำนวณตัวทด (Carry)

**2.6 การควอนไทซ์ (Quantization)**

Quantization [29] เป็นกระบวนการที่แปลงข้อมูลที่มีความละเอียดสูงลงมาเป็นข้อมูลที่มีความละเอียดต่ำลง โดยการควอนไทซ์จะลดจำนวนบิตที่ใช้ในการแทนค่าข้อมูล ทำให้การประมวลผลข้อมูลนั้นมีขนาดเล็กลงและสามารถทำงานได้รวดเร็วขึ้นในระบบที่จำกัดทรัพยากร

เช่น การประมวลผลสัญญาณเสียงหรือภาพในระบบดิจิตอล โดยในกรณีของเลข 8 บิตจะใช้การแทนค่าในช่วง 0-255 ($2^8$ - 1) ซึ่งหมายความว่า ข้อมูลที่มีความละเอียดสูงจะถูกแบ่งออกเป็น 256 ระดับ (หรือ 256 ค่าต่างๆ) แทนที่จะแทนค่าด้วยตัวเลขที่มีความแม่นยำสูงกว่า เช่น 16 บิตหรือ 32 บิต กระบวนการควอนไทซ์นี้อาจทำให้ข้อมูลสูญเสียบางส่วนไป แต่จะช่วยเพิ่มประสิทธิภาพในการประมวลผลโดยรวม ตัวอย่างเช่น ในการแปลงสัญญาณเสียงหรือภาพจะทำให้การจัดเก็บและการประมวลผลข้อมูลมีขนาดเล็กลงมากขึ้น ซึ่งมีหลายวิธี เช่น

1. Fixed-Point Quantization

- กำหนดช่วงค่าที่สามารถแทนได้ เช่น [−1,1] หรือ [0,1]

- ใช้จำนวนบิตที่แน่นอนในการแทนค่า เช่น 8-bit, 16-bit

- ใช้สเกลแฟกเตอร์ (scale factor) และออฟเซ็ต (offset) เพื่อนำค่าจาก floating-point มาอยู่ในช่วงของ fixed-point

2. Uniform Quantization

- กำหนดช่วงค่าที่สามารถแทนได้ แล้วแบ่งเป็นระดับย่อย ๆ ที่เท่ากัน

- ตัวอย่างเช่น ถ้าใช้ 8-bit และแทนค่าระหว่าง [−1,1] จะมี $2^8$ = 256 ระดับ

$$Q(x) = \text{round}\left(\boldsymbol{x} * \frac{2^b - 1}{\text{rmax} - \text{rmin}}\right) \quad (5)$$

โดยที่ b คือจำนวนบิต และ rmax ,rmin คือค่าสูงสุดและต่ำสุดที่ต้องการแทน

3. Min-Max Scaling (Linear Quantization)

ถ้ามีค่าทศนิยมในช่วง [xmin ,xmax] และต้องการแมปไปยังช่วง [−128,127] จะได้สมการดังนี้

$$Q(x) = \text{round}\left(\frac{(x - xmin)(127 + 128)}{\text{xmax} - \text{xmin}} - 128\right) \quad (6)$$

**2.7 FPGA (Field-Programmable Gate Array)**

FPGA [30] เป็นเครื่องมือที่ช่วยให้นักออกแบบสามารถพัฒนา และจำลองวงจรดิจิตอลที่ซับซ้อนได้อย่างรวดเร็ว สามารถตรวจสอบการทำงานของการใช้งานจริงได้ อุปกรณ์ต่างๆ ของ

FPGA สามารถรองรับการออกแบบ SoC (system on chip) ซึ่งรวมหน่วยประมวลผลหลัก โมดูลหน่วยความจำ อุปกรณ์ต่อพ่วง I/O และตัวเร่งฮาร์ดแวร์แบบกำหนดเองไว้ในชิปเดียวกัน

**ความสำคัญของ Hardware Description Language (HDL)**

HDL เป็นคำย่อของ Hardware Description Language ซึ่งก็คือภาษาบรรยายฮาร์ดแวร์นั่นเอง HDL เป็นการนำเอาหลักการของภาษาสำหรับใช้เขียนโปรแกรมคอมพิวเตอร์มาช่วยในการออกแบบฮาร์ดแวร์ เนื่องจากความก้าวหน้าในอุตสาหกรรมการผลิตวงจรรวมทำให้ในปัจจุบันวงจรรวมสามารถประกอบด้วยทรานซิสเตอร์ ในระดับล้านตัว ซึ่งเป็นความก้าวหน้าที่เป็นพลังขับเคลื่อนของอุตสาหกรรมคอมพิวเตอร์และเทคโนโลยีสารสนเทศในปัจจุบัน วงจรรวมในระดับล้านทรานซิสเตอร์ประกอบด้วยวงจรดิจิตอลซึ่งมีความซับซ้อนมากทำให้กระบวนการออกแบบจำเป็นต้องพัฒนาตามกระบวนการผลิต ในการออกแบบในอดีตการออกแบบวงจรรวมดิจิตอลโดยออกแบบโดยใช้ทรานซิสเตอร์ต่อเชื่อมทีละตัวนั้นไม่สามารถ ที่จะทำให้ออกแบบผลิตภัณฑ์ให้ทันความต้องการของตลาด (time to market) อีกยังต้องใช้กำลังคนและทรัพยากรทางการเงินจำนวนมากซึ่งไม่เหมาะสมกับ บริษัทขนาดเล็กและบริษัททั่วๆไป HDL จึงเป็นทางเลือกในการออกแบบวงจรรวมดิจิตอล

**ขั้นตอนการออกแบบ FPGA**

การออกแบบวงจรรวมดิจิตอลโดยใช้ ภาษา Verilog HDL นั้นจะมีส่วนคล้ายกันในส่วนของ ASIC คือวงจรที่ต้องส่งไปให้โรงงานเจือสาร กับ FPGA ส่วนสำคัญที่เหมือนกันก็คือ เราใช้ภาษา Verilog HDL ในการออกแบบ และใช้ซอฟต์แวร์สังเคราะห์วงจรเหมือนกัน แต่ในรายละเอียดจะมีบางส่วนที่แตกต่างกันบ้าง เนื่องการออกแบบ ASIC นั้นซับซ้อนกว่าและใช้ค่าใช้จ่ายมากกว่า อีกทั้งต้องใช้ ซอฟต์แวร์ช่วยออกแบบหลายตัว และมักจะมีราคาค่อนข้างสูง

2.7.1 การออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์(Hardware Description Language)

ในการออกแบบวงจรดิจิตอลนั้น ทำได้ทั้งโดยการวาดวงจร (schematic drawing) หรือใช้ภาษาอธิบายฮาร์ดแวร์(Hardware Description Language) ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะต่างกันโดยที่การทำวิธีนี้ผู้ออกแบบต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นได้ว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์

ทำได้สะดวกกว่าเพราะการทำวิธีนี้ผู้ออกแบบไม่ต้องมาคำนึงถึงเทคโนโลยีที่จะใช้ และที่สำคัญการออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล หรือเปลี่ยนเทคโนโลยีได้สะดวกเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

2.7.2 การสังเคราะห์วงจร (Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์สังเคราะห์วงจร ทำการสังเคราะห์โค้ด VHDL หรือ Verilog เพื่อให้ได้เป็นวงจรขึ้นมา ซึ่งทำได้โดยใช้ซอฟต์แวร์อีกเช่นกันแต่ต้องตรวจสอบด้วยว่า ซอฟต์แวร์นั้นๆสนับสนุนเทคโนโลยี FPGA ที่ต้องการใช้หรือไม่ ตัวอย่าง FPGA ที่มีการใช้งานกันอย่างแพร่หลาย เช่น FPGA ของบริษัท Xilink และบริษัท Altera ในการทำ FPGA นั้นมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Synopsys, Exemplar ในขั้นตอนนี้ซอฟต์แวร์สังเคราะห์วงจรจะแปลงโค้ด VHDL และทำการ optimization เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ ในการสังเคราะห์วงจรเพื่อทำ FPGAนั้นวงจรระดับเกต (gate-level) ไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ optimization นั้นซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ optimization ให้ได้เป็นวงจรที่ประกอบด้วยกลุ่มของลอจิก (logic) ที่เหมาะสมกับอุปกรณ์ FPGA

เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอฟต์แวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบนั้นเป็นไปอย่างไร เช่น มีความหน่วง (delay) เท่าไร ใช้ทรัพยากรใน FPGA อะไรบ้าง เมื่อมาถึงขั้นตอนนี้ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็ให้ทำการสังเคราะห์ใหม่จนกว่าเป็นไปตามที่กำหนด ในขั้นตอนการสังเคราะห์นี้มีเทคนิคต่างๆที่ผู้ออกแบบนำมาใช้เพื่อให้โมเดลเป็นไปตามข้อบังคับที่กำหนด

2.7.3 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBS หรือองค์ประกอบอื่นๆภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing)ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำ โดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท(gate), ฟลิป-ฟลอป(lip-lop) ลงในทรัพยากรต่างๆที่มีอยู่ภายในอุปกรณ์ PGA( CLBs, IOBs, BUFT และ edge decoder)

หลังจากทำขั้นตอนนี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ PGA ไปเท่าไรส่วนข้อมูลทางเวลานั้นผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความหน่วงลอจิก(logic delay)ส่วนซอฟต์แวร์ที่ใช้ในขั้นตอนนี้เช่น

M1 ของ XIinx ซึ่งซอฟต์แวร์ตัวนี้จะรวมเอาซอฟต์แวร์ ย่อยอื่นๆ อีกเพื่อให้การทำ PPR(Partitioning, Placement & Routing) เป็นไปอย่างต่อเนื่อง

2.7.4 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้ก็เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (partioning) มาแล้วว่าควรจะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ PGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (route)ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ PGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

การวางอุปกรณ์ที่ดีควรวางส่วนต่างให้อยู่ใกล้กันโดยเฉพาะส่วนที่มีการเชื่อมต่อสัญญาณด้วยกัน นอกจากนั้นการกำหนดตำแหน่งขา IO( /O pin ตามตำแหน่งขา 1/0 ของ FPGA บนแผ่น PCB ก็จะมีผลโตยตรงเลยคือซอฟต์แวร์ จะวาง 10 ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่งบางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา !O ควรกำหนดตำแหน่งให้เหมาะสมหรือไม่ก็ให้ซอฟต์แวร์จัดการเอง ในกรณีที่ใช้ซอฟต์แวร์ M1 ผู้ออกแบบสามารถแก้ไขตำแหน่งใหม่ได้แต่ควรกระทำด้วยความระมัดระวังเป็นอย่างยิ่ง แต่วิธีที่ดีที่สุดคือการให้ซอฟต์แวร์ทำซ้ำหลายๆครั้งเพื่อหา ครั้งที่ดีที่สุด สำหรับเกณฑ์ที่ใช้ในการตัดสินคือความหน่วงที่ได้หลังจากทำการค้นหาเส้นทางแล้ว หรือที่เรียกว่า routing delay

2.7.5 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆภายในอุปกรณ์ PGA เช่นระหว่าง CLBs หรือระหว่างCLB: กับ I0B: ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด(เนื่องจากจำนวนทรัพยากร (resource) สำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ

ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟต์แวร์เช่น M1 ของบริษัท Xlinx หรือผู้ออกแบบจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (manual layout ) ก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่าโดยให้ ทำการค้นหาเส้นทางหลายๆครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา( timing constraints)จะช่วยให้ผลที่ได้จากการทำเชื่อมต่อสัญญาณดีขึ้นได้2.3.5 การโปรแกรมอุปกรณ์ FPGA (configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR ( Paritioning, Placement & Routing แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด ( download ) ลงในอุปกรณ์ FPGA ได้แล้ว ใน

การดาวนโหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร( configuration data )ซึ่งอยู่ในรูปของบิตสตรีม( bi-stream ) ก่อน แล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับอุปกรณ์ PGA ของแต่ละบริษัทผู้ผลิตคือ ในกรณีที่เป็นอุปกรณ์ "PGA ชนิดที่ต้องโปรแกรมโดยวิธี SRAM นั้น ในการใช้งานผู้ออกแบบต้องเก็บข้อมูลวงจร(configuration data) ไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะได้ใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ต้องดาวนโหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีกเพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่แล้ว แต่ในกรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Antifuse ก็ไม่จำเป็นต้องมีหน่วยความจำไว้สำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูลวงจรลงไปแล้ว ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออกแบบไว้ได้เลย

2.7.6 การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะมี ซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจร ที่ใช้อยู่ เช่น ModelSim ของบริษัท Model Technology ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดลเกิดจากขั้นตอนไหนจะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้เกิดข้อผิดพลาด นั่นคือกรทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังจากเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) แต่ยังไม่สามารถตรวจสอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้วเพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องอยู่หรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตามข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ ถ้ามีก็จะได้แก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ และเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้จะเป็นผลลัพธ์ของโมเดลเลย งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้องตรวจสอบคุณสมบัติอื่นๆเช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบคำความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดไว้หรือไม่ หรือตรวจสอบว่าแบบวงจรรวม

สามารถใช้งานที่ความถี่สูงสุดเท่าไร นั่นเอง ในการจำลองการทำงานของวงจรควรใช้ ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

**2.8 วรรณกรรมที่เกี่ยวข้อง**

2.8.1 WinoNN: Optimizing FPGA-Based Convolutional Neural Network Accelerators Using Sparse Winograd Algorithm

งานวิจัยนี้นำเสนออัลกอริทึม Winograd [24] เป็นการปรับโครงสร้างการคอนโวลูชัน (Convolution) ไปอย่างสิ้นเชิง ซึ่งอัลกอริทึมนี้จะทำการเปลี่ยนโดเมนของอินพุต และฟิลเตอร์ โดยโดเมนอินพุตจะผ่านการแปลงโดยการคูณค่าคงที่ด้วยเมตริกซ์ B และ ฟิลเตอร์จะผ่านการแปลงโดยการคูณค่าคงที่ด้วยเมตริกซ์ G แล้วนำค่าที่ถูกแปลงทั้งสองค่ามาคูณแบบพิกเซลต่อพิกเซล (Element wise) และนำผลลัพธ์ที่คูณมาแปลงกลับไปโดมเมนเริ่มต้นด้วยการคูณค่าคงที่ด้วยเมตริกซ์ A โดยที่อัลกอริทึมนี้จะลดการคำนวณสำหรับการคูณลงอย่างมากเมื่อเทียบกับการคอนโวลูชันแบบปกติ (Convolution)

2.8.2 CSD multipliers for FPGA DSP applications

งานวิจัยนี้นำเสนอวิธีการคูณตัวเลขด้วย CSD [27] โดยจะนำเสนอรูปแบบตัวเลข CSD และวิธีการแปลงตัวเลขระบบ 2’complement เป็นเลขระบบ CSD ซึ่งระบบเลข CSD จะมีบิตที่ไม่ใช่ 0 ห้ามติดกัน ดังนั้นตัวเลขระบบ CSD จะมีข้อได้เปรียบทางการคำนวณด้วยการคูณ โดยจะลดวงจรบวก ทำให้ลดการคำนวณด้วยการคูณเลขระบบ CSD เมื่อเทียบกับการคูณเลขด้วยระบบ 2’complement

2.8.3 A New Algorithm for Carry-Free Addition of Binary Signed-Digit Numbers

งานวิจัยนี้จะนำเสนออัลกอริทึม Online Bit Adder [28] โดยอัลกอริทึมนี้มีความพิเศษ คือจะทำการคำนวณบิต MSB ก่อน LSB โดยจะมีสัญญาณสองตัวควบคุมเพื่อให้การบวกเป็นไปอย่างถูกต้อง เนื่องจากมีการคำนวณจาก MSB ไปยัง LSB ส่งผลให้วงจรไม่จำเป็นต้องรอบิตทด (carry bit) ถ้าสร้างเป็นวงจร FIR filter ดังนั้น อัลกอริทึมจะช่วยลดเวลาการบวกเลขอย่างมาก เพราะไม่ต้องรอบิตทด

# บทที่ 3
# วิธีดำเนินงานวิจัย

ในส่วนของวิธีในการดำเนินงานวิจัยนี้ จะกล่าวถึงขั้นตอนการดำเนินงานของงานวิจัยฉบับนี้ ซึ่งสามารถแบ่งขั้นตอนการดำเนินงานได้เป็นดังนี้ คือ รวบรวมข้อมูลและกำหนดขอบเขตของงาน ศึกษาสถาปัตยกรรมฮาร์ดแวร์ของตัวเร่งความเร็วของการเรียนรู้เชิงลึก สำหรับใช้งานในระบบสมองกลฝังตัว ออกแบบสร้างวงจรดิจิทัลในเอฟพีจีเอ ออกแบบสถาปัตกรรมฮาร์ดแวร์เร่งความเร็ว ทดสอบและวัดประสิทธิภาพฮาร์ดแวร์ที่ออกแบบ และสรุปผลการดำเนินงานวิจัยและจัดทำรายงานวิทยานิพนธ์ ซึ่งมีรายละเอียดของแต่ละขั้นตอนการดำเนินงานดังนี้

## 3.1 รวบรวมข้อมูล และกำหนดขอบเขตงาน

ในขั้นตอนนี้จะเป็นการศึกษารวบรวมข้อมูลที่เกี่ยวข้องกับระบบงาน ผลงานที่เกี่ยวข้อง แล้วนำข้อมูลที่ได้มาทำการวิเคราะห์ และกำหนดถึงขอบเขตงานวิจัย

3.1.1 กำหนดโปรแกรมที่จะใช้ในการสร้างฮาร์ดแวร์การเรียนรู้เชิงลึกสำหรับการวิจัย

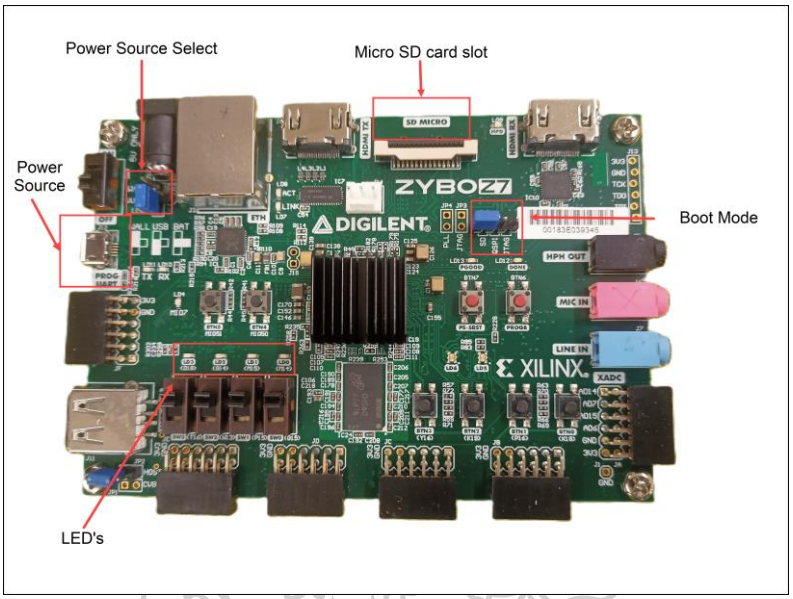
จากการศึกษางานวิจัยที่เกี่ยวข้องพบว่าปัจจุบันมีซอฟต์แวร์ และภาษาที่ใช้ในการเขียนโปรแกรม และพัฒนาฮาร์ดแวร์การเรียนรู้เชิงลึก เช่น โปรแกรม Vivado ที่ใช้ภาษา Verilog ในการออกแบบ และพัฒนาฮาร์ดแวร์การเรียนรู้เชิงลึก ในงานวิจัยนี้จะทำการออกแบบ และสร้างฮาร์ดแวร์การเรียนรู้เชิงลึก โดยใช้โปรแกรม Vivado และใช้ภาษา Verilog ในการออกแบบ และสร้างฮาร์ดแวร์การเรียนรู้เชิงลึก และใช้บอร์ด Zybo Z7 ในการทดลองกับระบบ

3.1.2 กำหนดฐานข้อมูลของรูปภาพที่ใช้ในการทดสอบและประเมินความสามารถของระบบ

ในการทดลองประสิทธิภาพของฮาร์ดแวร์การเรียนรู้เชิงลึก ที่สร้างโดยใช้ FPGA เพื่อให้การทดสอบ และประเมินความสามารถของระบบสามารถเชื่อถือได้และมีมาตรฐาน งานวิจัยฉบับนี้จึงเลือกที่จะทำการทดสอบ และประเมินความสามารถของฮาร์ดแวร์การเรียนรู้เชิงลึกดังกล่าวด้วยฐานข้อมูลรูปภาพ Cifar-10 และเปรียบเทียบกับงานวิจัยที่เกี่ยวข้อง

## 3.2 เครื่องมือที่ใช้ในการวิจัย

3.2.1 ฮาร์ดแวร์ที่ใช้




ภาพที่ 10: บอร์ด Zybo Z7-20 Zynq-7000 ARM/FPGA AP SoC XC7Z020

**สเปคการใช้งาน**

| หมวดหมู่ | รายละเอียด |
|---|---|
| Processor | 667 MHz Dual-core Cortex-A9 |
| Memory Controller | DDR3L Controller, 8 DMA Channels, 4 High Performance AXI3 Slave Ports |
| High-bandwidth Peripherals | 1G Ethernet, USB 2.0, SDIO |
| Low-bandwidth Peripherals | SPI, UART, CAN, I2C |
| Programmable Logic | Equivalent to Artix-7 FPGA |
| Programmable From | JTAG, Quad-SPI Flash, microSD Card |
| DDR3L RAM | 1 GB DDR3L, 32-bit bus @ 533 MHz (1066 MT/s) |
| Quad-SPI Flash | 16 MB, Factory programmed 128-bit random number + 48-bit globally unique EUI-48/64™ ID |
| microSD Slot | รองรับการโปรแกรมและเก็บข้อมูล |
| Power Input | USB หรือ 5V DC External |

| | |
|---|---|
| Gigabit Ethernet | Gigabit Ethernet PHY |
| USB Interfaces | USB-JTAG, USB-UART, USB 2.0 OTG (Host & Device) |
| Camera & Video | Pcam (MIPI CSI-2), HDMI In (with/without CEC), HDMI Out (with CEC) |
| Audio | Audio Codec: Stereo Headphone, Stereo Line-in, Microphone |
| Switches & LEDs | 6 Push-buttons (2 Processor-connected), 4 Slide switches, 5 LEDs (1 Processor-connected), 2 RGB LEDs (1 usable) |

ตารางที่ 1 : สเปคของบอร์ด Zybo Z7-20 Zynq-7000 ARM/FPGA AP SoC XC7Z020

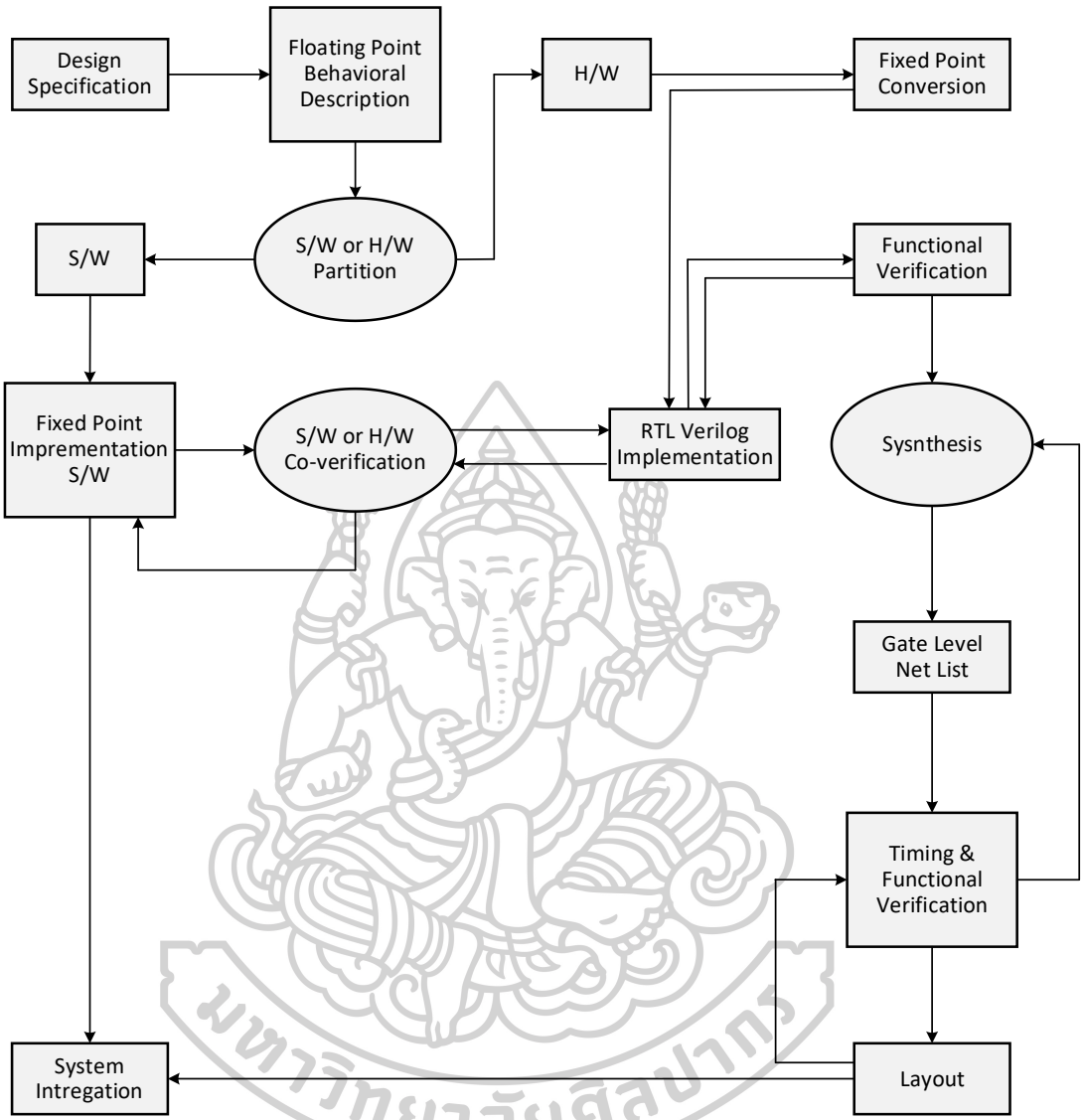
3.2.2 ซอฟต์แวร์ที่ใช้

ซอฟต์แวร์ที่ใช้ในงานนี้คือ Pycharm Community Edition กรอบงาน Pytorch และ Vivado 2019.1 SDK Development

**3.3 ศึกษาสถาปัตยกรรมฮาร์ดแวร์ของตัวเร่งความเร็วของการเรียนรู้เชิงลึก**

ในขั้นตอนนี้จะเป็นการศึกษางานวิจัยที่เกี่ยวข้องกับตัวเร่งความเร็วเอฟพีจีเอสำหรับการเรียนรู้เชิงลึก เพื่อดูเทคนิค และวิธีการออกแบบตัวเร่งความเร็ว โดยอ้างอิงจากงานวิจัยที่เกี่ยวข้อง

**3.4 ออกแบบและสร้างวงจรดิจิทัลในเอฟพีจีเอ**



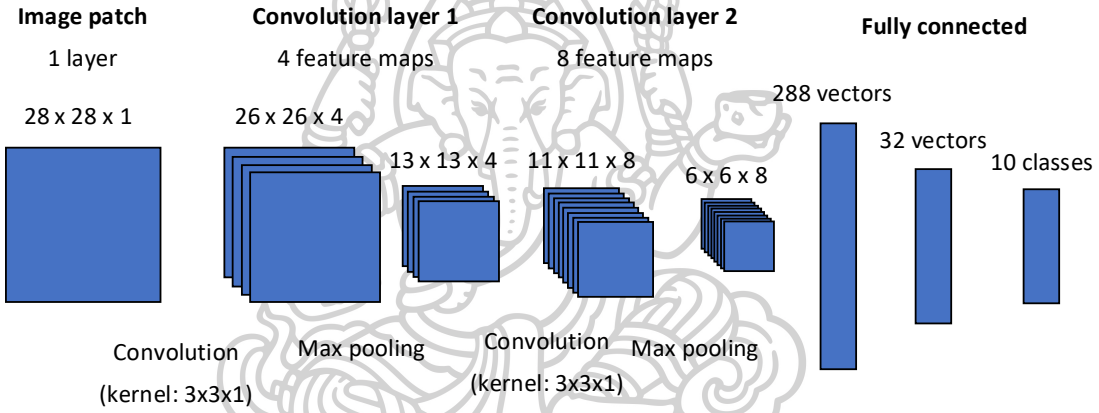

ภาพที่ 11: ขั้นตอนการออกแบบวงจรดิจิทัลในเอฟพีจีเอ [30]

สำหรับการออกแบบระบบดิจิทัลที่เกี่ยวข้องกับฮาร์ดแวร์และซอฟต์แวร์ โดยเริ่มจาก Design Specification (ข้อกำหนดการออกแบบ) ซึ่งพัฒนาเป็น Floating Point Behavioral Description (คำอธิบายเชิงพฤติกรรมของจุดลอยตัว) จากนั้นแบ่งส่วนงานระหว่าง ซอฟต์แวร์ (S/W) หรือ ฮาร์ดแวร์ (H/W) เพื่อกำหนดว่าแต่ละฟังก์ชันจะถูกดำเนินการในรูปแบบใด ฝั่งฮาร์ดแวร์จะทำ Fixed Point Conversion (การแปลงเป็นจุดคงที่) และทำ Functional Verification (การตรวจสอบการทำงาน) ในขณะที่ฝั่งซอฟต์แวร์ดำเนินการ Fixed Point Implementation (การใช้งานจุดคงที่) พร้อมกับกระบวนการ S/W or H/W Co-verification (การตรวจสอบร่วมกันระหว่างฮาร์ดแวร์และซอฟต์แวร์) จากนั้นพัฒนา RTL Verilog Implementation (การพัฒนาโค้ดระดับ

Register Transfer Level) และนำไปสู่กระบวนการ Synthesis (การสังเคราะห์) เพื่อสร้าง Gate Level Net List (รายการเน็ตลิสต์ระดับเกต) จากนั้นดำเนินการ Timing & Functional Verification (ตรวจสอบเวลาและการทำงาน) และ Layout (การออกแบบทางกายภาพ) สุดท้ายกระบวนการทั้งหมดถูกบูรณาการเข้าสู่ System Integration (การรวมระบบ) เพื่อให้แน่ใจว่าระบบทำงานได้อย่างถูกต้องตามข้อกำหนดการออกแบบ

### 3.5 ออกแบบสถาปัตกรรมคอนโวลูชัน

ในขั้นตอนนี้จะออกแบบสถาปัตกรรมคอนโวลูชันต้นแบบ เพื่อเป็นโมเดลตั้งต้นสำหรับทดสอบในงานวิจัยนี้ ดังรูปภาพที่ 12




ภาพที่ 12: สถาปัตกรรมคอนโวลูชันต้นแบบ

### 3.6 ออกแบบสถาปัตกรรมฮาร์ดแวร์เร่งความเร็ว

ในขั้นตอนนี้จะออกแบบสถาปัตยกรรมที่ช่วยเร่งความเร็วโดยใช้แนวทาง Winograd Convolution และใช้ระบบเลข Canonical Signed Digit (CSD)

3.6.1 Winograd Convolution

สำหรับแนวคิดการออกแบบ Winograd convolution เป็นการออกแบบที่ง่าย เพราะค่าคงที่ในเมตริกซ์ G, B และ A เป็นค่า 1, -1 และ 0.5 ดังแสดงในหัวข้อที่ 2 ทฤษฎีพื้นฐาน ซึ่งเทคนิคการออกแบบสำหรับเลข 1 คือ จะคงค่าตัวเลขไว้, สำหรับค่า -1 จะอินเวิร์สค่าตัวเลข และ และสำหรับ 0.5 คือ จะทำการ Shift Right ตัวเลข ซึ่งเป็นการหารด้วยสอง จะสังเกตเห็นว่าการคูณเมตริกซ์แทบจะไม่ใช้การคูณเลย ทำให้การออกแบบ Winograd convolution ในฮาร์ดแวร์มีความได้เปรียบทางการคำนวณมากกว่าการทำงานในซอฟต์แวร์

3.6.2 Canonical Signed Digit (CSD)

สำหรับแนวคิดการออกแบบ CSD เป็นการทำให้เลขบิต 1 มีค่าน้อยที่สุด แต่อาจจะเพิ่มการใช้หน่วยความจำที่เยอะขึ้น ซึ่งส่งผลโดยตรงสำหรับการคูณ เพราะเลขฐานสองมีค่าแค่ 1 กับ 0 ถ้าเลขฐานสองยิ่งมีบิตที่เป็น 1 เยอะ จะส่งผลให้มีการใช้งานวงจรบวกที่เยอะขึ้น โดยวิธีการเข้ารหัสตัวเลขแบบ CSD จะแทนค่าด้วย (-1, 0, 1) => (10, 00, 01)

## 3.7 ทดสอบและวัดประสิทธิภาพฮาร์ดแวร์ที่ออกแบบ

ในงานวิจัยจะออกแบบฮาร์ดแวร์เร่งความเร็วของการเรียนรู้เชิงลึก โดยใช้ซอฟต์แวร์ Vivado เขียนโปรแกรมโดยใช้ภาษา Verilog และทำการสังเคราะห์โปรแกรมลงบนบอร์ด Zybo Z7 ในส่วนของฮาร์ดแวร์เร่งความเร็ว Winograd convolution จะวิเคราะห์ทรัพยากรที่ใช้เปรียบเทียบกับ Winograd convolution + CSD Representation

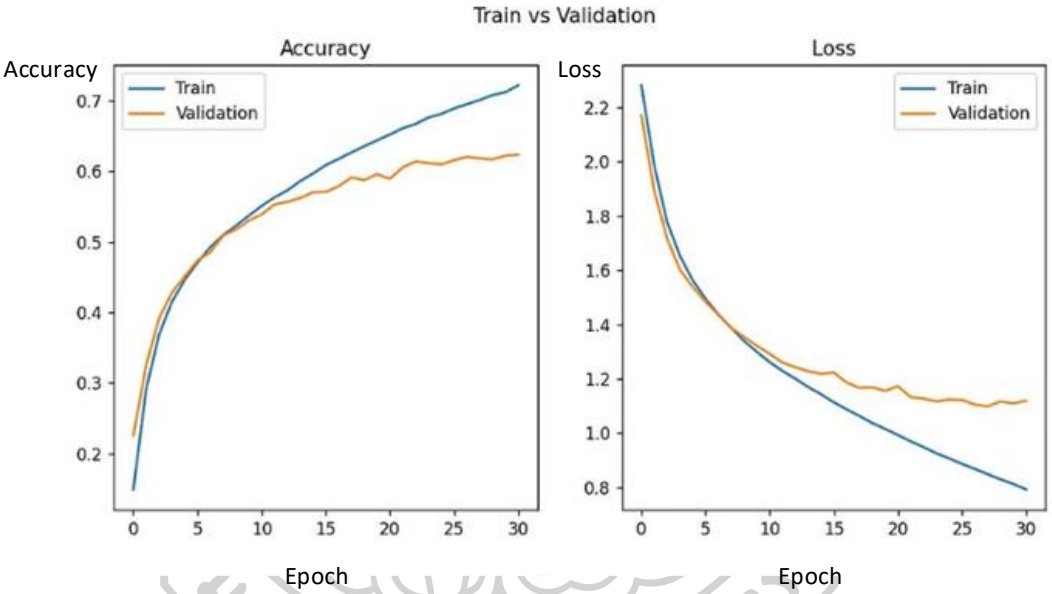
## 3.8 สรุปผลการดำเนินงานวิจัย

สรุปผลของการดำเนินงานวิจัยและจัดทำรายงานวิทยานิพนธ์ถือเป็นขั้นตอนสุดท้ายของการวิจัยนี้ โดยจะนำผลลัพธ์ที่ได้จากการทดลองทั้งหมดมาทำการวิเคราะห์และสรุป

# บทที่4
# ผลการวิจัย

## 4.1 ผลลัพธ์การสร้างแบบจำลองโครงข่ายประสามเทียมคอนโวลูชันกรอบงาน Pytorch

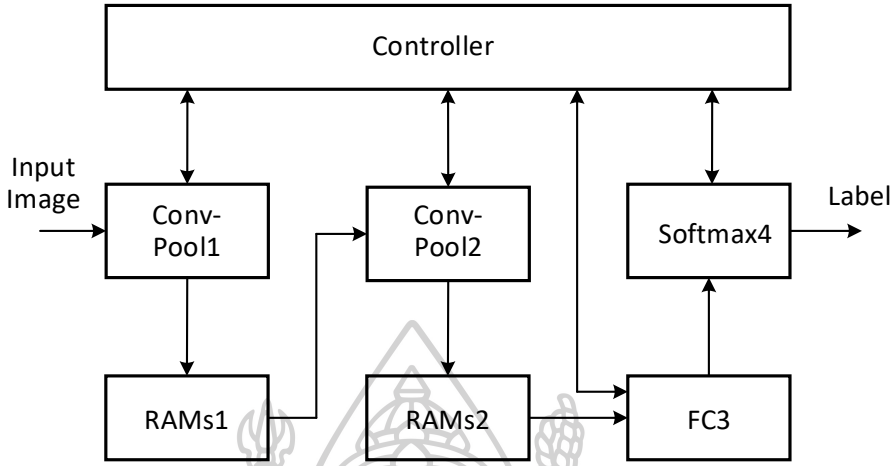
ในหัวข้อนี้จะแสดงผลลัพธ์การสอนของโมเดลคอนโวลูชันต้นแบบ ใช้ข้อมูลทดสอบ MNIST โดยกรอบงาน Pytorch โดยจะแสดงผลลัพธ์เป็นกราฟตามภาพที่ 13 กราฟด้านซ้ายจะแสดงผลลัพธ์ Accuracy และกราฟด้านขวาจะแสดงผลลัพธ์ Loss โดยกราฟเส้นสีฟ้าคือชุดข้อมูลที่สอน และเส้นกราฟสีส้มคือชุดข้อมูลที่ทดสอบ




ภาพที่ 13: ด้านซ้ายจะแสดงผลลัพธ์ Accuracy และด้านขวาจะแสดงผลลัพธ์ Loss จากโมเดลคอนโวลูชันต้นแบบ

**4.2 ผลลัพธ์การออกแบบสถาปัตกรรมคอนโวลูชันต้นแบบ**

4.2.1 ภาพรวมโมดูลคอนโวลูชัน

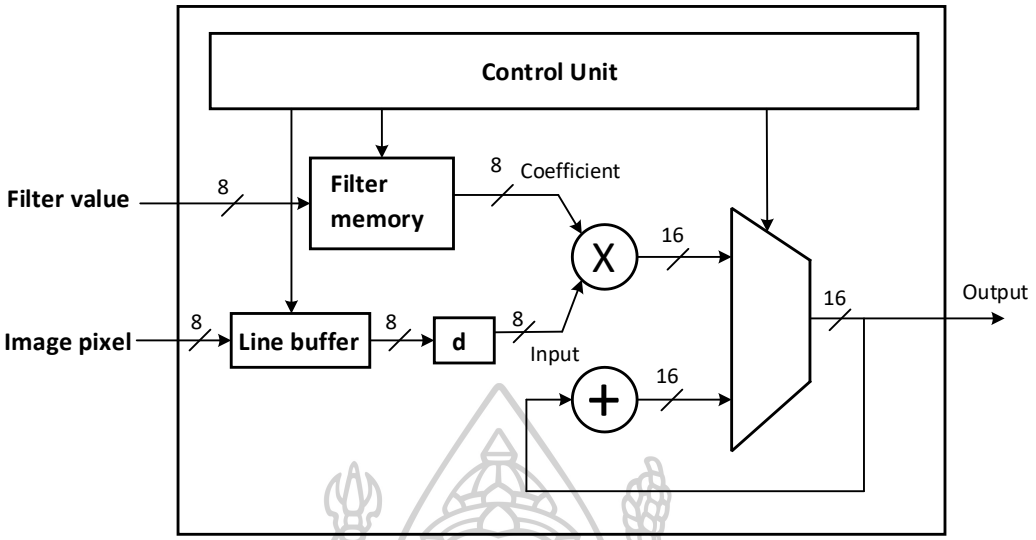



ภาพที่ 14: ภาพรวมโมดูลคอนโวลูชัน

จากภาพที่ 14 เป็นบล็อกไดอะแกรมของโครงสร้างระบบประมวลผลภาพด้วย Convolutional Neural Network (CNN) แบบฮาร์ดแวร์ โดยมีส่วนประกอบดังนี้:

1) Input Image: รับภาพเข้ามาเป็นข้อมูลเริ่มต้น
2) Conv-Pool1: เป็นส่วนที่ทำ Convolution และ Pooling ชั้นที่ 1
3) RAMs1: หน่วยความจำที่ใช้เก็บผลลัพธ์จาก Conv-Pool1 เพื่อส่งต่อไปยังขั้นตอนถัดไป
4) Conv-Pool2: เป็นส่วนที่ทำ Convolution และ Pooling ชั้นที่ 2
5) RAMs2: หน่วยความจำที่ใช้เก็บผลลัพธ์จาก Conv-Pool2 ก่อนส่งไปยัง Fully Connected Layer
6) FC3 (Fully Connected Layer 3): ทำหน้าที่เชื่อมต่อข้อมูลที่ได้จาก Conv-Pool2 เพื่อนำไปคำนวณต่อ
7) Softmax4: เป็นขั้นตอนที่ใช้ Softmax Function แปลงค่าความน่าจะเป็นเพื่อเลือก class label
8) Label: ส่งผลลัพธ์ออกมาเป็น class label สุดท้าย
9) Controller: เป็นหน่วยควบคุมหลักของระบบ ทำหน้าที่ควบคุมการทำงานของแต่ละโมดูลให้ประสานกัน

4.2.2 โมดูลคอนโวลูชัน



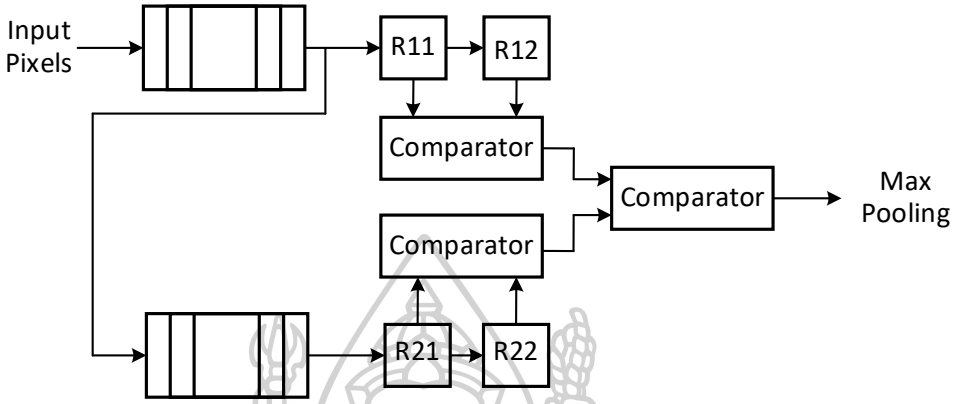

ภาพที่ 15: โมดูลคอนโวลูชัน

จากภาพที่ 15 เป็นบล็อกไดอะแกรมของหน่วยประมวลผลการคูณค่าฟิลเตอร์กับพิกเซลของภาพ (Convolution Unit) ซึ่งใช้ในระบบ CNN แบบฮาร์ดแวร์ โดยมีส่วนประกอบและการทำงานดังนี้:

1) Filter value: ค่า filter (weight) ขนาด 8 บิต ที่รับเข้ามา
2) Image pixel: ค่าพิกเซลของภาพ ขนาด 8 บิต ที่รับเข้ามา
3) Filter Memory: หน่วยความจำเก็บค่า filter coefficients
4) Line Buffer: หน่วยบัฟเฟอร์เก็บค่าพิกเซลของภาพที่รับเข้ามา เพื่อหน่วงไว้ให้พร้อมใช้งาน
5) d: เอาต์พุตจาก Line Buffer (ขนาด 8 บิต) ที่จะนำไปคูณกับ filter
6) × (Multiplier): ตัวคูณระหว่างค่า Coefficient (8 บิต) กับ Input (8 บิต) ออกมาเป็นค่า 16 บิต
7) + (Adder): ตัวบวกค่า 16 บิต เพื่อนำผลคูณแต่ละตำแหน่งมารวมกัน (accumulate)
8) Multiplexer (MUX): ตัวเลือกข้อมูลที่จะส่งออก ระหว่างผลรวม หรือข้อมูลอื่นตามที่ Control Unit ควบคุม
9) Output: ผลลัพธ์ขนาด 16 บิต จากการคำนวณ convolution

10) Control Unit: หน่วยควบคุมการทำงานของทุกส่วนในโมดูลนี้ ตั้งแต่การอ่านข้อมูล การเลือก filter และการควบคุมการคำนวณ

4.2.3 โมดูล Max Pooling

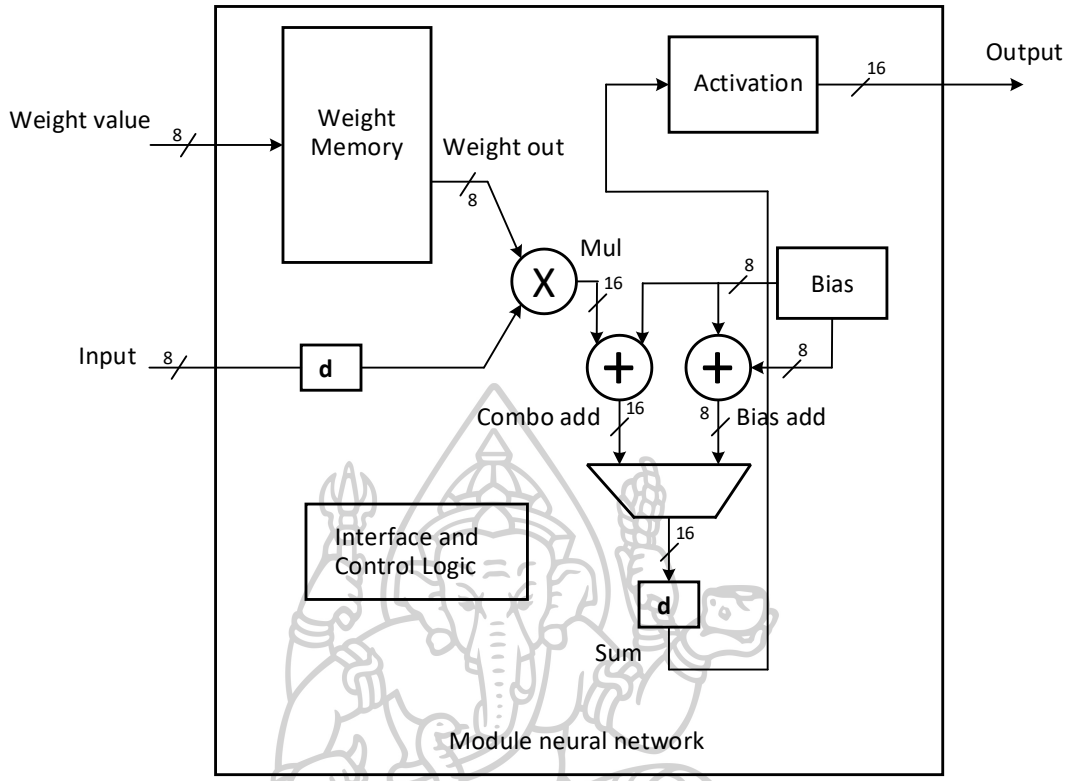



ภาพที่ 16: โมดูล Max Pooling

จากภาพที่ 16 เป็นบล็อกไดอะแกรมของกระบวนการ Max Pooling ซึ่งใช้ใน Convolutional Neural Network (CNN) แบบฮาร์ดแวร์ Max Pooling เป็นกระบวนการเลือกค่าพิกเซลที่มีค่ามากที่สุดจากกลุ่มพิกเซล (เช่น 2x2 หรือ 3x3) เพื่อลดขนาดข้อมูลและรักษาคุณสมบัติเด่นของภาพไว้

1) Input Pixels: ข้อมูลพิกเซลที่รับเข้ามา
2) R11, R12, R21, R22: เป็นค่าพิกเซลแต่ละตัวในกลุ่ม 2x2 ที่จะถูกนำมาเปรียบเทียบกัน (เช่นถ้าเป็น max pooling 2x2 ก็จะมีพิกเซล 4 ตัว)
3) Comparator: ตัวเปรียบเทียบค่าพิกเซล 2 ค่า เพื่อเลือกค่าที่มากกว่า
4) Max Pooling Output: ผลลัพธ์ที่ได้จากการเลือกค่าพิกเซลที่มีค่ามากที่สุดในกลุ่ม

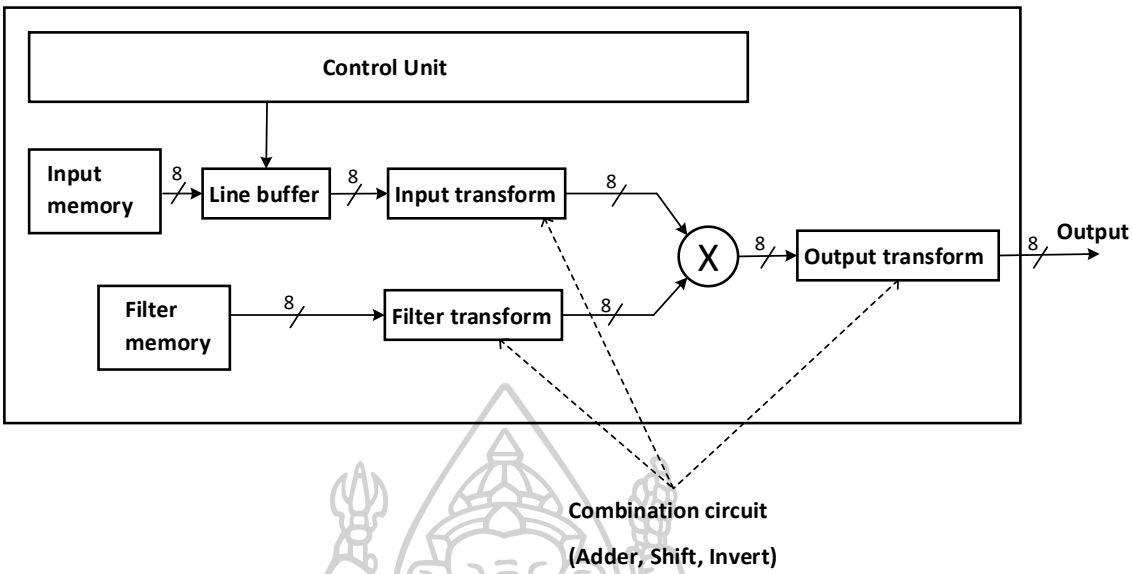
4.2.4 โมดูล Fully Connected




ภาพที่ 17: โมดูล Fully Connected

จากภาพที่ 17 แสดงโครงสร้างของ โมดูลโครงข่ายประสาทเทียม (Neural Network Module) ซึ่งทำงานประมวลผลข้อมูลด้วยการคำนวณแบบพื้นฐานในโครงข่ายประสาทเทียมแบบ feedforward layer หนึ่ง โดยเริ่มจากการรับค่า Input ขนาด 8 บิต และค่าถ่วงน้ำหนัก (Weight value) ขนาด 8 บิต เข้ามาเก็บไว้ใน Weight Memory จากนั้นนำค่าถ่วงน้ำหนักที่อ่านออกมา (Weight out) มาคูณกับค่า Input ผ่านตัวคูณ (Mul) ให้ได้ผลลัพธ์ขนาด 16 บิต หลังจากนั้นผลลัพธ์จากการคูณจะนำไปบวกสะสมกับค่าผลรวมก่อนหน้า (Sum) ผ่านตัวบวก (Comb add) โดยมีค่าปรับแต่งค่าคงที่ (Bias) ขนาด 8 บิต ที่นำมาบวกกับผลรวมผ่านตัวบวกอีกตัว (Bias add) ให้ได้ค่าผลรวมสุดท้ายขนาด 16 บิต หลังจากนั้นค่าดังกล่าวจะถูกส่งผ่านหน่วย Activation Function เพื่อแปลงให้อยู่ในช่วงหรือรูปแบบที่เหมาะสมสำหรับการนำไปใช้งานต่อ ผลลัพธ์สุดท้ายจะถูกส่งออกทาง Output ขนาด 16 บิต นอกจากนี้ภายในโมดูลยังมี Interface and Control Logic สำหรับควบคุมการทำงานลำดับต่าง ๆ ของวงจร ทั้งในส่วนของการอ่านค่า การคูณ การบวก การปรับ Bias และการส่งข้อมูลออก

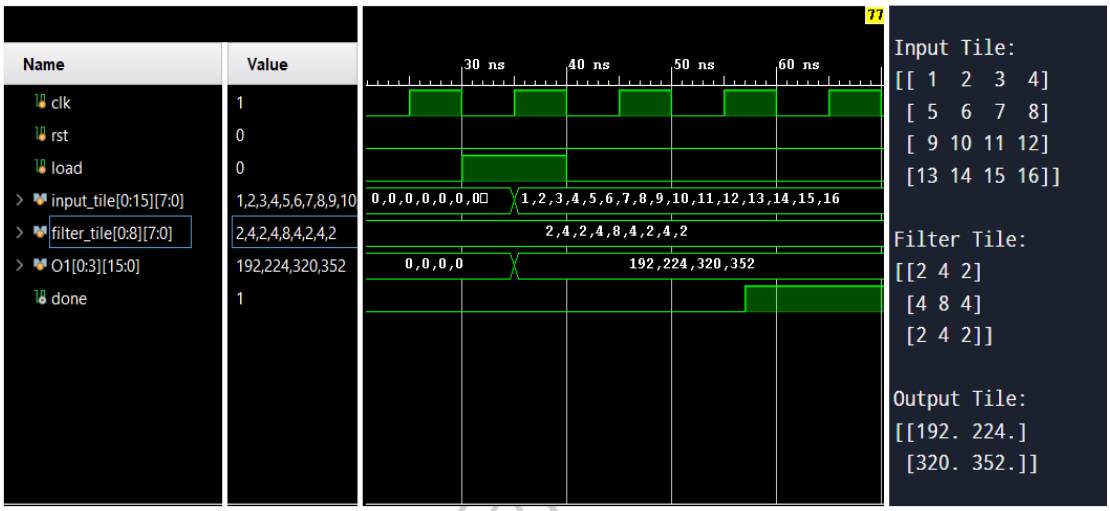
**4.3 ผลลัพธ์การออกแบบสถาปัตกรรมฮาร์ดแวร์โดยใช้ Winograd convolution**




ภาพที่ 18: ผลลัพธ์การออกแบบฮาร์ดแวร์ Winograd convolution

จากภาพที่ 18 ฮาร์ดแวร์ Winograd convolution ประกอบด้วยโมดูลย่อยหลายส่วน คือ Input memory, Filter memory, Buffer, Invert input, Invert filter และ Convert output โดยมีหน้าที่การทำงาน ดังนี้

1) Input memory มีหน้าที่เก็บข้อมูลชั่วคราวจากรูปภาพหรืออินพุต
2) Filter memory มีหน้าที่เก็บข้อมูลชั่วคราวของ Filter โดยค่าจะเป็นเลข CSD
3) Buffer เนื่องจาก Winograd convolution จะคำนวณค่าเมทิรกซ์เป็นไทล์ ดังนั้นจะมีหน้าที่ดึงค่าจากอินพุตเป็นแต่ละไทล์ โดย 1 ไทล์ คือ 4x4 pixel
4) Input transform มีหน้าที่แปลงอินพุตไปอีกโดเมนหนึ่ง โดยในโมดูลนี้จะประกอบไปด้วยวงจรบวกเป็นหลัก
5) Filter transform มีหน้าที่แปลงอินพุตไปอีกโดเมนหนึ่ง โดยในโมดูลนี้จะประกอบไปด้วยวงจรบวก CSD และ shift register
6) Output transform มีหน้าที่แปลงผลลัพธ์ที่ได้จากการคูณผลลัพธ์ระหว่าง Input transform และ Filter transform ไปโดเมนเริ่มต้นเพื่อให้ได้ผลลัพธ์ที่ถูกต้อง โดยในโมดูลนี้จะประกอบไปด้วยวงจรคูณ CSD

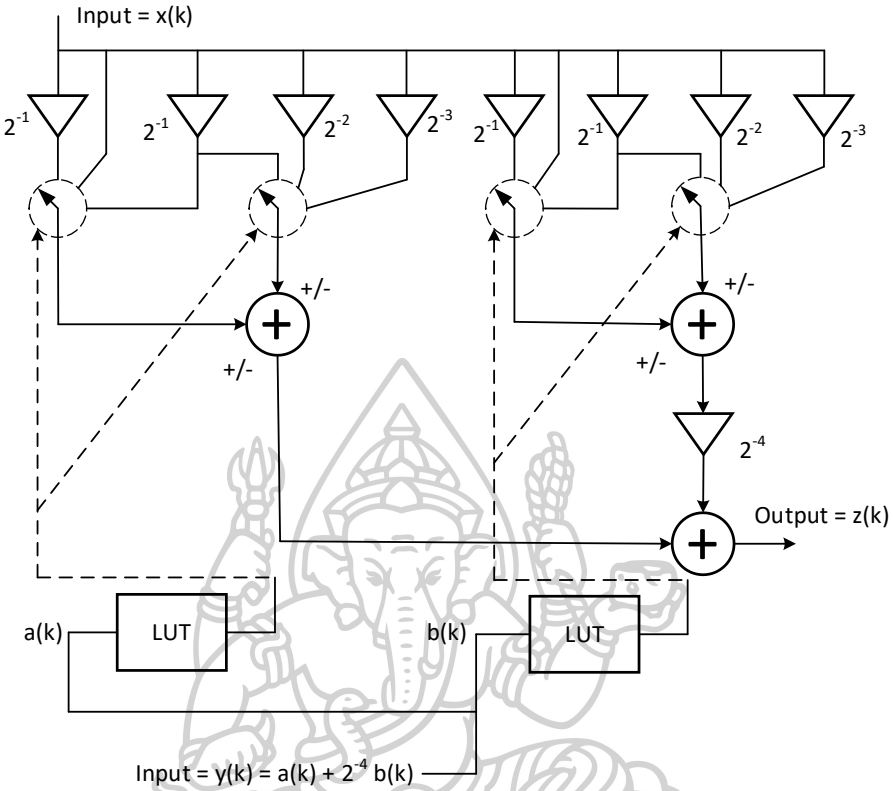



ภาพที่ 19: (ซ้าย)ผลการจำลองการทำงานของวงจรฮาร์ดแวร์ และ(ขวา)ผลการรันโค๊ดภาษา Python Winograd convolution

### 4.4 ผลลัพธ์การออกแบบสถาปัตกรรมฮาร์ดแวร์โดยใช้ระบบเลข Canonical Signed Digit (CSD)

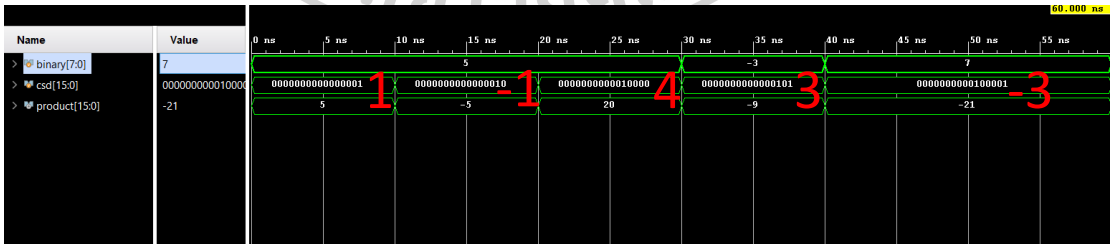
นัวข้อนี้จะทำการออกแบบฮาร์ดแวร์สำหรับ การคูณ และการบวกของระบบเลข Canonical Signed Digit (CSD)

4.4.1 CSD Multiplication




ภาพที่ 20: วงจรการคูณ CSD ขนาด 8 บิต

จากภาพที่ 20 วงจรคูณ CSD ขนาด 8 บิตประกอบไปด้วย LUT ที่ควบคุมการไหลของข้อมูลโดยผลลัพธ์ Output จะมีการบวกข้อมูลไม่เกิน 4 ตัว

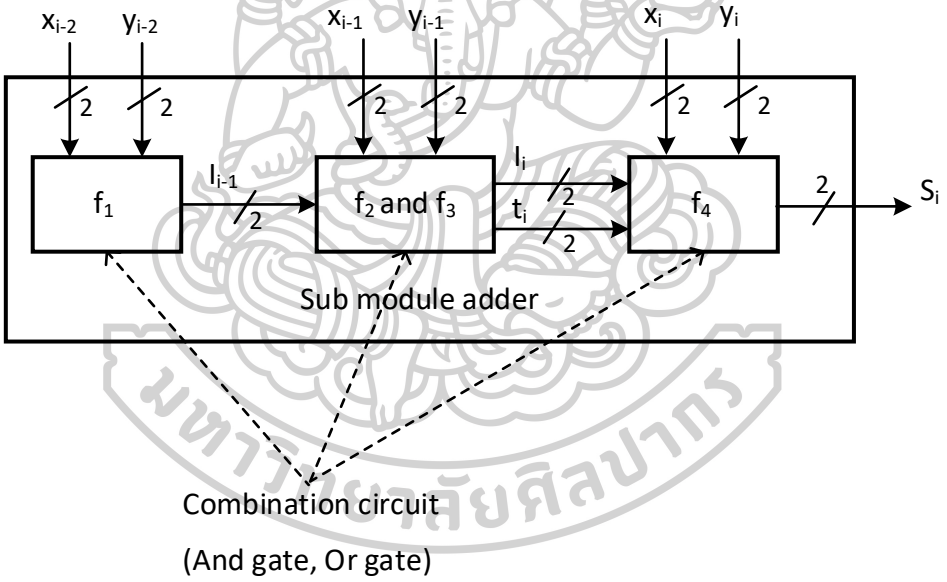



ภาพที่ 21: ผลการจำลองการทำงานของฮาร์ดแวร์ CSD Multiplier

4.4.2 CSD Addition

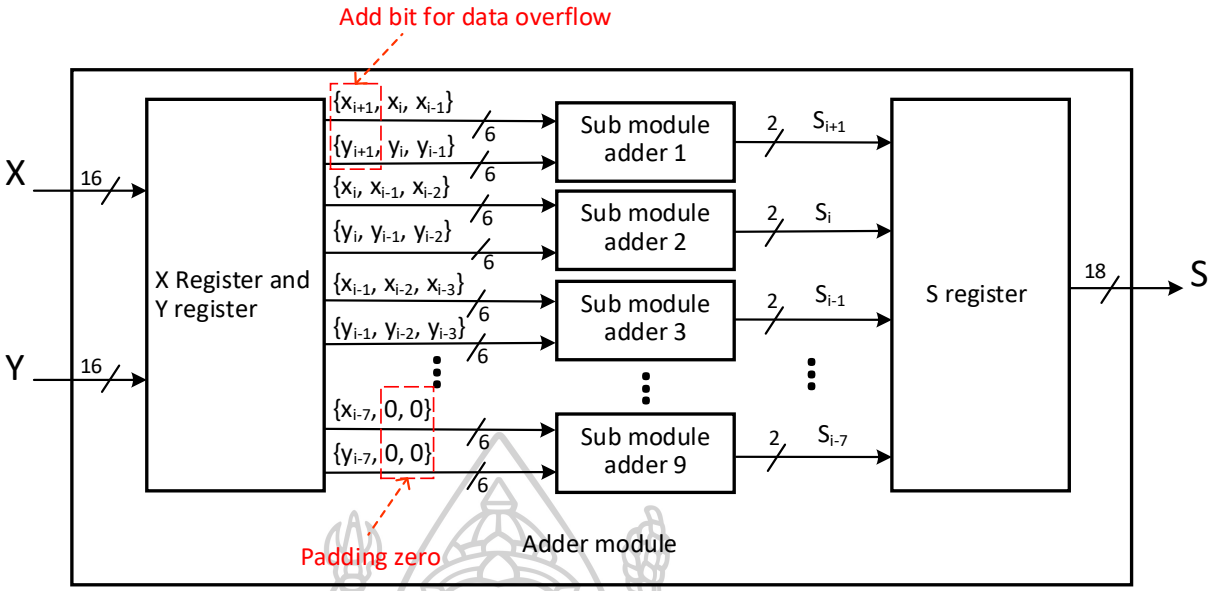
```
input [1:0] tin, x, y,
input lin,
output [1:0] tout, s
output lout
// Define the critical input cases
wire w1 = ~x[1] & ~x[0] & y[0];
wire w2 = ~x[1] & ~x[0] & y[1];
wire w3 = ~y[1] & ~y[0] & x[0];
wire w4 = ~y[1] & ~y[0] & x[1];
wire w = w1 | w2 | w3 | w4;
wire u1 = ~lin & w;  // tin != -1 & critical input
wire u0 = lin & w;   // tin != +1 & critical input
// Determine lout := x == -1 | y == -1
lout = x[1] | y[1];
tout[1] = (x[1] & y[1]) | (lin & (w2 | w4));
tout[0] = (x[0] & y[0]) | (~lin & (w1 | w3));
// Determine sum digit
s[0] = (tin[0] & ~u1) | (u0 & ~tin[1]);
s[1] = (tin[1] & ~u0) | (u1 & ~tin[0]);
```

ภาพที่ 22: โค๊ดภาษา Verilog ของวงจรบวก CSD




ภาพที่ 23: วงจรบวกย่อยของ CSD

ภาพที่ 23 วงจรบวกย่อย CSD จะมีโมดูลหลักๆ อยู่ 4 โมดูล คือ f1, f2, f3 และ f4 โดยที่แต่ละโมดูลจะมีสัญญาณควบคุม I และ t




ภาพที่ 24: ผลลัพธ์การออกแบบฮาร์ดแวร์ CSD Addition

ภาพที่ 24 มี Sub module adder ถึง 9 ตัว เพื่อทำการบวกตัวเลข CSD ขนาด 8 บิต โดยแต่ละโมดูลย่อยจะให้ผลลัพธ์ออกมาทีละบิต จะสังเกตเห็นว่า จำเป็นต้องเพิ่มบิต $x_{i+1}$ กับ $y_{i+1}$ เพื่อให้มีการคำนวณเผื่อการเกิด data overflow โดยจะให้ทั้งสองค่านี้มีค่าเป็น 0 และ จะมีการ padding zero โดยการเติมเลขศูนย์เพื่อให้เกิดการคำนวณที่บิต LSB


ภาพที่ 25: ผลการจำลองการทำงานของฮาร์ดแวร์ CSD Adder

## 4.5 วิเคราะห์และประเมินประสิทธิภาพของวงจรที่ได้ออกแบบ

4.5.1 ผลลัพธ์การใช้ทรัพยากรทั้งหมดของโมเดลต้นแบบ

| Resources | Conv+Pool1 | Conv+Pool2 | FC3 | Total |
|---|---|---|---|---|
| LUTs | 3500 | 5500 | 3000 | 12000 |
| DSPs | 25 | 35 | 21 | 81 |
| FF | 1500 | 2200 | 800 | 4500 |

ตารางที่ 2 : ผลลัพธ์การใช้ทรัพยากรทั้งหมดของโมเดลต้นแบบ

4.5.2 ผลลัพธ์การเปรียบเทียบการใช้ทรัพยากร และความเร็วกับวิธีการอื่นๆ

| | [10] | [31] | [10] | งานวิจัยนี้ |
|---|---|---|---|---|
| โมเดล CNN | VGG16 | VGG16 | VGG16 | VGG16 |
| Device | ZCU102 | XC7Z045 | ZC706 | XC7Z020 |
| Freq.(MHz) | 200 | 200 | 166 | 125 |
| Precision | 16 Bit | 8 Bit | 16 Bit | 8 Bit |
| DSP Used | 2520 | 182 | 900 | 220 |
| Logic Used | 600K | 17.4K | 350K | 43K |
| BRAM | 1824 | 112 | 1090 | 128 |
| Perf.(GOP/s) | 2479 | 99.11 | 130.4 | 44 |
| Power(W) | 23.6 | 0.62 | 9.4 | 2.8 |
| GOPs/W | 105.4 | 159.85 | 13.8 | 15.7 |

ตารางที่ 3 : เปรียบเทียบการใช้ทรัพยากร และความเร็วระหว่าง FPGA ด้วยกัน

จากตารางที่ 2 งานวิจัยนี้เลือกใช้ FPGA XC7Z020 ซึ่งมีข้อดีคือใช้งานบนอุปกรณ์ขนาดเล็ก ราคาประหยัด และใช้พลังงานต่ำที่สุดเพียง 2.8W ขณะที่ยังให้สมรรถนะระดับ 44 GOP/s และประสิทธิภาพต่อวัตต์ที่ 15.7 GOPs/W สูงกว่าบางอุปกรณ์ระดับกลางอย่าง ZC706 (13.8 GOPs/W) แม้จะด้อยกว่าชิประดับสูงอย่าง ZCU102 และ XC7Z045 ที่มี GOPs/W สูงกว่า 100 แต่ก็แลกมากับการใช้ทรัพยากร (DSP, Logic, BRAM) และพลังงานที่สูงตามไปด้วย ดังนั้นจุดเด่นของงานวิจัยนี้คือการออกแบบ CNN บนแพลตฟอร์มทรัพยากรจำกัดที่คุ้มค่าต่อพลังงานและต้นทุน เหมาะสำหรับงานฝังตัวหรือ edge device ส่วนข้อด้อยคือสมรรถนะรวม (GOP/s) ยังห่างจากอุปกรณ์ระดับสูงมาก และอาจไม่เหมาะสำหรับงานที่ต้องการ throughput สูงในระดับหลายร้อยหรือพัน GOP/s

4.5.3 ผลลัพธ์การเปรียบเทียบการใช้ทรัพยากร และความเร็วระหว่าง GPU และ FPGA

| Implementation | Nvidia Tesla T4 [32] | Nvidia Titan X [32] | Nvidia RTX3090 [32] | FPGA (XC7Z020) |
|---|---|---|---|---|
| Model | VGG16 | VGG16 | VGG16 | VGG16 |
| GOP/s | 14958.4 | 5600 | 28780 | 44 |
| Power(W) | 68.8 | 134 | 277 | 2.8 |
| GOPs/W | 217.4 | 41.8 | 103.9 | 15.7 |

ตารางที่ 4 : เปรียบการใช้ทรัพยากร และความเร็วระหว่าง GPU และ FPGA

จากตารางที่ 3 จากตารางเปรียบเทียบ พบว่า FPGA (XC7Z020) ให้ข้อได้เปรียบด้านการใช้พลังงานต่ำที่สุดเพียง 2.8W แต่มีสมรรถนะรวมต่ำสุดเพียง 44 GOP/s และประสิทธิภาพต่อวัตต์ (GOPs/W) อยู่ที่ 15.7 ซึ่งน้อยกว่าทั้ง Nvidia Tesla T4, Titan X และ RTX3090 ที่ทำได้สูงถึง 217.4, 41.8 และ 103.9 GOPs/W ตามลำดับ โดยเฉพาะ Tesla T4 ที่โดดเด่นทั้งด้านสมรรถนะรวมและความคุ้มค่าต่อพลังงาน ส่วน RTX3090 แม้จะให้สมรรถนะสูงสุดถึง 28780 GOP/s แต่ก็แลกกับการใช้พลังงานสูงถึง 277W ดังนั้น FPGA จึงเหมาะกับงานที่ต้องการประหยัดพลังงานและมีข้อจำกัดด้านทรัพยากร เช่น ระบบ edge AI หรือ embedded device ขณะที่ GPU เหมาะกับงานที่ต้องการ throughput สูงและพื้นที่พลังงานไม่ใช่ข้อจำกัด

# บทที่ 5
# สรุป และข้อเสนอแนะ

## 5.1 บทสรุปของงานวิจัย

วิทยานิพนธ์นี้ได้นำเสนอแนวคิดการเร่งความเร็วฮาร์ดแวร์ โดยนำค่าน้ำหนักที่ได้จากการสอนของโมเดล CNN (Convolution neural network) โดยใช้ภาษา Python กรอบงาน Pytorch แล้วนำค่าน้ำหนักของโมเดลมา Quantization เพื่อให้ได้ตัวเลขที่เป็นสัดส่วนจำเต็ม และสุดท้ายนำตัวเลขที่เป็นสัดส่วนจำนวนเต็ม มาแปลงเป็นตัวเลขระบบ Canonical Signed Digit (CSD) แนวคิดที่ใช้ในการเร่งความเร็วฮาร์ดแวร์ในวิทยานิพนธ์มีอยู่สองวิธี คือ Winograd convolution และ CSD Representation

จากการวิเคราะห์จะพบว่า FPGA ในงานวิจัยนี้เป็นอุปกรณ์ระดับกลางทำให้ไม่สามารถสังเคราะห์โมเดลที่มีขนาดใหญ่ได้อย่างมีประสิทธิภาพอย่างโมเดล VGG16 เนื่องจากทรัพยากรมีจำกัด ส่งผลให้ทางด้านความเร็วในการประมวลค่อนข้างต่ำ

## 5.2 ข้อเสนอแนะ

สำหรับฮาร์ดแวร์เร่งความเร็ว Winograd convolution ในขั้นตอนการ Transform Filter จะมีขั้นตอนที่ใช้การ Shift Right แทนการหารด้วย 2 ส่งผลให้เกิดค่าข้อมูลที่ผิดพลาด สำหรับฮาร์ดแวร์เร่งความเร็ว CSD Representation ทั้งวงจรบวก และวงจรคูณ การคำนวณบางกรณีอาจเกิดการ Overflow ส่งผลให้ค่าข้อมูลที่มีค่าผิดพลาด สามารถแก้ไขโดยการเพิ่มหน่วยความจำให้กับอินพุตและเอาท์พุต

# รายการอ้างอิง


[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems,* vol. 25, 2012.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1-9.

[5] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, 2014: Springer, pp. 818-833.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580-587.

[7] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News,* vol. 42, no. 1, pp. 269-284, 2014.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161-170.

[9] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," *IEEE transactions on neural networks and learning systems,* vol. 28, no. 7, pp. 1572-1583, 2016.

[10] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *2017 IEEE 25th annual international symposium on field-*

*programmable custom computing machines (FCCM)*, 2017: IEEE, pp. 101-108.

[11] J. Yu *et al.*, "Instruction driven cross-layer CNN accelerator with winograd transformation on FPGA," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017: IEEE, pp. 227-230.

[12] M. Zhu, Q. Kuang, C. Yang, and J. Lin, "Optimization of convolutional neural network hardware structure based on FPGA," in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018: IEEE, pp. 1797-1802.

[13] A. Hadnagy, B. Fehér, and T. Kovácsházy, "Efficient implementation of convolutional neural networks on FPGA," in *2018 19th International Carpathian Control Conference (ICCC)*, 2018: IEEE, pp. 359-364.

[14] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE transactions on computer-aided design of integrated circuits and systems,* vol. 37, no. 1, pp. 35-47, 2017.

[15] L.-W. Kim, "DeepX: Deep learning accelerator for restricted boltzmann machine artificial neural networks," *IEEE transactions on neural networks and learning systems,* vol. 29, no. 5, pp. 1441-1453, 2017.

[16] G. Natale, M. Bacis, and M. D. Santambrogio, "On how to design dataflow FPGA-based accelerators for convolutional neural networks," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017: IEEE, pp. 639-644.

[17] S. Li, Y. Luo, K. Sun, N. Yadav, and K. K. Choi, "A novel FPGA accelerator design for real-time and ultra-low power deep convolutional neural networks compared with titan X GPU," *IEEE Access,* vol. 8, pp. 105455-105471, 2020.

[18] C. Bao, T. Xie, W. Feng, L. Chang, and C. Yu, "A power-efficient optimizing framework fpga accelerator based on winograd for yolo," *Ieee Access,* vol. 8, pp. 94307-94317, 2020.

[19] C. Liu, C. Wang, and J. Luo, "Large-scale deep learning framework on FPGA for fingerprint-based indoor localization," *IEEE Access,* vol. 8, pp. 65609-65617, 2020.

[20] J. Zhu, L. Wang, H. Liu, S. Tian, Q. Deng, and J. Li, "An efficient task assignment framework to accelerate DPU-based convolutional neural network inference on FPGAs," *IEEE Access,* vol. 8, pp. 83224-83237, 2020.

[21] J. Faraone *et al.*, "AddNet: Deep neural networks using FPGA-optimized multipliers,"

*IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 28, no. 1, pp. 115-128, 2019.

[22] C. Wang, L. Gong, X. Li, and X. Zhou, "A ubiquitous machine learning accelerator with automatic parallelization on FPGA," *IEEE Transactions on Parallel and Distributed Systems,* vol. 31, no. 10, pp. 2346-2359, 2020.

[23] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of big Data,* vol. 8, pp. 1-74, 2021.

[24] X. Wang, C. Wang, J. Cao, L. Gong, and X. Zhou, "WinoNN: Optimizing FPGA-based convolutional neural network accelerators using sparse Winograd algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 39, no. 11, pp. 4290-4302, 2020.

[25] S. A. Khan, *Digital design of signal processing systems: a practical approach*. John Wiley & Sons, 2011.

[26] R. M. Hewlitt and E. Swartzlantler, "Canonical signed digit representation for FIR digital filters," in *2000 IEEE Workshop on SiGNAL PROCESSING SYSTEMS. SiPS 2000. Design and Implementation (Cat. No. 00TH8528)*, 2000: IEEE, pp. 416-426.

[27] M. A. Soderstrand, "CSD multipliers for FPGA DSP applications," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, 2003, vol. 5: IEEE, pp. V-V.

[28] K. Schneider and A. Willenbücher, "A new algorithm for carry-free addition of binary signed-digit numbers," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014: IEEE, pp. 44-51.

[29] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*, 2016: PMLR, pp. 2849-2858.

[30] P. P. Chu, *FPGA Prototyping by SystemVerilog Examples: Xilinx MicroBlaze MCS SoC Edition*. John Wiley & Sons, 2018.

[31] J.-C. See *et al.*, "Cryptensor: A resource-shared co-processor to accelerate convolutional neural network and polynomial convolution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 42, no. 12, pp. 4735-4748, 2023.

[32] M. Vardhana and R. Pinto, "High-Performance Winograd Based Accelerator Architecture

for Convolutional Neural Network," *IEEE Computer Architecture Letters,* 2025.

มหาวิทยาลัยศิลปากร

# ประวัติผู้เขียน

| | |
|---|---|
| **ชื่อ-สกุล** | ธนพล ทองคำ |
| **วุฒิการศึกษา** | วิศวกรรมศาสตรบัณฑิต (วิศวกรรมอิเล็กทรอนิกส์ และระบบคอมพิวเตอร์) มหาวิทยาลัยศิลปากร |