หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้

PROGRAMMABLE ARTIFICIAL NEURAL NETWORK PROCESSING UNIT

จีรนันท์ บุญอยู่

Geranun Boonyuu

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้

PROGRAMMABLE ARTIFICIAL NEURAL NETWORK PROCESSING UNIT

จีรนันท์ บุญอยู่

Geranun Boonyuu

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้

PROGRAMMABLE ARTIFICIAL NEURAL NETWORK PROCESSING UNIT

โดย

จีรนันท์ บุญอยู่ 58010242

อาจารย์ที่ปรึกษา

ดร.สุเมฆ วิศยทักษิณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้

PROGRAMMABLE ARTIFICIAL NEURAL NETWORK PROCESSING UNIT

โดย

จีรนันท์ บุญอยู่ 58010242

อาจารย์ที่ปรึกษา

ดร.สุเมฆ วิศยทักษิณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา 2561

ภาควิชา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้

PROGRAMMABLE ARTIFICIAL NEURAL NETWORK PROCESSING UNIT

ผู้จัดทำ นายจีรนันท์ บุญอยู่ รหัสนักศึกษา 58010242

รายงานชิ้นนี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

Sumet Wit

(ดร. สุเมฆ วิศยทักษิณ)

อาจารย์ที่ปรึกษา

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อรายงาน หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้
นักศึกษา นายจีรนันท์ บุญอยู่ รหัสนักศึกษา 58010242
ภาควิชา วิศวกรรมอิเล็กทรอนิกส์
ปีการศึกษา 2561
อาจารย์ที่ปรึกษา ดร. สุเมฆ วิศยทักษิณ

บทคัดย่อ

เทคโนโลยีปัญญาประดิษฐ์และโครงข่ายประสาทเทียมกำลังเป็นที่สนใจในยุคปัจจุบัน เนื่องจากขจัดข้อจำกัดในการประมวลผลในด้านความเร็วและการตัดสินใจ แต่เนื่องจากโครงข่ายประสาทเทียมโดยทั่วไปมีลักษณะโครงสร้างและใช้การประมวลผลที่ซับซ้อนใช้พลังงานสูงจึงไม่สามารถนำมาประยุกต์ใช้กับระบบฝังตัวได้ง่าย โครงงานนี้จัดทำขึ้นเพื่อพัฒนาโครงสร้างหน่วยประมวลผลโครงข่ายประสาทเทียมที่สามารถทำงานในระบบฝังตัวโดยสามารถโปรแกรมโหนดให้สามารถรองรับการประมวลผลที่หลากหลายแต่ยังสามารถประมวลผลได้รวดเร็ว โดยโครงสร้างจะประกอบด้วยหน่วยประมวลผลจำนวน 4 ชุดทำงานขนานกันแบบ Pipeline ในรูปแบบการประมวลผลแบบ Fix Point ขนาด 16 บิท รองรับโหนดต่างๆ เช่น อินพุตได้ถึง 800 โหนด, ส่วนของ Hidden จำนวนถึง 32 โหนด และเอาต์พุตจำนวนถึง 32 โหนด ข้อมูลของน้ำหนัก (Weight) และอื่นๆ จะถูกเก็บในหน่วยความจำขนาด 57344 ไบต์จำนวน 2 หน่วยความจำ นอกจากนี้หน่วยประมวลผลที่สร้างขึ้นได้ทำการทดสอบการประมวลลายมือตัวเลขอารบิกจากดาต้าเบสของ MNIST เพื่อตรวจสอบความถูกต้องของวงจรทั้งการทดสอบด้วยการจำลองการทำงานและการทดลองใน FPGA

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Project Title	Programable Artificial Neural Network Processing Unit
Student	Mr. Geranun Boonyuu Student ID 58011455
Degree	Bachelor of Engineering
Department	Electronic Engineering
Year	2018
Project Advisor	Dr. Sumek Wisayataksin

ABSTRACT

Nowadays, Artificial Intelligence Technology and Neural Network are interesting because it can break the limitation of the processing time and decision of human. However, Artificial Neural Networks (ANNs) are complex processing structures and high energy consumption which cannot implement on Embedded systems easily. In this project, we develop the processing unit structures of ANNs which are flexible, high performance, yet suitable for Embedded systems. The structure consists of processing nodes which can be programmed for various applications and fast computation. It comprises 4 parallel processing units working in pipeline fashion. The processing format is 16-bit fixed point which supports various node settings such as input node up to 800 nodes, hidden node up to 32 nodes and output node up to 32 nodes. Weight data and others are stored in 2 memories with the size of 57344 bytes. In addition, the processing units were tested with Digits handwriting databases from MNIST to examine the accuracy of circuit on both logic simulation platform and emulator on the Xilinx FPGA board.

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

ปริญญานิพนธ์หน่วยประมวลผลโครงข่ายประสาทเทียมแบบโปรแกรมได้นี้สำเร็จลุล่วงได้ด้วยความกรุณาจากอาจารย์สุเมฆ วิศยทักษิณ อาจารย์ที่ปรึกษาโครงงานที่ได้ให้คำเสนอแนะ แนวคิด ตลอดจนแก้ไขข้อบกพร่องต่าง ๆ มาโดยตลอด จนปริญญานิพนธ์เล่มนี้เสร็จสมบูรณ์ นักศึกษาจึงขอกราบขอบพระคุณเป็นอย่างสูง

ขอกราบขอบคุณคุณแม่ และคุณป้า ที่ให้คำปรึกษาในเรื่องต่าง ๆ เป็นกำลังใจที่ดีเสมอมา

ขอคุณเพื่อน ๆ ที่ช่วยให้คำแนะนำดี ๆ สำหรับการปรับแต่งตัวโครงงานให้ดูเป็นมิตรกับผู้ใช้งานมากขึ้น

จีรนันท์ บุญอยู่

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่1

บทนำ

1.1 ความเป็นมาและความสำคัญ

ทุกวันนี้โครงข่ายประสาทเทียมกลายเป็นสิ่งที่ถูกพูดถึงอยู่เป็นจำนวนมาก เนื่องด้วยความสามารถในการทำงานที่เท่าเทียมหรือบางทีอาจจะล้ำหน้ามนุษย์อย่างเราๆ แต่การที่จะได้รับโครงข่ายประสาทเทียมที่มีประสิทธิภาพสูงนั้น ย่อมจำเป็นต้องใช้ฮาร์ดแวร์ที่มีประสิทธิภาพสูงตามไปด้วย

1.2 จุดมุ่งหมายของโครงงาน

ต้องการเพิ่มประสิทธิภาพของโครงข่ายประสาทเทียมโดยเพิ่มความเร็วการคำนวณให้เป็นแบบขนาน และทำให้มันสามารถใช้งานได้ในระบบฝังตัว

1.3 สมมุติฐานการศึกษา

หน่วยประมวลผลโครงข่ายประสาทเทียมจะสามารถคำนวนผลลัพธ์แบบขนานพร้อมกัน 4 ชุด และทำงานแบบไปป์ไลน์ ซึ่งแน่นอนว่าจะต้องมีความเร็วมากกว่าการประมวลผลด้วยซอฟต์แวร์ นอกจากนี้เนื่องจากเป็นการออกแบบวงจรเอง วงจรที่ออกแบบนั้นจะต้องใช้ทรัพยากรให้คุ้มค่า และทำให้มันสามารถทำงานได้ในระบบฝังตัวได้

1.4 ขอบเขตการวิจัย

สร้างวงจรดิจิตอลที่ทำงานขนานกัน 4 ชุด แบบไปป์ไลน์สตอลสำหรับการคำนวณโครงข่ายประสาทเทียมที่สามารถรองรับอินพุตได้สูงสุด 800 โหนด, ฮิตเดนสูงสุด 32 โหนด และเอาต์พุตสูงสุด 32 โหนด

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ได้ตัวประมวลผลโครงข่ายประสาทเทียมที่สามารถใช้แก้ไขปัญหาต่างๆ ได้รวดเร็วกว่าการใช้ซอฟต์แวร์ และยังสามารถทำงานได้ในระบบฝังตัวได้

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2

ทฤษฎี

2.1 โครงข่ายประสาทเทียม

เป็นการจำลองสมองของมนุษย์ โดยที่สมองของมนุษย์นั้นจะประกอบด้วยหน่วยประมวลผลขนาดเล็ก และถูกเชื่อมต่อกันด้วยโครงข่ายประสาทจำนวนมาก ช่วยให้มนุษย์นั้นสามารถเรียนรู้และคิดวิเคราะห์ได้อย่างรวดเร็ว

ในส่วนของคอมพิวเตอร์ ไม่ได้มีโครงข่ายที่ซับซ้อนเหมือนกับสมองของเรา มันมีหน้าที่ทำตามคำสั่งที่ได้โปรแกรมไว้เท่านั้น ดังนั้นเมื่อต้องให้คอมพิวเตอร์เรียนรู้อะไรซักอย่าง จึงเป็นเรื่องยากที่จะเขียนโปรแกรมแบบปกติ ทำให้เกิดการจำลองแนวทางการเรียนรู้ของมนุษย์ ไปสู่คอมพิวเตอร์ด้วยโครงข่ายประสาทเทียม

2.2 การเรียนรู้ของมนุษย์

มีอยู่ด้วยกันหลายวิธีหนึ่งในนั้นคือการรับข้อมูล และเข้าใจได้ว่าข้อมูลที่รับเข้ามานั้นคืออะไร จากนั้นก็จะเกิดการจดจำ และเรียนรู้ในสิ่งคล้ายๆกันที่ได้รับเข้ามา สมองของมนุษย์แต่ละคนก็จะหาวิธีจดจำ เรียนรู้ และหาวิธีแยกแยะข้อแตกต่าง ว่ารูปแบบข้อมูลลักษณะนี้คืออะไร ยิ่งเมื่อได้รับข้อมูลที่แตกต่างกันเล็กน้อยเพื่อเรียนรู้มากเท่าไหร่ ก็จะยิ่งทำให้สามารถแยกแยะได้แม่นยำยิ่งขึ้น

2.3 การเรียนรู้ของโครงข่ายประสาทเทียม

Supervised Learning (การเรียนรู้แบบมีการสอน)คือ การเรียนรู้ที่มีการตรวจคำตอบเพื่อให้โครงข่ายประสาทเทียมมีการปรับปรุงตัว ข้อมูลที่ใช้สอนก็จะมีคำตอบไว้คอยตรวจสอบว่าถูกต้องหรือไม่ ถ้าไม่ก็จะปรับปรุงตัวให้เองเพื่อให้ได้คำตอบที่ถูกต้องมากยิ่งขึ้น

Unsupervised Learning (การเรียนรู้แบบไม่มีการสอน)คือ ไม่มีการตรวจคำตอบว่าถูกหรือผิด โครงข่ายประสาทเทียม จะจัดเรียงโครงสร้างด้วยตัวเองตามลักษณะของข้อมูล และผลลัพธ์ที่ได้โครงข่ายประสาทเทียม จะสามารถจัดหมวดหมู่ของข้อมูลได้

2.4 Machine กับ Machine Learning

โปรแกรมคอมพิวเตอร์ทั่วไป เป็นการเขียนโปรแกรมเพื่อระบุการทำงานด้วยเงื่อนไขต่างๆ ที่มนุษย์เป็นตัวกำหนดเอง(ไม่มีการเรียนรู้) มันจะทำตามคำสั่งที่ถูกโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แต่ใน Machine learning เป็นการใช้แบบจำลองการเรียนรู้ของสมองมนุษย์ ซึ่งไม่ต้องเขียนโปรแกรมที่ระบุกระบวนการทำงานตรงๆ แต่จะให้ข้อมูล และผลลัพธ์ เข้าไปสอนให้คอมพิวเตอร์ให้พัฒนา และเรียนรู้ได้ ต่อมาเมื่อเราป้อนข้อมูลใหม่เข้าไป มันก็จะสามารถให้คำตอบออกมาได้ โดยอ้างอิงกับสิ่งที่มันได้เรียนรู้ไปแล้ว

2.5 เซลล์ประสาท

ส่วนที่เล็กที่สุดของโครงข่ายประสาทเทียมคือ เซลล์ประสาท ซึ่งมีหน้าที่คำนวณข้อมูลที่ได้รับเข้ามา และมีส่วนประกอบต่างๆ ดังนี้

- Input คือข้อมูลที่ได้รับเข้ามา
- Target คือสิ่งที่ให้ต้องการให้เป็น
- Weight คือการให้น้ำหนักกับแต่ละอินพุตที่เข้ามา
- Bias คือค่าที่บวกเข้าไปเพื่อทำให้ได้ผลลัพธ์ถูกต้องมากขึ้น
- Output คือผลลัพธ์
- Back Propagation คืออัลกอริทึมชนิดหนึ่งที่เซลล์ประสาทเทียมใช้ปรับปรุงตัวเองโดยนำค่าความผิดพลาดระหว่างเอาต์พุตและทาร์เก็ต ไปปรับปรุงค่าน้ำหนัก และค่าไบแอส เพื่อให้ได้ผลลัพธ์ที่ใกล้เคียงกับทาร์เก็ตมากที่สุด

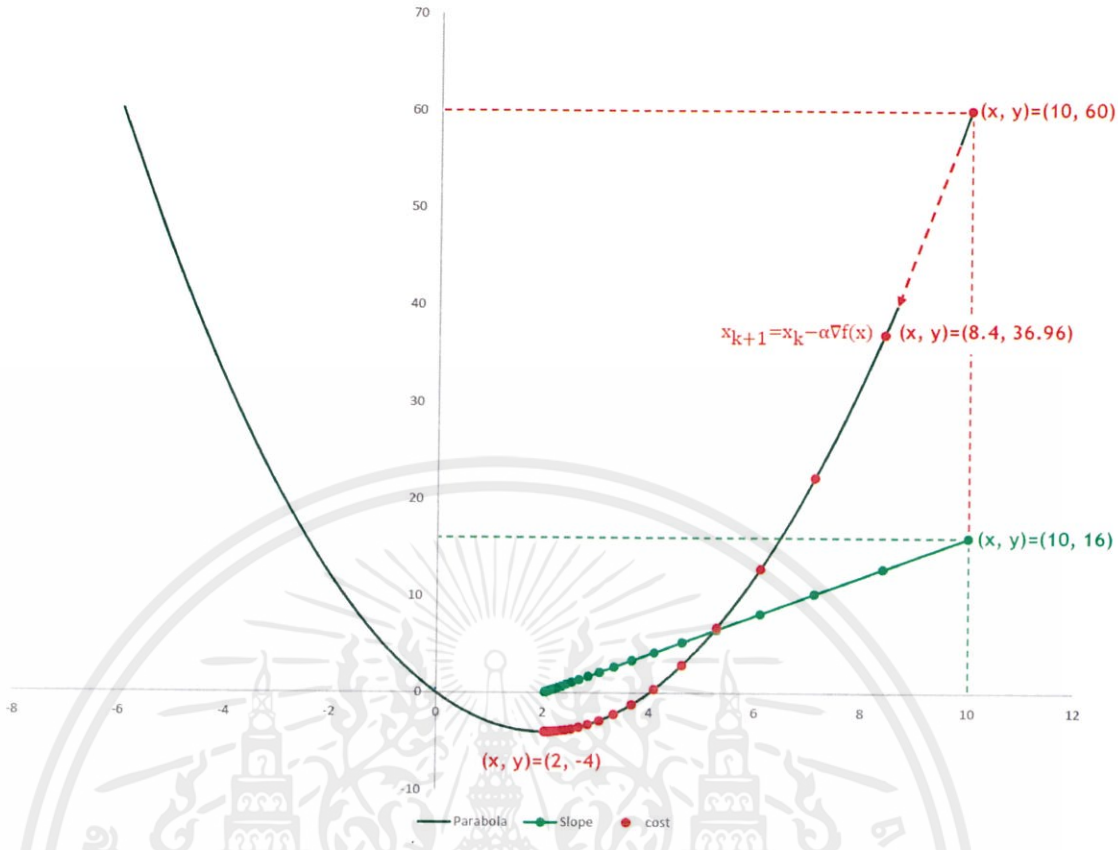
2.6 Back Propagation

การแพร่ย้อนหลัง หรือการแพร่ข้อผิดพลาดย้อนหลัง เป็นวิธีการเรียนรู้ของโครงข่ายประสาทเทียม แบบมีผู้ฝึกสอน ซึ่งมีความสำคัญมากในการปรับเปลี่ยนค่าน้ำหนักของโครงข่ายประสาทเทียม การปรับเปลี่ยนค่าน้ำหนักเรียกว่าการเรียนรู้

2.6.1 Gradient Descent

อัลกอริทึม ที่ใช้หาจุดที่เหมาะสมที่สุดของฟังก์ชั่นหรือ Cost ฟังก์ชั่น โดยการวนหาค่าที่ทำให้ cost มีค่าต่ำสุด จากการคำนวณความชัน ณตำแหน่งปัจจุบัน แล้วพยามเดินไปทิศทางที่ตรงข้ามกับความชัน ตัวอย่างการหา Gradient descent ของพาราโบลาหงายซึ่งมีสมการคือ $f(x)=x^2-4x$ และมี Gradient คือ $f'(x)=\nabla f(x)=2x-4$ โดนสามารถเริ่มต้นที่จุดใดๆ ก็ได้บนพาราโบลานี้ เช่นจุดที่ $x=10$ จากนั้นก็หาความชัน ณที่ตำแหน่งนั้นดังนี้ $\nabla f(x)=2(10)-4=16$ แล้วพยามเลื่อนค่า x ไปในทิศทางที่ตรงข้ามกับความชัน กำหนดให้ $\alpha=0.1$ ทำซ้ำไปเรื่อยๆ จนกระทั่ง ความชันมีค่าเท่ากับ 0 ทิศทางการเลื่อนของค่า x สามารถเขียนเป็นสมการได้ดังนี้ $x_{new}=x_{old}-\alpha\nabla f(x)$

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 1 กราฟตัวอย่างการลู่เข้า

2.6.2 การเรียนรู้ของโครงข่ายประสาทเทียม

การคำนวณของโครงข่ายประสาทเทียมประกอบไปด้วย 2 ขั้นตอนคือ ไปข้างหน้าใช้สำหรับหาคำตอบของโครงข่าย และแพร่กลับหลังใช้สำหรับปรับปรุงค่าน้ำหนัก เพื่อให้การคำนวณไปข้างในรอบถัดไปมีความถูกต้องมากขึ้น

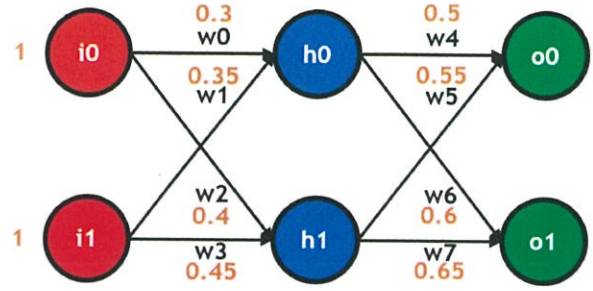
2.6.3 ตัวอย่างการคำนวณของโครงข่ายประสาทเทียม

สำหรับการแก้ปัญหาว่าวัตถุใดเป็นแอปเปิล หรือกล้วยโดยที่ “i_0” คือการรับค่าสีของวัตถุ “1” คือสีแดง และ “0” คือสีเหลืองสำหรับ “i_1” คือการรับค่ารูปทรงของวัตถุ “1” คือวัตถุกลม และ“0” คือวัตถุยาว

กำหนดให้ α=4.8 และสอนโครงข่ายว่าวัตถุนี้คือเปิล โดยให้“t_0”จะมีค่าเป็น“1” และ “t_1” จะมีค่าเป็น“0”

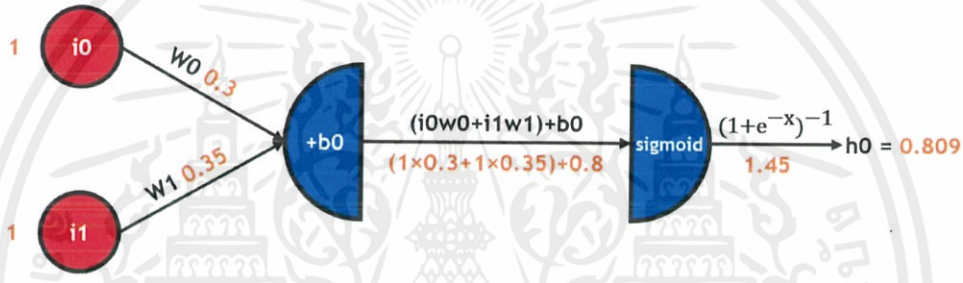
ขั้นตอนแรก การคำนวณแบบไปข้างหน้า โดยสุ่มค่าน้ำหนักขึ้นมาซึ่งเป็นเลขอะไรก็ได้ที่ไม่เท่ากับ 0 ไม่ซ้ำกัน และเหมาะสมกับ “Activate Function”

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2 โครงข่ายอย่างง่ายที่มีค่าน้ำหนัก

คำตอบของแต่ละโหนดสามารถหาได้ตามรูปด้านล่างนี้



รูปที่ 3 วิธีการหาผลลัพธ์ของแต่ละโหนด

สมการผลลัพธ์ของแต่ละโหนด

$net_0^0 = [i_0w_0+i_1w_1]+b_0 = [1\times0.3+1\times0.35]+0.8 = 1.45$

$net_1^0 = [i_0w_2+i_1w_3]+b_1 = [1\times0.4+1\times0.45]+0.85 = 1.7$

$h_0 = (1+e^{-net_0^0})^{-1} = (1+e^{-1.45})^{-1} = 0.809$

$h_1 = (1+e^{-net_1^0})^{-1} = (1+e^{-1.7})^{-1} = 0.846$

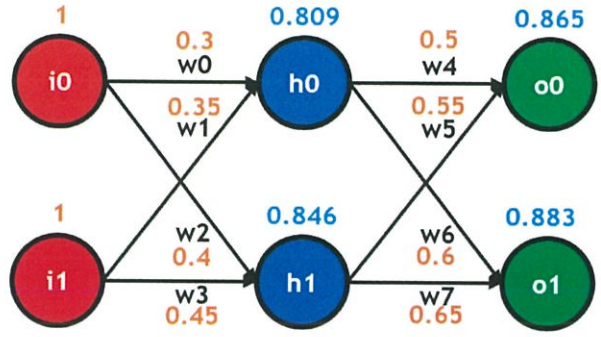
$net_0^1 = [h_0w_4+h_1w_5]+b_2 = [0.809\times0.5+0.846\times0.55]+0.9 = 1.86$

$net_1^1 = [h_0w_6+h_1w_7]+b_3 = [0.809\times0.6+0.846\times0.65]+0.95 = 2.02$

$o_0 = (1+e^{-net_1^0})^{-1} = (1+e^{-1.86})^{-1} = 0.865$

$o_1 = (1+e^{-net_1^0})^{-1} = (1+e^{-2.02})^{-1} = 0.883$

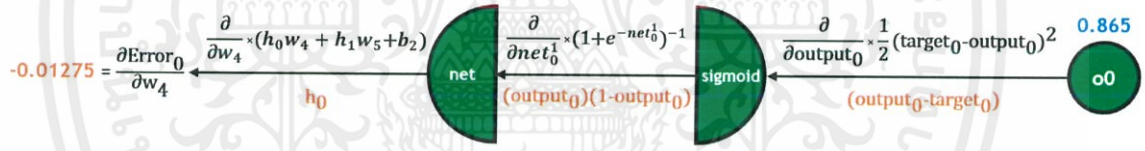
เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4 ผลลัพธ์ของการคำนวณแบบไปข้างหน้า

เมื่อได้คำคอบครบทุกโหนด พบว่าที่โหนด “o_1” มีค่ามากกว่า “o_0” ตอนนี้โครงข่ายประสาทเทียมจะเข้าใจว่าอินพุตที่ถูกป้อนเข้ามาเป็น “กล้วย” ซึ่งเป็นคำตอบที่ผิด

ขั้นตอนต่อมาคือ แพร่ความผิดพลาดย้อนกลับซึ่งเป็นการเรียนรู้ของโครงข่าย เริ่มต้นด้วยการปรับปรุงค่าน้ำหนักใหม่ของชั้นสุดท้าย โดยการแพร่กลับหลังชั้นเดียว รูปxx แสดงการแพร่กลับหลังเพื่อปรับปรุงค่าน้ำหนักตัวที่ 4 ค่าน้ำหนักตัวอื่นๆ ก็ทำเช่นเดียวกันแต่จะมีความแตกต่างกันเล็กน้อย



รูปที่ 5 แพร่ความผิดพลาดย้อนกลับชั้นเดียว

สมการอัตราการเปลี่ยนแปลงของค่าน้ำหนักแต่ละตัว

$$\frac{\partial Error_0}{\partial w_4} = [(output_0 - target_0)(output_0)(1-output_0)]\times h_0$$

$$\frac{\partial Error_0}{\partial w_5} = [(output_0 - target_0)(output_0)(1-output_0)]\times h_1$$

$$\frac{\partial Error_1}{\partial w_6} = [(output_1 - target_1)(output_1)(1-output_1)]\times h_0$$

$$\frac{\partial Error_1}{\partial w_7} = [(output_1 - target_1)(output_1)(1-output_1)]\times h_1$$

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แทนค่า

$$\frac{\partial Error_0}{\partial w_4} = [(0.865-1)(0.865)(1-0.865)]\times 0.809 = -0.01275$$

$$\frac{\partial Error_0}{\partial w_5} = [(0.865-1)(0.865)(1-0.865)]\times 0.846 = -0.01334$$

$$\frac{\partial Error_1}{\partial w_6} = [(0.883-0)(0.883)(1-0.883)]\times 0.809 = 0.0738$$

$$\frac{\partial Error_1}{\partial w_7} = [(0.883-0)(0.883)(1-0.883)]\times 0.846 = 0.0772$$

เมื่อได้ *Gradient* แล้วก็นำมาเข้าสมการเพื่อหาค่าน้ำหนักใหม่ได้ดังนี้

$$w_n^{new} = w_n^{old} - \alpha \frac{\partial Error_{total}}{\partial w_n}$$

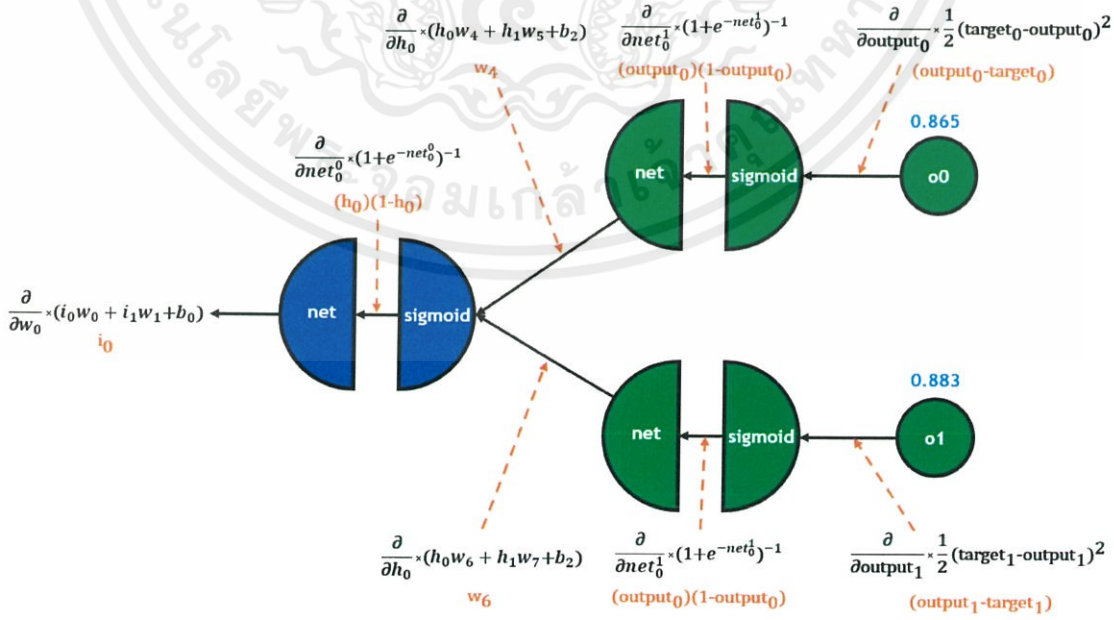
$$w_4^{new} = 0.5-[4.8\times(-0.01275)] = 0.561$$

$$w_5^{new} = 0.55-[4.8\times(-0.01334)] = 0.614$$

$$w_6^{new} = 0.6-[4.8\times(0.0738)] = 0.246$$

$$w_7^{new} = 0.65-[4.8\times(0.0772)] = 0.279$$

ต่อมาเป็นการปรับปรุงค่าน้ำหนักของชั้นแรก ซึ่งเป็นการแพร่กลับหลังหลายชั้นโดยจะต้องผ่านชั้นฮิตเดนก่อน รูป *xx* เป็นการแสดงการแพร่กลับหลังที่ผ่านชั้นฮิตเดน



รูปที่ 6 แพร่ความผิดพลาดย้อนกลับหลายชั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สมการอัตราการเปลี่ยนแปลงของค่าน้ำหนักตัวที่ 0 และ 1

$$\frac{\partial Error_{total}}{\partial h_0} = \frac{\partial Error_0}{\partial h_0} + \frac{\partial Error_1}{\partial h_0}$$

$$\frac{\partial Error_0}{\partial h_0} = [(output_0 - target_0)(output_0)(1-output_0)] \times w_4$$

$$\frac{\partial Error_1}{\partial h_0} = [(output_1 - target_1)(output_1)(1-output_1)] \times w_6$$

$$\frac{\partial h_0}{\partial net_0^0} = (h_0)(1-h_0)$$

$$\frac{\partial net_0^0}{\partial w_0} = i_0$$

$$\frac{\partial net_0^0}{\partial w_1} = i_1$$

แทนค่า

$$\frac{\partial Error_0}{\partial h_0} = [(0.865-1)(0.865)(1-0.865)] \times 0.5 = -0.0079$$

$$\frac{\partial Error_1}{\partial h_0} = [(0.883-0)(0.883)(1-0.883)] \times 0.6 = 0.0547$$

$$\frac{\partial h_0}{\partial net_0^0} = (0.809)(1-0.809) = 0.1545$$

$$\frac{\partial net_0^0}{\partial w_0} = i_0 = 1$$

$$\frac{\partial net_0^0}{\partial w_1} = i_1 = 1$$

$$\frac{\partial Error_0}{\partial w_0} = \left[\frac{\partial Error_0}{\partial h_0} + \frac{\partial Error_1}{\partial h_0}\right] \times \frac{\partial h_0}{\partial net_0^0} \times \frac{\partial net_0^0}{\partial w_0}$$

$$\frac{\partial Error_0}{\partial w_0} = [-0.0079+0.0547] \times 0.1545 \times 1 = 0.0072$$

$$\frac{\partial Error_0}{\partial w_1} = [-0.0079+0.0547] \times 0.1545 \times 1 = 0.0072$$

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปรับค่าน้ำหนักตัวที่ 2 และ 3

$$\frac{\partial Error_0}{\partial h_1} = [(output_0 - target_0)(output_0)(1-output_0)]\times w_5$$

$$\frac{\partial Error_1}{\partial h_1} = [(output_1 - target_1)(output_1)(1-output_1)]\times w_7$$

$$\frac{\partial h_1}{\partial net_1^0} = (h_1)(1-h_1)$$

$$\frac{\partial net_1^0}{\partial w_2} = i_0 = 1$$

$$\frac{\partial net_1^0}{\partial w_3} = i_1 = 1$$

แทนค่า

$$\frac{\partial Error_0}{\partial h_1} = [(0.865-1)(0.865)(1-0.865)]\times 0.55 = -0.0087$$

$$\frac{\partial Error_1}{\partial h_1} = [(0.883-0)(0.883)(1-0.883)]\times 0.65 = 0.0593$$

$$\frac{\partial h_1}{\partial net_0^0} = (0.846)(1-0.846) = 0.1303$$

$$\frac{\partial Error_0}{\partial w_3} = [\frac{\partial Error_0}{\partial h_1} + \frac{\partial Error_1}{\partial h_1}]\times \frac{\partial h_1}{\partial net_1^0} \times \frac{\partial net_1^0}{\partial w_3}$$

$$\frac{\partial Error_0}{\partial w_3} = [-0.0087+0.0593]\times 0.1303\times 1 = 0.0066$$

$$\frac{\partial Error_0}{\partial w_1} = [-0.0087+0.0593]\times 0.1303\times 1 = 0.0066$$

แทนค่าในสมการ Gradient Decent เพื่อปรับปรุงค่าน้ำหนักใหม่

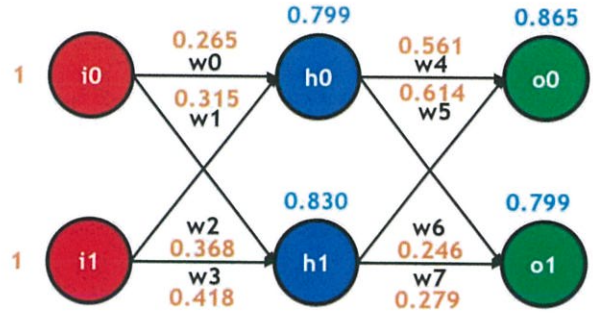
$$w_0^{new} = 0.3-[4.8\times(0.0072)] = 0.265$$

$$w_1^{new} = 0.35-[4.8\times(0.0072)] = 0.315$$

$$w_2^{new} = 0.4-[4.8\times(0.0066)] = 0.368$$

$$w_3^{new} = 0.45-[4.8\times(0.0066)] = 0.418$$

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7 ผลลัพธ์หลังเรียนรู้

หลังจากจบการเรียนรู้ เมื่อป้อนอินพุตเดิมเข้าไปโหนด“o_0”มีค่ามากกว่า“o_1”หมายความว่าตอนนี้โครงข่ายจะเข้าใจว่าวัตถุที่ป้อนเข้ามาคือแอปเปิลซึ่งเป็นคำตอบที่ถูกต้อง

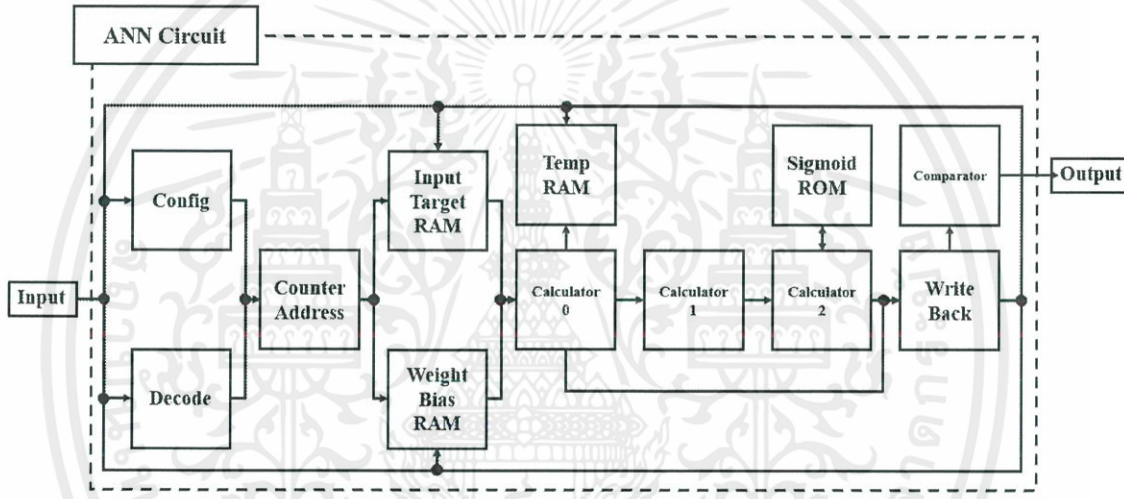
เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3

การออกแบบ

3.1 ลักษณะการทำงานที่ต้องการ

วงจรที่ถูกสร้างขึ้นเป็นระบบการคำนวณแบบ 16 บิต ประกอบด้วยตัวคำนวณ 4 ชุด และทำงานแบบไปป์ไลน์-สตอล เพื่อให้สามารถคำนวณด้วยความเร็วสูง



รูปที่ 8 การเชื่อมต่อของวงจรแต่ละส่วน

บล็อกตั้งค่า ทำหน้าที่กำหนดรอบการทำงานของบล็อกนับที่อยู่

บล็อกถอดรหัส ทำหน้าที่ถอดรหัสสัญญาณที่เข้ามาเพื่อนำไปใช้ควบคุมส่วนต่างๆของวงจร

บล็อกนับที่อยู่ ทำหน้าที่ส่งที่อยู่สำหรับการคำนวณของแต่ละสถานะเครื่องไปยังหน่วยความจำ

บล็อกหน่วยความจำ ทำหน้าที่เก็บข้อมูล ตัวทดในการคำนวณ และโหลดกลับมาใช้งานในสถานะเครื่องถัดไป

บล็อกคำนวณศูนย์ ทำหน้าที่คูณ เลื่อนบิต และรวมผลลัพธ์แบบซ้อนทับ

บล็อกคำนวณหนึ่ง ทำหน้าที่รวมผลลัพธ์จากส่วนที่แล้ว จนกระทั่งเปลี่ยนโหนดฮิตเดน

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บล็อกคำนวณสอง ทำหน้าที่นำผลลัพธ์จากบล็อกคำนวณหนึ่งบวกกับค่าไบแอส หลังจากนั้นก็จะส่งเข้าไปที่ฟังก์ชั่นซิกมอยด์ซึ่งอยู่ภายในบล็อกนี้

บล็อกเขียนข้อมูลกลับ ทำหน้าที่นำผลลัพธ์การคำนวณต่างๆ กลับเข้าไปเก็บไว้ที่หน่วยความจำเพื่อโหลดกลับมาใช้งานในการคำนวณรอบต่อไป

บล็อกเปรียบเทียบ ทำหน้าที่เปรียบเทียบผลลัพธ์สุดท้ายว่าที่ตำแหน่งใดของชั้นเอาต์พุต มีค่ามากที่สุดเพื่อส่งทำแหน่งนั้นๆ ออกมา

3.1 การแบ่งสถานะเครื่อง

การคำนวณของโครงข่ายประสาทเทียมประกอบด้วยการคำนวณไปข้างหน้า และแพร่ย้อนกลับ ซึ่งการคำนวณเหล่านี้มีความซับซ้อนมาก ดังนั้นจึงต้องแบ่งการคำนวณออกเป็นส่วนย่อยๆ และกำหนดสถานะเครื่องให้กับแต่ละส่วนที่ถูกแบ่งออกดังนี้

สถานะ 0 การคำนวณไปข้างหน้าเพื่อหา ผลรวมการคูณระหว่างค่าอินพุตกับค่าน้ำหนัก, เอาต์พุตของฮิตเดนโหนด และค่าก่อนซิกมอยด์(ค่าสูงสุดลบด้วยเอาต์พุต) ตามลำดับ

- $net_n^0 = a_0w_0+...+a_nw_n$
- $output_n^0 = (1+e^{-net_n^0})^{-1}$
- before differential $sigmoid_n^0 = (1-output_n^0)$

สถานะ 1 การคำนวณไปข้างหน้าเพื่อหา ผลรวมการคูณระหว่างค่าอินพุตกับค่าน้ำหนัก, เอาต์พุตของฮิตเดนโหนด, ค่าก่อนซิกมอยด์(ค่าสูงสุดลบด้วยเอาต์พุต) และค่าความผิดพลาดระหว่างโหนดเอาต์พุตที่ได้กับเอาต์พุตที่ออกแบบไว้ ตามลำดับ

- $net_n^1 = a_0w_0+...+a_nw_n$
- $output_n^1 = (1+e^{-net_n^1})^{-1}$
- before differential $sigmoid_n^1 = (1-output_n^1)$
- $error_n = (output_n^1-target_n)$

สถานะ 2 การคำนวณแพร่กลับหลังเพื่อหา ค่าซิกมอยด์ของชั้นแรก

- $sigmoid_n^0$ = (before differential $sigmoid_n^0$) × $output_n^0$

สถานะ 3 การคำนวณแพร่กลับหลังเพื่อหา ค่าซิกมอยด์ของชั้นสอง

- $sigmoid_n^1$ = (before differential $sigmoid_n^1$) × $output_n^1$

สถานะ 4 การคำนวณแพร่กลับหลังเพื่อหา ค่าก่อนเป็นนาบลาชั้นสอง(ก่อนได้ค่า Gradient)

- before $nabla_n^1 = error_n \times sigmoid_n^1$

สถานะ 5 การคำนวณแพร่กลับหลังเพื่อหา ผลรวมของค่าก่อนเป็นนาบลาชั้นสอง

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- summation of before $nabla_n^1 = \sum$ before $nabla_n^1 \times w_n^1$

สถานะ 6 การคำนวณแพร่กลับหลังเพื่อหา ค่าก่อนเป็นนาบลาชั้นแรก

- before $nabla_n^0$ = (summation of before $nabla_n^1$) × $sigmoid_n^0$

สถานะ 7 การคำนวณแพร่กลับหลังเพื่อหา ค่านาบลา(Gradient) ชั้นแรก

- $nabla_n^1$ = (before $nabla_n^1$) × $output_n^0$

สถานะ 8 การคำนวณแพร่กลับหลังเพื่อหา ค่าก่อนเป็นนาบลาชั้นสองไปอีกหน่วยความจำ

- ย้าย before $nabla_n^1$ ไปหน่วยความจำที่สอง

สถานะ 9 การคำนวณแพร่กลับหลังเพื่อหา ค่านาบลา(Gradient) ชั้นสอง

- $nabla_n^0$ = (before $nabla_n^0$) × $input_n$

สถานะ 10 การคำนวณเพื่อปรับค่าไบแอสของชั้นแรก

- $b_n^{new} = b_n^{old} - \alpha$before $nabla_n^1$

สถานะ 11 การคำนวณเพื่อปรับค่าไบแอสของชั้นสอง

- $b_n^{new} = b_n^{old} - \alpha$before $nabla_n^0$

สถานะ 12 การคำนวณเพื่อปรับค่าน้ำหนักของชั้นแรก

- $w_n^{new} = w_n^{old} - \alpha nabla_n^1$

สถานะ 13 การคำนวณเพื่อปรับค่าน้ำหนักของชั้นสอง

- $w_n^{new} = w_n^{old} - \alpha nabla_n^0$

รูปที่ 9 การคำนวณแบบไปข้างของสถานะเครื่องที่ 0 และ 1

3.2 รูปแบบของข้อมูลในหน่วยความจำ

ข้อมูลภายในหน่วยความจำแบ่งเป็น 2 รูปแบบ สีแดงคือเครื่องหมาย สีฟ้าคือจำนวนเต็ม สีเขียวคือทศนิยม แสดงให้เห็นในรูปด้านล่าง

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2.1 แบบทศนิยม 12 ตำแหน่ง

ใช้กับข้อมูลอินพุต เอาต์พุต และทาร์เก็ท เนื่องจากอินพุตของซิกมอยด์ฟังก์ชั่น ที่มีค่าตั้งแต่ -8 จนถึง 7.99 ก็สามารถให้คำตอบของฟังก์ชั่นซิกมอยด์มีค่าได้ตั้งแต่ 0.000335 จนถึง 0.999665 ซึ่งเป็นค่าที่สามารถยอมรับได้ วิธีการแปลงตัวเลขให้อยู่ในรูปทศนิยมทำได้ด้วยวิธีการต่อไปนี้

เริ่มจาก 0111111111111111_2 หรือ 32767_{10}

แปลงข้อมูลให้มีทศนิยมความละเอียด 12 ตำแหน่ง โดยไม่ต้องสนใจเครื่องหมาย

0111.111111111111_2 หรือ 7.999_{10}

$(1\times2^2)+(1\times2^1)+(1\times2^0)+(1\times2^{-1})+(1\times2^{-2})+...+(1\times2^{-12})$

$4+2+1+0.5+0.25+...+0.000244 = +7.999$

ถ้าเป็นซีกลบ แปลง 32767_{10} ด้วยวิธี 2'Complement

$0111111111111111_2 \rightarrow 1000000000000000_2$

$1000000000000000_2 + 1_2 \rightarrow 1000000000000001_2$ หรือ -32767_{10}

สังเกตว่าในซีกลบสามารถทำให้มีน้อยสุดได้ถึง -32768_{10} หรือ 1000000000000000_2

วิธีจัดรูปให้ทศนิยมมีความละเอียด 12 ตำแหน่งก็สามารถทำด้วยวิธีการเดียวกัน

รูปที่ 10 รูปแบบข้อมูลที่มีทศนิยม 12 ตำแหน่ง

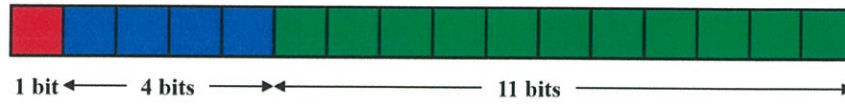
3.2.2 แบบทศนิยม 11 ตำแหน่ง

ค่าน้ำหนัก และค่าไบแอสเป็นค่าที่ถูกสุ่มขึ้นมาดังนั้นจึงต้องใช้โปรแกรมช่วยหาค่ามากสุดและน้อยสุดโดยใช้โปรแกรมโครงข่ายประสาทเทียมสำหรับการจำแนกตัวเลขที่เขียนด้วยลายมือ ที่มี

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

784 โหนดอินพุต 16 โหนดฮิตเดนและ 10 โหนดเอาต์พุตแล้วสั่งให้เรียนรู้ 50000 ตัวอย่าง 30 รอบ เก็บค่าน้ำหนัก และค่าไบแอสที่มากที่สุด และน้อยที่สุด

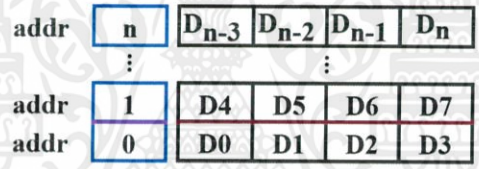
วิธีแปลงเลขให้อยู่ในรูป 16 บิทก็สามารถทำด้วยวิธีการเดียวกันกับหัวข้อก่อนหน้านี้



รูปที่ 11 รูปแบบข้อมูลที่มีทศนิยม 11 ตำแหน่ง

3.3 การออกแบบหน่วยความจำ

เนื่องจากการทำงานเป็นแบบ 16 บิต และขนานกัน 4 ชุดดังนั้น จึงต้องออกแบบหน่วยความจำให้เก็บข้อมูลครั้งละ 64 บิต ต่อหนึ่งที่อยู่ ข้อมูลชุดแรก D0 จะถูกต่อเข้ากับตัวประมวลผลชุดแรก ส่วนข้อมูลD1,D2 และ D3 ก็จะถูกต่อเข้ากับตัวประมวลผลชุดอื่นๆ ตามลำดับ



รูปที่ 12 ตำแหน่งของข้อมูลในหน่วยความจำ

จากการแบ่งสถานะย่อยๆ ทำให้เราต้องคำนวณพื้นที่สำหรับจัดเก็บตัวทดจากการคำนวณต่างๆ และเก็บค่าตัวทดได้ครบทั้ง 14 สถานะเครื่อง โดยแบ่งจำนวนข้อมูลต่างๆ ดังนี้

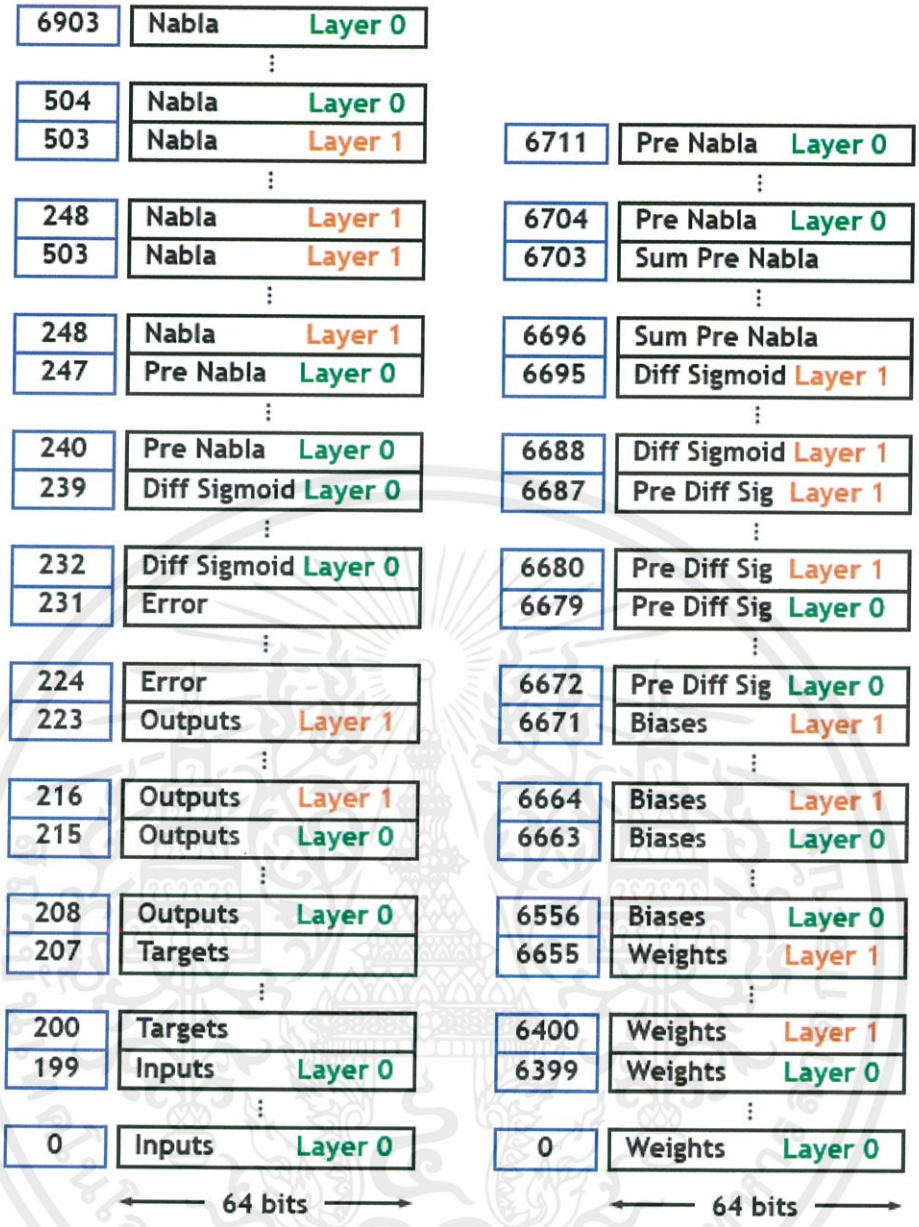
ข้อมูล	การคำนวณ	ที่อยู่ที่ใช้ (หารด้วย 4)
input	800	200
Target	32	8
Weight 0	800×32 = 25600	6400
Weight 1	32×32 = 1024	256
Bias 0	32	8
Bias 1	32	8
Error	32	8
Pre Diff sigmoid 0	32	8
Pre Diff sigmoid 1	32	8

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Diff Sigmoid 0	32	8
Diff Sigmoid 1	32	8
Pre Nabla 1	32	8
Sum Pre Nabla	32	8
Pre Nabla 0	32	8
Nabla 1	32×32 = 1024	256
Nabla 0	800×32 = 25600	6400
รวม		**13600**

ที่อยู่ที่ใช้มีจำนวน 13600 ที่อยู่ ดังนั้นจึงแบ่งหน่วยความจำออกเป็น 2 ก้อนที่มีขนาดเท่าๆกัน ก้อนละ 7168 ตำแหน่ง หรือก้อนละ 57344 ไบต์ และจะแบ่งที่อยู่ให้กับข้อมูลต่างอีก ดังรูปด้านล่างนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 13 ที่อยู่ของข้อมูลในแต่ละหน่วยความจำ

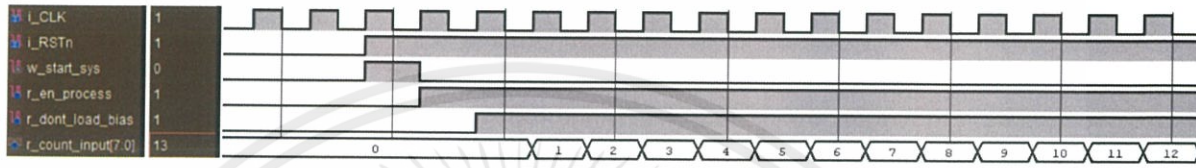
3.4 การออกแบบวงจรส่งที่อยู่และการตั้งค่าโหนด

ในส่วนนี้จะทำหน้าที่ส่งที่อยู่ไปที่หน่วยความจำ เนื่องจากว่ามีหน่วยความจำเพียงแค่ 2 หน่วยความจำ แต่พารามิเตอร์สำหรับการคำนวณมี 4 พารามิเตอร์ ดังนั้นวงจรนับนี้จะมีการหยุด 1 จังหวะเพื่อสลับไปยังอีกพารามิเตอร์หนึ่ง ระบบแบบนี้เรียกว่า Pipeline-Stall

และในส่วนตั้งค่าโหนดนั้นจะใช้สำหรับเป็นตัวกำหนดรอบการทำงานของวงจรนับภายในวงจรส่งที่อยู่ ซึ่งสามารถโปรแกรมโหนดได้จำกัดที่ 800 อินพุตโหนด 32 ฮิตเดนโหนด 32 เอาท์พุตโหนด อันเนื่องมากจากข้อจำกัดของหน่วยความจำที่ใช้งาน

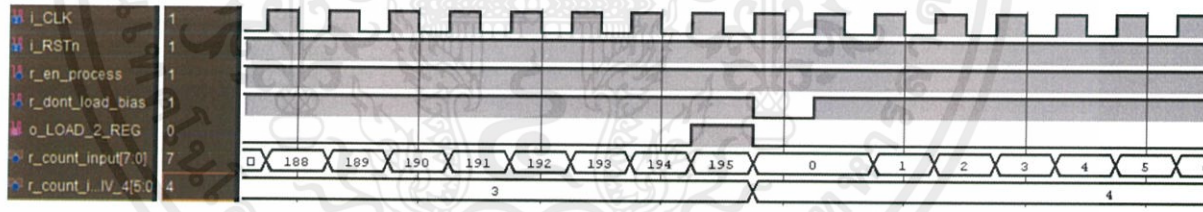
เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เริ่มจากสัญญาณส่งสัญญาณ start_sys เข้าไปในระบบ ต่อมาในสัญญาณนาฬิกาลูกถัดไป สัญญาณ en_process ก็จะกลายเป็น"1"ในจังหวะเดียวกันนี้เองที่สัญญาณ en_process = 1 และสัญญาณ dont_load_bias = 0 วงจรนับอินพุต และวงนับค่าน้ำหนัก จะถูกหยุดและสลับไปเป็นวงจรนับไบแอสแทน จากนั้นในสัญญาณนาฬิกาลูกถัดไปก็จะกลับมาใช้วงจรนับอินพุต และวงจรนับค่าน้ำหนัก เหมือนเดิม



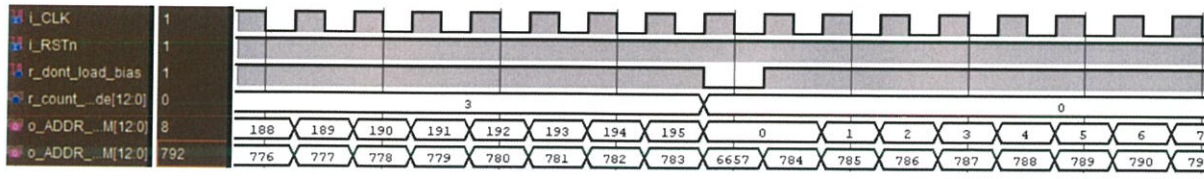
รูปที่ 14 เริ่มต้นการนับ

เมื่อวงจรนับส่งค่าที่อยู่ออกไปจนครบ 4 โหนดและถึงตัวสุดท้ายของโหนดที่ 4 (จากรูปก็คือ นับอินพุต = 195 และวงจรนับรอบโหนด = 3) ก็จะส่งสัญญาณ Load_2_Reg เพื่อทำการสลับไปวงจรนับไปที่วงจรนับไบแอสอีกครั้ง



รูปที่ 15 สลับตัวนับ

รูปด้านล่างนี้คือตัวอย่างเมื่อผ่านกระบวนการต่างๆ เรียบร้อยจนได้ที่อยู่ออกมา สังเกตุว่าเมื่อสัญญาณ dont_load_bias = 0 ที่อยู่ของวงจรนับค่าน้ำหนัก จะถูกสลับไปที่วงจรนับไบแอสแทน

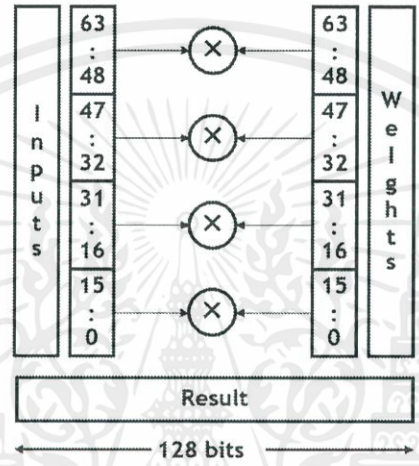


รูปที่ 16 ส่งที่อยู่ออกไป

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 การออกแบบตัวคูณ

ข้อมูลขนาด 64 บิต จากหน่วยความจำศูนย์ และหน่วยความจำหนึ่ง จะถูกส่งเข้าไปที่ตัวคูณทั้ง 4 ตัว โดยที่ข้อมูลบิตที่ 48-63 จะเข้าไปที่ตัวคูณที่หนึ่ง ข้อมูลบิตที่ 32-47 จะเข้าไปที่ตัวคูณที่สอง ข้อมูลบิตที่ 16-31 จะเข้าไปที่ตัวคูณที่สาม และสุดท้ายข้อมูลบิตที่ 0-15 จะเข้าไปที่ตัวคูณที่สี่ โดยที่ตัวคูณทั้งหมดนี้เป็นตัวคูณที่สามารถ คูณแบบคิดเครื่องหมาย และผลลัพธ์ที่ได้จะถูกขยายออกมาเป็นข้อมูล 32 บิต และสุดท้ายเอาต์พุตของวงจรนี้จะถูกเรียงต่อกันจนกลายเป็นข้อมูลขนาด 128 บิต

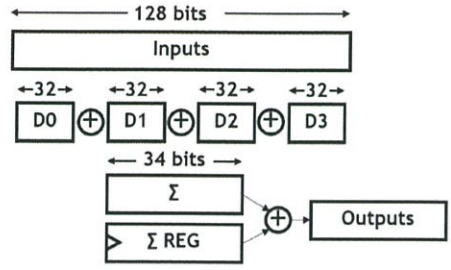


รูปที่ 17 รูปแบบการคูณ

3.4 การออกแบบตัวบวกแบบซ้อนทับ

ข้อมูลขนาด 128 บิต จะถูกแบ่งออกเป็น 4 ส่วน ส่วนละ 32 บิต และจะถูกบวกกันเอง ผลลัพธ์ที่ได้จะถูกขยายออกเป็นข้อมูลขนาด 34 บิต จากนั้นในสัญญาณนาฬิกาลูกถัดไป จะมีรีจิสเตอร์ขนาด 34 บิต บวกตัวเองไปเรื่อยๆห จนกว่าจะครบรอบตามที่กำหนดไว้ ในขณะเดียวกันนั้นรีจิสเจอร์ก็จะถูกรีเซ็ตตัวเองในสัญญาณนาฬิกาลูกถัดไป แต่ก่อนที่รีจิสเตอร์จะถูกรีเซ็ตค่า วงจรบวกชุดที่สองจะนำค่าของรีจิสเตอร์บวกด้วยค่าจากวงจรชุดแรก ผลลัพธ์ของวงจรชุดที่สองคือคำตอบที่แท้จริงของส่วนนี้

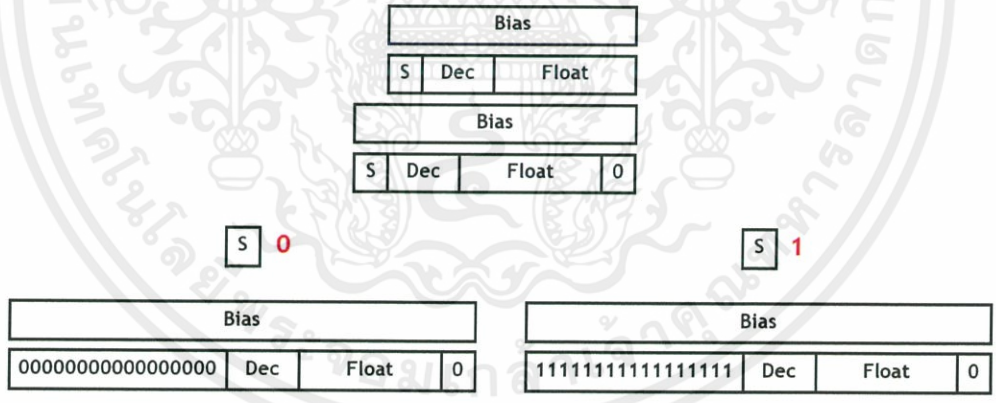
เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 18 การบวกผลลัพธ์แบบซ้อนทับ

3.5 การออกแบบตัวบวกค่าไบแอส

ค่าไบแอสที่ถูกโหลดมาจากหน่วยความจำหนึ่ง เป็นข้อมูลขนาด 16 บิต ซึ่งมีทศนิยมขนาด 11 บิต แต่ค่าของผลลัพธ์จากบล็อคบวกแบบทบค่า มีขนาด 34 บิต เป็นทศนิยม 12 บิต จึงไม่สามารถนำมาบวกกันตรงๆ ได้ ฉะนั้นจะต้องเลื่อนบิตของค่าไบแอสไปทางซ้าย 1 บิต เพื่อให้ค่าไบแอสมีทศนิยม 12 บิต และสุดท้ายก็นำบิตที่ 17 ของค่าไบแอสมาตรวจสอบว่าเป็นค่าลบหรือบวก ถ้าเป็นลบหมายความว่าบิตที่ 17 มีค่าเป็น “1” ก็จะถูกเติม “1” เข้าไปที่ด้านหน้าจนครบ 34 บิต ในทำนองเดียวกันถ้าเป็น “0” ก็จะถูกเติมค่า “0” เข้าไปแทน จึงจะสามารถนำค่าไบแอสบวกกับผลลัพธ์ของบล็อคบวกแบบซ้อนทับได้อย่างถูกต้อง

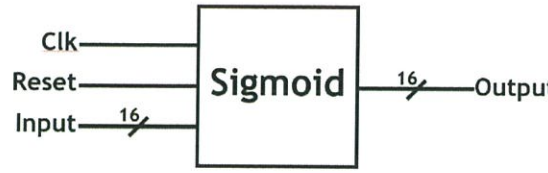


รูปที่ 19 การเลื่อนตำแหน่งของค่าไบแอส

3.6 การออกแบบวงจรคำนวณฟังก์ชั่นซิกมอยด์

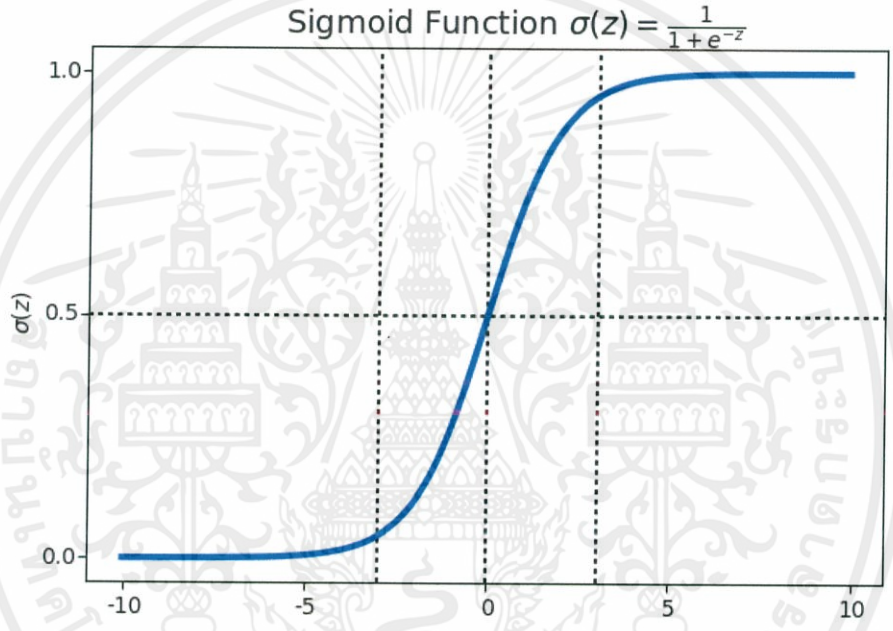
การออกแบบนั้นเริ่มจากเลือกสัญญาณที่จะใช้ทั้งหมดภายในบล็อกนี้ก่อนซึ่งจะประกอบไปด้วย สัญญาณนาฬิกา สัญญาณรีเซ็ต อินพุตขนาด 16 บิต และเอาต์พุตขนาด 16 บิต

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 20 วงจรซิกมอยด์

เนื่องจากฟังก์ชันซิกมอยด์มีลักษณะเป็นเส้นโค้ง ดังรูป ซึ่งไม่สามารถประมาณค่าด้วยเส้นตรงได้ครบทุกจุด ดังนั้นจึงได้มีการออกแบบฟังก์ชันซิกมอยด์โดยใช้ความจำ และการประมาณร่วมกัน



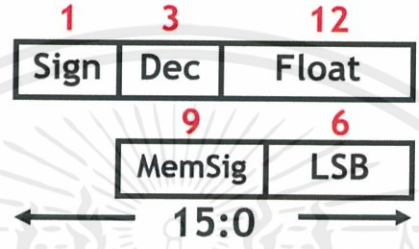
รูปที่ 21 กราฟผลลัพธ์ของฟังก์ชั่นซิกมอยด์

ซิกมอยด์ฟังก์ชั่นเป็นฟังก์ชันที่สมมาตรกัน อินพุตของซิกมอยด์ฟังก์ชั่นตั้งแต่ -8 จนถึง 7.99 ก็สามารถให้คำตอบของฟังก์ชั่นได้ตั้งแต่ 0.000335 จนถึง 0.999665 ดังนั้นค่าจริงของซิกมอยด์จะถูกเก็บแค่ซีกบวกเพียงซีกเดียว ซึ่งจะใช้หน่วยความจำที่มี 512 ตำแหน่ง และใช้ Python คำนวณสมการซิกมอยด์ ตั้งแต่ $\left(\frac{0\times8}{512}\right)$ จนถึง $\left(\frac{511\times8}{512}\right)$ โดยจะสเต็ปค่าครั้งละ $\left(\frac{1\times8}{512}\right)$ ค่าทั้ง 512 ตัวนี้จะขอเรียกว่า MemSi (x8 เพราะว่าค่าอินพุตของซิกมอยด์ที่ใช้อยู่ในช่วง -8 ถึง 8) ต่อมาก็คือค่า ΔY ใช้ Python เช่นเดียวกันโดยการนำค่า MemSi[n+1] ลบกับ MemSi[n] ซึ่งจะได้ออกมาทั้งหมด 511 ค่า และเราจะเติม 0 ลงไปเองที่ตำแหน่ง 512 จากนั้นก็จะนำค่า ΔY มาต่อที่ด้านหน้า MemSi รวมเป็น 17 บิต ซึ่งมีที่อยู่ทั้งหมด 512 ตำแหน่ง ดังนั้นหน่วยความจำนี้จะมีขนาด 1088 ไบต์

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 22 ตำแหน่งข้อมูลในหน่วยความจำของซิกมอยด์

ต่อมาจะเป็นการทำงานของวงจรซิกมอยด์ซึ่งจะรับอินพุตขนาด 16 บิตเข้ามาโดยประกอบด้วยส่วนต่างๆดังนี้

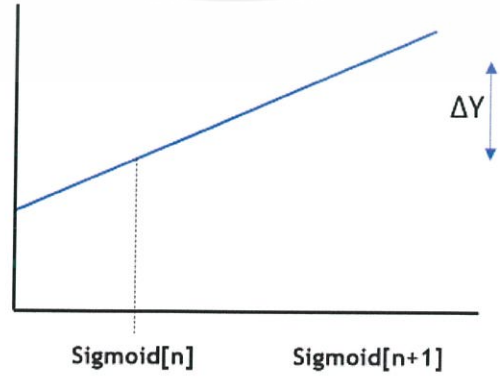


รูปที่ 23 ตำแหน่งของข้อมูลที่ใช้ประมาณ

ส่วนแรก ทำหน้าที่ตรวจสอบบิตที่ 15 ถ้าเป็น"0"ก็จะทำให้ผลลัพธ์ของซิกมอยด์มีค่าเป็นบวก แต่ถ้าบิตที่ 15 เป็น"1"ก็จะให้ผลลัพธ์ออกมาเป็นค่าลบแทน

ส่วนที่สอง จะใช้งานบิตที่ 14 ถึง 6 ซึ่งก็คือที่อยู่ของหน่วยความจำของซิกมอยด์ โดยหน่วยความจำนั้นจะส่งค่าออกมาทั้ง ΔY และ MemSig

ส่วนที่สาม จะใช้งานบิตที่ 5 ถึง 0 ทำหน้าที่เป็นตัวประมาณค่าตำแหน่งของ ΔY เมื่อได้รับค่า ΔY จากหน่วยความจำซิกมอยด์แล้ว ก็จะนำค่า LSB คูณกับ ΔY (หาความสูงของΔY) ผลลัพธ์ที่ได้จะมีขนาด 11 บิต ซึ่งยังไม่สามารถเอาไปรวมกับค่าซิกมอยด์จากหน่วยความจำได้ ดังนั้นจึงต้องทำการบวกด้วย 32 และทั้งหมดหารด้วย 64 ซึ่งเรียกวิธีการนี้ว่าเลื่อนทศนิยมให้เหลือ 5 ตำแหน่ง



เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 24 การประมาณค่าซิกมอยด์

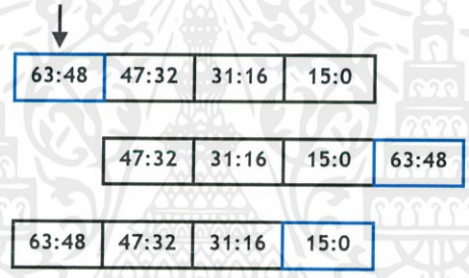
เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ส่วนที่สี่ นำผลลัพธ์ของส่วนที่สาม และ ส่วนที่สองเฉพาะ MemSig มาบวกกัน

ส่วนสุดท้ายจะนำค่าจากส่วนแรกมาเข้าที่มัลติเพล็กเซอร์ ถ้าเป็นค่าบวกผลลัพธ์ก็จะถูกส่งออกไปทันที แต่ถ้าเป็นค่าลบ ก็จะนำ 4096 ลบด้วย ผลลัพธ์ซิกมอยด์ ก่อนจึงจะส่งผลลัพธ์ซีกลบออกไปแทน

3.7 การออกแบบการเขียนข้อมูลกลับไปที่หน่วยความจำ

ผลลัพธ์ขนาด 16 บิตจากบล็อคซิกมอยด์ จะถูกเรียงต่อกัน 4 ข้อมูล โดยเริ่มจากข้อมูลแรกจะเข้าไปที่รีจิสเตอร์บิตที่ 63-48 ในสัญญาณนาฬิกาลูกถัดไปก็วงจรก็จะเลื่อนไปทางซ้าย 16 บิต และเมื่อข้อมูลใหม่เข้ามาก็จะถูกโหลดลงไปในรีจิสเตอร์บิตที่ 63-48 เหมือนเดิมจนกระทั่งครบทั้ง 4 ข้อมูล เมื่อครบแล้ววงจรส่วนนี้ก็จะส่งข้อมูลขนาด 64 บิตไปที่หน่วยความจำ ในขณะเดียวกันวงจรนับที่อยู่ภายในก็จะส่งที่อยู่ออกไปพร้อมกัน



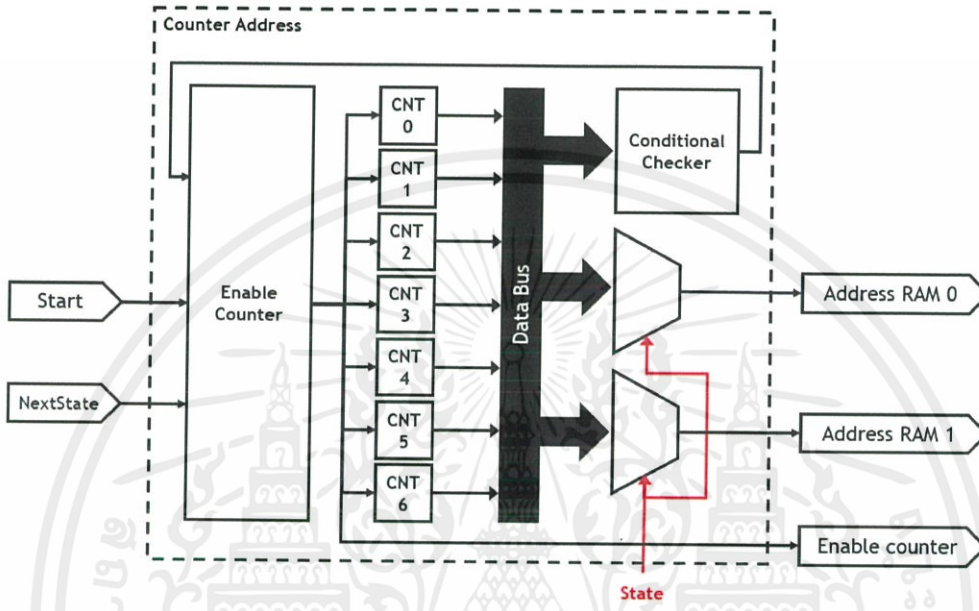
รูปที่ 25 การโหลดข้อมูลลงรีจิสเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.8 ส่วนควบคุมวงจรแรก

สัญญาณเปิดใช้งานตัวนับจะถูกส่งไปที่บล็อกเขียนข้อมูลกลับเพื่อนำไปสร้างสัญญาณสถานะถัดไปสำหรับสถานะที่ 2 ถึง 13

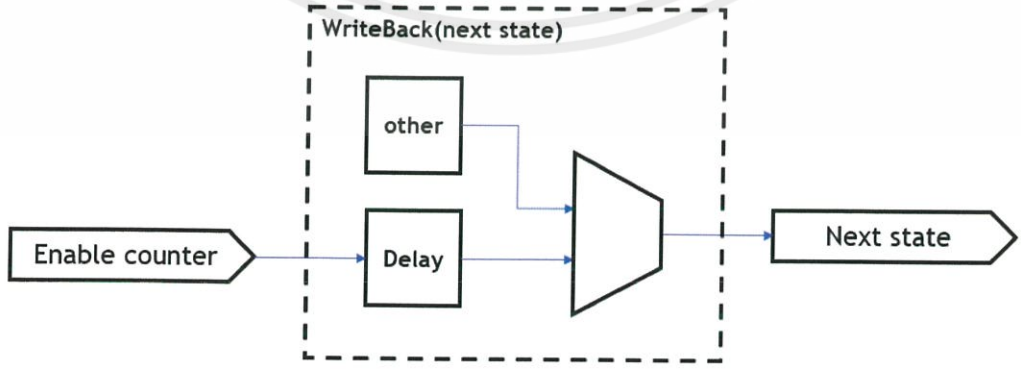
สัญญาณสถานะถัดไปของสถานะที่ 0 และ 1 จะใช้สัญญาณโหลดข้อมูลของตัวคำนวณสอง



รูปที่ 26 บล็อคไดอะแกรมควบคุมวงจรหนึ่ง

3.9 ส่วนควบคุมวงจรสอง

นำสัญญาณเปิดใช้งานมาเข้าดีเลย์ก่อน และส่งเข้าไปที่มัลติเพล็กซ์เซอร์เพื่อเลือกสัญญาณสถานะถัดไป



รูปที่ 27 บล็อคไดอะแกรมควบคุมวงจรสอง

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

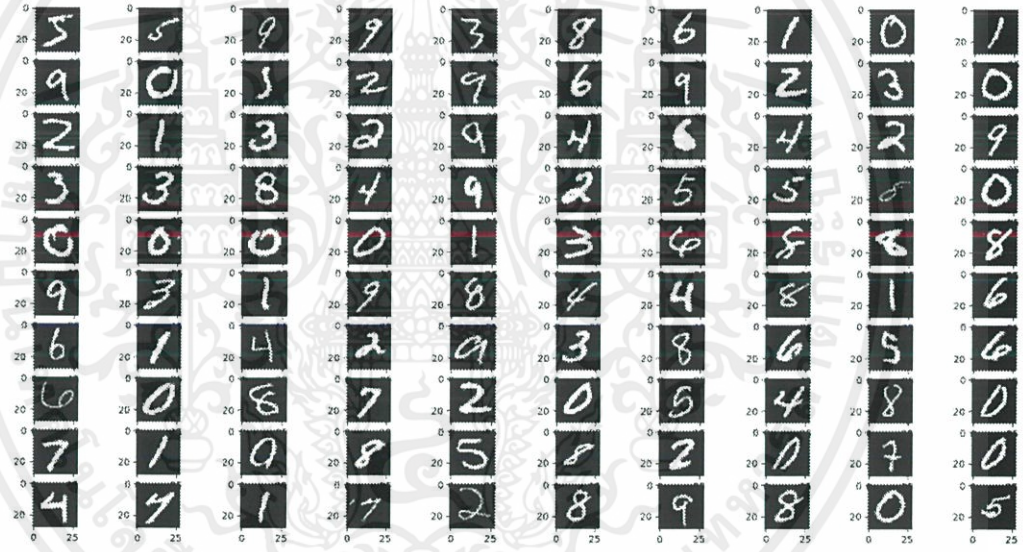
บทที่ 4

ผลการทดลอง

4.1 ความเร็วและความถูกต้องของวงจร

ข้อมูลที่ใช้ทดสอบการทำงานของวงจร เป็นข้อมูลตัวเลขที่เขียนด้วยลายมือซึ่งมีขนาด 28x28 พิเซล หรืออินพุต 784 โหนด โดยกำหนดระบบการทำงานดังต่อไปนี้

- ฮิตเดนเลเยอร์ 16, 24 และ 32 โหนดตามลำดับ และมีเอาท์พุตเลเยอร์ 10 โหนด
- ใช้สัญญาณนาฬิกา 72 MHz



รูปที่ 28 ตัวอย่างข้อมูลตัวเลขที่เขียนด้วยลายมือของ MNIST

การจำลองการทำงานของวงจรบน Vivado ใช้เวลานาน ดังนั้นจึงจำลองการทำงานการเรียนรู้ของโครงข่ายประสาทเทียม 16, 24 และ 32 โหนดแล้วนำผลไปเทียบเปรียบกับ Python ต่อมาเพื่อความสะดวกและรวดเร็วในการตรวจสอบความถูกต้อง จึงได้เขียนโปรแกรมโครงข่ายประสาทเทียมที่เป็นระบบการคำนวณแบบ 16 บิตซึ่งจะทำงานขนานไปกับโครงข่ายประสาทเทียมแบบธรรมดา และวิธีนับสัญญาณนาฬิกาของโครงข่ายประสาทเทียมที่เป็นโปรแกรมได้ใช้โปรแกรม “UVISION” ส่งค่าอินพุตขนาด 784 และเปลี่ยนฮิตเดนเลเยอร์ 16, 24 และ 32 ตามลำดับ ผลลัพธ์ที่ได้จากการตรวจสอบวงจรมีสองส่วนคือความเร็ว และความถูกต้องดังตารางด้านล่างนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหนด	FPGA			โปรแกรม		
	เรียนรู้ (clk)	ทดสอบ (clk)	ความถูกต้อง (%)	เรียนรู้ (clk)	ทดสอบ (clk)	ความถูกต้อง (%)
784 : 16 : 10	6,454	3,198	92.86	304,597	116,276	92.98
784 : 24 : 10	9,669	4,794	94.62	453,463	172,370	94.45
784 : 32 : 10	12,873	6,386	95.21	602,327	228,470	95.42

4.2 ระยะเวลาที่นานที่สุดในการคำนวณของวงจร

WNS คือระยะเวลาการทำงานของวงจรที่ช้าที่สุดก่อนจะถึงสัญญาณนาฬิกาลูกถัดไป วงจรนี้ใช้เวลานานที่สุดคือ 10ns + 3.052ns = 13.052ns ดังนั้นความเร็วของสัญญาณนาฬิกาสูงสุดที่ทำได้คือ 76.6MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-3.052 ns	Worst Hold Slack (WHS):	0.050 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	-660.575 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	369	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5671	Total Number of Endpoints:	5671	Total Number of Endpoints:	1756

Timing constraints are not met.

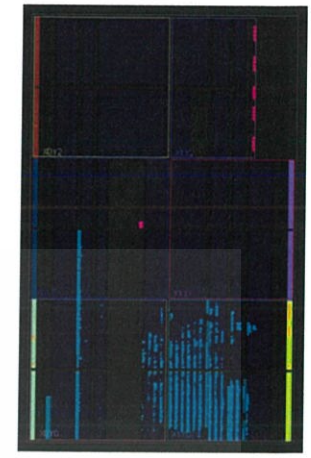
รูปที่ 29 ผลการจำลองระยะเวลาของพาร์ท

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

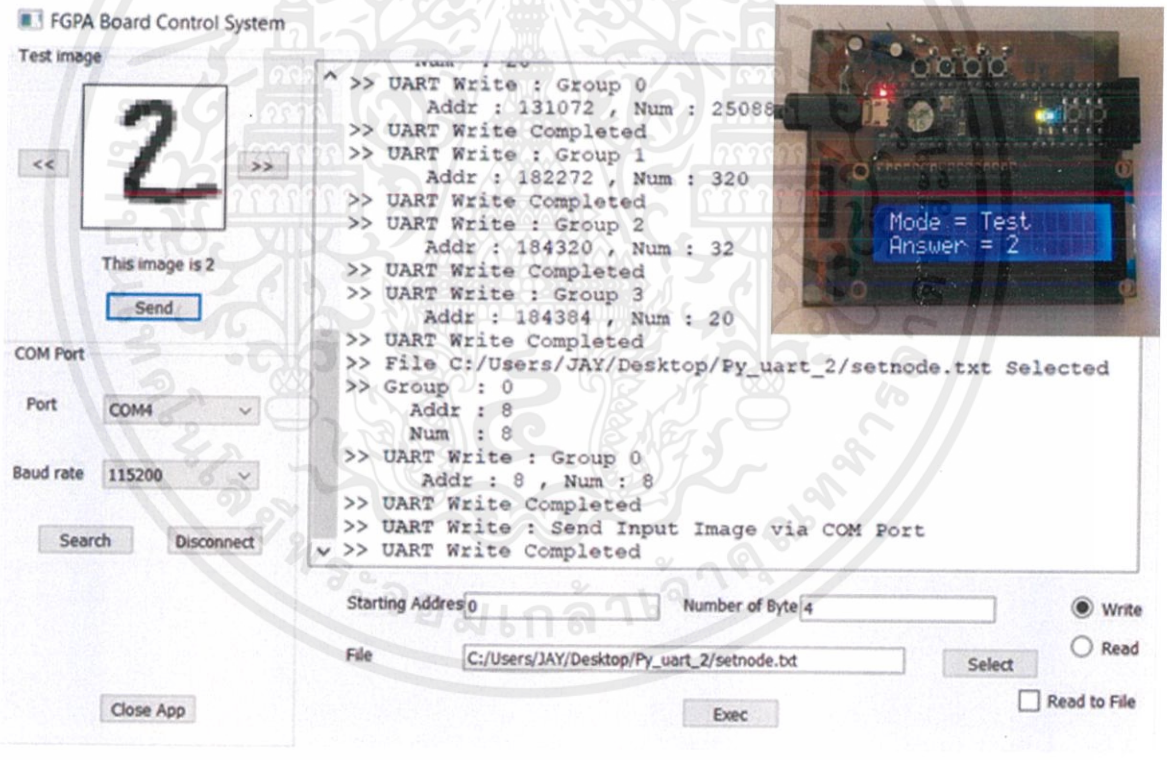
4.3 ปริมาณทรัพยากรที่ใช้โดยประมาณ

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	3039	0	20800	14.61
LUT as Logic	3039	0	20800	14.61
LUT as Memory	0	0	9600	0.00
Slice Registers	1686	0	41600	4.05
Register as Flip Flop	1686	0	41600	4.05
Register as Latch	0	0	41600	0.00
F7 Muxes	32	0	16300	0.20
F8 Muxes	0	0	8150	0.00

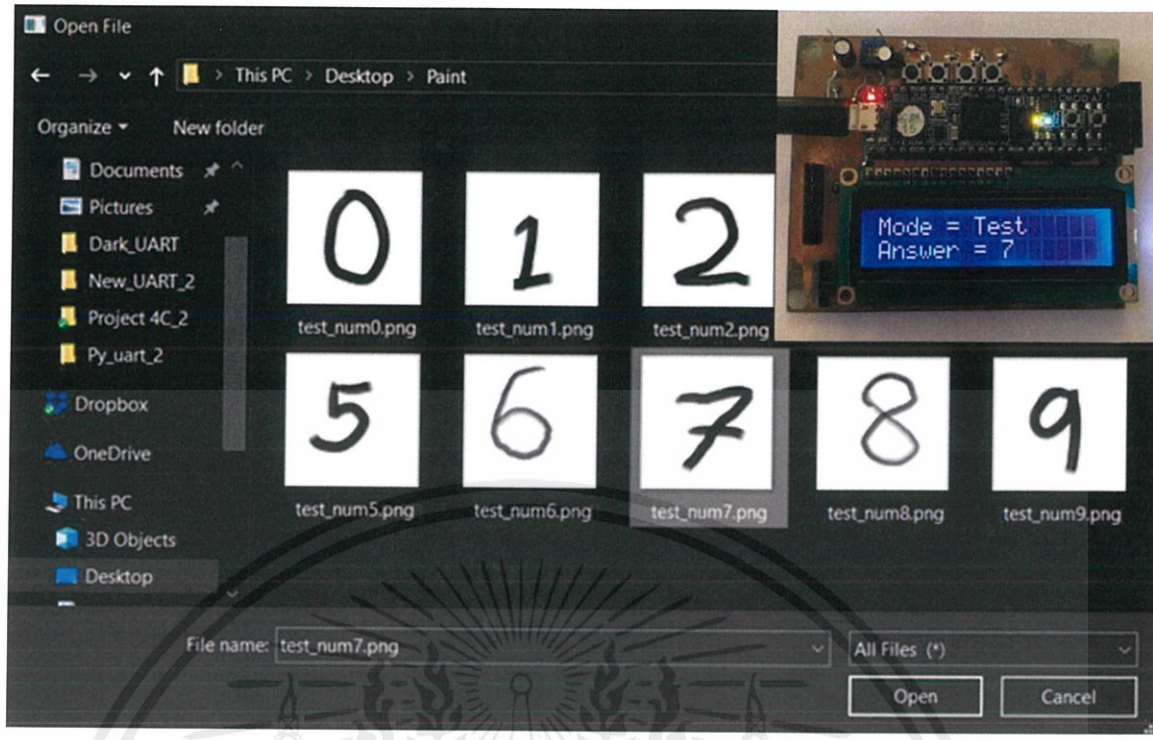


รูปที่ 30 ทรัพยากรที่ใช้งาน



รูปที่ 31 ทดสอบโดยใช้ข้อมูลจาก MNIST

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 32 ทดสอบโดยวาดรูปจากโปรแกรม Paint

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5

สรุปผลการทดลอง

สรุปผลการทดลอง

วงจรหน่วยประมวลผลโครงข่ายประสาทเทียมสามารถโปรแกรมโหนดต่างๆ ได้ดังนี้ อินพุต 800 โหนด ฮิตเดน 32 โหนด และเอาท์พุต 32 โหนด ระบบการประมวลผลขนาด 4 ชุด และเป็นแบบไปป์ไลน์-สตอล แต่อินพุตและฮิตเดนโหนดต้องเป็นจำนวนที่หารด้วย 4 ลงตัว สุดท้ายสามารถเพิ่มจำนวนโหนดได้โดยการเพิ่มขนาดของหน่วยความจำ

จากตารางผลการทดลองพบว่า วงจรทำงานเร็วกว่าโปรแกรมประมาณ 47 เท่าสำหรับโหมดการเรียนรู้ และเร็วกว่าประมาณ 36 เท่าสำหรับโหมดทดสอบ อีกหนึ่งสิ่งที่สังเกตได้คือยิ่งจำนวนฮิตเดนเลเยอร์มากขึ้นวงจรจะยิ่งมีความถูกต้องแม่นยำมากขึ้น

สุดท้ายความถี่ที่ใช้ในการทำงานของวงจรสามารถทำได้ถึงแค่ 76 MHz แต่ใช้จริงเพียง 72 MHz

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง

[1] Aurélien Géron, "Neural networks and deep learning," O'Reilly Media, Inc., March 2018

[2] Michael Nielsen, "Using neural nets to recognize handwritten digits", http://neuralnetworksanddeeplearning.com/chap1.html

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TOP Neural Network

```
module Top_NN(
i_CLK,
i_RSTn,
i_ADDR_READ,
i_ADDR_WRITE,
i_DATA_WRITE,
i_EN_WRITE,
o_done,
o_DATA_READ,
o_OUTPUT_NODE,
lcd_done,
lcd_mode
);
input          i_CLK;
input          i_RSTn;
input   [14:0] i_ADDR_READ;
input   [14:0] i_ADDR_WRITE;
input   [63:0] i_DATA_WRITE;
input          i_EN_WRITE;
output  [63:0] o_DATA_READ;
output         o_done;
output  [5:0]  o_OUTPUT_NODE;
output         lcd_done;
output         lcd_mode;
reg     [63:0] o_DATA_READ;
reg     [3:0]  r_cycle;
wire           w_load_RAM_Result;
wire    [12:0] w_Addr_RAM_input;
wire    [12:0] w_Addr_RAM_W_B;
wire           w_load_BIAS;
wire           w_start_pulse;
wire           w_en_SYS;
wire    [7:0]  w_n_replay;
wire    [12:0] w_Addr_RAM_Result;
wire    [63:0] w_Data_RAM_Result;
wire    [63:0] w_o_RAM_0;
wire    [63:0] w_o_RAM_1;
wire    [63:0] w_OP1;
wire    [63:0] w_OP2;
wire    [63:0] w_OP3;
wire    [127:0] w_ANS_STATE_0n1;
wire    [31:0] w_ANS_dot;
wire           w_load_data_0;
wire    [15:0] w_Sigmoid_Result;
wire    [15:0] w_Pre_Diff_Sigmoid;
wire    [15:0] w_Delta;
wire           w_load_data_1;
wire    [15:0] w_REG2_to_WRITE_BACK;
wire    [63:0] w_WB_Data_RAM1;
wire    [12:0] w_WB_Addr_RAM1;
wire           w_WB_Wen_RAM1;
wire           w_Activate_target;
wire           w_done;
wire    [9:0]  w_INPUTnode;
wire    [5:0]  w_HIDDENnode;
wire    [5:0]  w_OUTPUTnode;
wire    [7:0]  w_inputDIV4;
wire    [3:0]  w_hiddenDIV4;
wire    [3:0]  w_outputDIV4;
wire    [4:0]  w_numberHIDDEN;
wire    [4:0]  w_numberOUTPUT;
wire    [4:0]  w_dont_round_DIV4_output;
wire    [11:0] w_eta;
wire    [7:0]  w_cycle;
wire           w_EN_WRITE_RAM0;
wire           w_EN_WRITE_RAM1;
wire           w_EN_SETUPNODE;
wire           w_EN_START;
wire           w_EN_READ_RAM0;
wire           w_EN_READ_RAM1;
wire           w_EN_READ_RAM2;
wire    [1:0]  w_MODE;
wire    [1:0]  w_select_out;
wire    [3:0]  w_State;
wire           w_New_Cycle;
wire           w_NEXT_Layer;
wire           w_en_process;
wire    [4:0]  w_posi;
wire    [12:0] w_ADDR_TEMP_CA;
wire    [12:0] mux_addr_read_0;
wire    [12:0] mux_addr_write_0;
wire    [63:0] mux_data_write_0;
wire           mux_en_write_0;
wire    [12:0] mux_addr_read_1;
wire    [12:0] mux_addr_write_1;
wire    [63:0] mux_data_write_1;
wire           mux_en_write_1;
wire    [63:0] w_RAM2_DATA_IN2;
wire    [12:0] w_RAM2_ADDR_IN2;
wire           w_RAM2_EN_WRITE2;
wire    [63:0] w_RAM2_DATA_OUT;
wire    [12:0] mux_r_addr;
wire    [63:0] w_Result_cycle_2up;
wire           w_load2reg;
wire    [63:0] w_TEMP_DATA;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
wire          o_done;
wire          w_start_delay0;
wire   [5:0]  o_OUTPUT_NODE;
wire          lcd_done;
wire   [12:0] check_addr_read;
assign check_addr_read = i_ADDR_READ[12:0];
//_____________________ State 0
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_cycle <= 0;
else if (r_cycle==w_cycle&&w_done==1)
r_cycle <= 0;
else if (w_done)
r_cycle <= r_cycle + 1;
end
Decoder_NN Decoder_NN(
.i_CLK                  (i_CLK),
.i_RSTn                 (i_RSTn),
.i_EN_WRITE             (i_EN_WRITE),
.i_DATA_WRITE           (i_DATA_WRITE),
.i_ADDR_WRITE           (i_ADDR_WRITE),
.i_ADDR_READ            (i_ADDR_READ),
.i_DONE                 (w_done),
.o_EN_SETUPNODE         (w_EN_SETUPNODE),
.o_EN_START             (w_EN_START),
.o_EN_WRITE_RAM0        (w_EN_WRITE_RAM0),
.o_EN_WRITE_RAM1        (w_EN_WRITE_RAM1),
.o_EN_READ_RAM0         (w_EN_READ_RAM0),
.o_EN_READ_RAM1         (w_EN_READ_RAM1),
.o_EN_READ_RAM2         (w_EN_READ_RAM2),
.o_MODE                 (w_MODE),
.o_SELECT_OUT           (w_select_out),
.r_start_delay0         (w_start_delay0)
);
assign o_done = ~w_start_delay0;
Config_NN Config_NN(
.i_CLK                  (i_CLK),
.i_RSTn                 (i_RSTn),
.i_EN_SETUPNODE         (w_EN_SETUPNODE),
.i_DATA_WRITE           (i_DATA_WRITE),
.INPUTnode              (w_INPUTnode),      //784
.HIDDENnode             (w_HIDDENnode),     //16
.OUTPUTnode             (w_OUTPUTnode),     //10
.inputDIV4              (w_inputDIV4),      //195
.hiddenDIV4             (w_hiddenDIV4),     //3
.outputDIV4             (w_outputDIV4),     //2
.numberHIDDEN           (w_numberHIDDEN),   //15
.numberOUTPUT           (w_numberOUTPUT),   //9
.dont_round_DIV4_output (w_dont_round_DIV4_output),
//1
.eta                    (w_eta),
.cycle                  (w_cycle)
);
Counter_Addr Counter_Addr(
.i_CLK                  (i_CLK),
.i_RSTn                 (i_RSTn),
.i_NEXT_LAYER           (w_NEXT_Layer),
.i_EN_START             (w_EN_START),
.i_inputDIV4            (w_inputDIV4),
.i_hiddenDIV4           (w_hiddenDIV4),
.i_outputDIV4           (w_outputDIV4),
.i_numberHIDDEN         (w_numberHIDDEN),
.i_numberOUTPUT         (w_numberOUTPUT),
.i_dont_round_DIV4_output (w_dont_round_DIV4_output),
.i_numberCYCLE          (w_cycle),
.i_MODE                 (w_MODE),
.i_DONE                 (w_done),
.i_CYCLE                (r_cycle),
.o_ADDR_INPUT_RAM       (w_Addr_RAM_input),
.o_ADDR_WnB_RAM         (w_Addr_RAM_W_B),
.o_ADDR_TEMP            (w_ADDR_TEMP_CA),
.o_LOADBIAS             (w_load_BIAS),
.o_STARTPULSE           (w_start_pulse),
.o_EN_SYSTEM            (w_en_SYS),
.o_NUMBER_REPLAY        (w_n_replay),
.o_STATE                (w_State),
.o_NEW_STATE            (w_New_Cycle),
.o_ACTIVE_TARGET        (w_Activate_target),
.o_EN_PROCESS           (w_en_process),
.o_POSITION             (w_posi)
);
assign mux_en_write_0 =
(w_start_delay0==0&&w_EN_READ_RAM0==1) ? i_EN_WRITE :
w_load_RAM_Result;
assign mux_addr_write_0 =
(w_start_delay0==0&&w_EN_READ_RAM0==1) ?
i_ADDR_WRITE[12:0] : w_Addr_RAM_Result;
assign mux_data_write_0 =
(w_start_delay0==0&&w_EN_READ_RAM0==1) ? i_DATA_WRITE
: w_Data_RAM_Result;
assign mux_addr_read_0 =
(w_start_delay0==0&&w_EN_READ_RAM0==1) ?
i_ADDR_READ[12:0] : w_Addr_RAM_input;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
RAM_Result RAM_Result(
.clka                (i_CLK),
.ena                 (1'b1),
.wea                 (mux_en_write_0),
.addra               (mux_addr_write_0),          //test
.dina                (mux_data_write_0),
.clkb                (i_CLK),
.enb                 (1'b1),
.addrb               (mux_addr_read_0),
.doutb               (w_o_RAM_0)                  //test
);
assign mux_en_write_1 =
(w_start_delay0==0&&w_EN_READ_RAM1==1) ? i_EN_WRITE :
w_WB_Wen_RAM1;
assign mux_addr_write_1 =
(w_start_delay0==0&&w_EN_READ_RAM1==1) ?
i_ADDR_WRITE[12:0] : w_WB_Addr_RAM1;
assign mux_data_write_1 =
(w_start_delay0==0&&w_EN_READ_RAM1==1) ? i_DATA_WRITE
: w_WB_Data_RAM1;
assign mux_addr_read_1 =
(w_start_delay0==0&&w_EN_READ_RAM1==1) ?
i_ADDR_READ[12:0] : w_Addr_RAM_W_B;
RAM_W_B RAM_W_B(
.clka            (i_CLK),
.ena             (1'b1),
.wea             (mux_en_write_1),
.addra           (mux_addr_write_1),
.dina            (mux_data_write_1),
.clkb            (i_CLK),
.enb             (1'b1),
.addrb           (mux_addr_read_1),
.doutb           (w_o_RAM_1)
);
assign mux_r_addr =
(w_start_delay0==0&&w_EN_READ_RAM2==1) ?
i_ADDR_READ[12:0] : w_ADDR_TEMP_CA;
RAM_TEMP RAM_TEMP(
.clka            (i_CLK),
.ena             (1'b1),
.wea             (w_RAM2_EN_WRITE2),
.addra           (w_RAM2_ADDR_IN2),
.dina            (w_RAM2_DATA_IN2),
.clkb            (i_CLK),
.enb             (1'b1),
.addrb           (mux_r_addr),
.doutb           (w_RAM2_DATA_OUT)
);
always @(w_select_out or w_o_RAM_1 or w_o_RAM_0 or
w_RAM2_DATA_OUT)
begin
case (w_select_out)
0      : o_DATA_READ <= 0;
1      : o_DATA_READ <= w_o_RAM_0;
2      : o_DATA_READ <= w_o_RAM_1;
3      : o_DATA_READ <= w_RAM2_DATA_OUT;
default : o_DATA_READ <= 0;
endcase
end
reg_RAM reg_RAM(
.i_CLK           (i_CLK),
.i_RSTn          (i_RSTn),
.i_INPUTREG_0    (w_o_RAM_0),
.i_INPUTREG_1    (w_o_RAM_1),
.i_INPUTREG_2    (w_RAM2_DATA_OUT),
.o_OUTPUTREG_0   (w_OP1),
.o_OUTPUTREG_1   (w_OP2),
.o_OUTPUTREG_2   (w_OP3)
);
//_____________ State 1 (dot product
Calcu_0 Calcu_0(
.i_CLK           (i_CLK),
.i_RSTn          (i_RSTn),
.i_numberOUTPUT  (w_numberOUTPUT),
.i_OP1           (w_OP1),
.i_OP2           (w_OP2),
.i_OP3           (w_OP3),
.i_EN_PROCESS    (w_en_process),
.i_STATE         (w_State),
.i_POSITION      (w_posi),
.i_CYCLE         (r_cycle),
.i_ETA           (w_eta),
.o_STATE_01_OUTPUT  (w_ANS_STATE_0n1),
.o_STATE_2up_OUTPUT (w_Result_cycle_2up),
.o_LOAD2REG_STATE_5 (w_load2reg),
.o_TEMP_DATA     (w_TEMP_DATA)
);
//________ State 1 (Sumation of Dot product)__________
Calcu_1 Calcu_1(
.i_CLK           (i_CLK),
.i_RSTn          (i_RSTn),
.i_STATE         (w_State),
.i_EN_PROCESS    (w_en_SYS),
.i_CYCLE_NUMBER  (w_n_replay),
.i_INPUT         (w_ANS_STATE_0n1),
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
.o_OUTPUT          (w_ANS_dot),
.o_EN_NEXTREG      (w_load_data_0)
);
//___ State 2 (Sumation of Dot product with bias)_____________
Calcu_2 Calcu_2(
.i_CLK             (i_CLK),
.i_RSTn            (i_RSTn),
.i_BIAS            (w_OP2),
.i_TARGET          (w_OP1),
.i_LOADBIAS        (w_load_BIAS),
.i_INPUT           (w_ANS_dot),
.i_EN_REG          (w_load_data_0),
.i_STARTPULSE      (w_start_pulse),
.i_ACTIVE_TARGET   (w_Activate_target),
.i_STATE           (w_State),
.o_FF_ANSWER       (w_Sigmoid_Result),
.o_FF_PRESIG       (w_Pre_Diff_Sigmoid),
.o_FF_DELTA        (w_Delta),
.o_EN_NEXTREG      (w_load_data_1)
);
//_________ State 3 (Write back)_____________________
WRITE_BACK WRITE_BACK(
.i_CLK             (i_CLK),
.i_RSTn            (i_RSTn),
.i_EN_PROCESS      (w_en_process),
.i_STATE           (w_State),
.i_HIDDENnode      (w_HIDDENnode),     //16
.i_OUTPUTnode      (w_OUTPUTnode),     //10
.i_NEW_STATE       (w_New_Cycle),
.i_LOAD2REG        (w_load_data_1),
.i_LOAD2REG_STATE_5 (w_load2reg),
.i_FF_ANSWER       (w_Sigmoid_Result),
.i_FF_PRESIG       (w_Pre_Diff_Sigmoid),
.i_FF_DELTA        (w_Delta),
.i_BACKWARD        (w_Result_cycle_2up),
.i_TEMP_DATA       (w_TEMP_DATA),
.i_numberCYCLE     (w_cycle),
.i_CYCLE           (r_cycle),
.i_MODE            (w_MODE),
.o_OUTPUT_RAM0     (w_Data_RAM_Result),
.o_OUTPUT_RAM1     (w_WB_Data_RAM1),
.o_OUTPUT_RAM2     (w_RAM2_DATA_IN2),
.o_ADDRESS_RAM0    (w_Addr_RAM_Result),
.o_ADDRESS_RAM1    (w_WB_Addr_RAM1),
.o_ADDRESS_RAM2    (w_RAM2_ADDR_IN2),
.o_EN_WRITE_RAM0   (w_load_RAM_Result),
.o_EN_WRITE_RAM1   (w_WB_Wen_RAM1),
.o_EN_WRITE_RAM2   (w_RAM2_EN_WRITE2),
.o_NEXT_Layer      (w_NEXT_Layer),
.test_done         (w_done)
);
Comparator Comparator(
.i_CLK             (i_CLK),
.i_RSTn            (i_RSTn),
.i_STATE           (w_State),
.i_DONE            (w_done),
.i_LOAD2REG        (w_load_data_1),
.i_FF_ANSWER       (w_Sigmoid_Result),
.o_OUTPUT_NODE     (o_OUTPUT_NODE)
);
assign lcd_done = w_done;
assign lcd_mode = w_MODE;
endmodule
```

Decoder

```
module Decoder_NN(
i_CLK,
i_RSTn,
i_EN_WRITE,
i_DATA_WRITE,
i_ADDR_WRITE,
i_ADDR_READ,
i_DONE,
o_EN_SETUPNODE,
o_EN_START,
o_EN_WRITE_RAM0,
o_EN_WRITE_RAM1,
o_EN_READ_RAM0,
o_EN_READ_RAM1,
o_EN_READ_RAM2,
o_MODE,
o_SELECT_OUT,
r_start_delay0
);
input          i_CLK;
input          i_RSTn;
input          i_EN_WRITE;
input  [63:0]  i_DATA_WRITE;
input  [14:0]  i_ADDR_WRITE;
input  [14:0]  i_ADDR_READ;
input          i_DONE;
output         o_EN_SETUPNODE;
output         o_EN_START;
output         o_EN_WRITE_RAM0;
output         o_EN_WRITE_RAM1;
output         o_EN_READ_RAM0;
output         o_EN_READ_RAM1;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
output          o_EN_READ_RAM2;
output  [1:0]   o_MODE;
output  [1:0]   o_SELECT_OUT;
output          r_start_delay0;
wire            w_PRE_en_setupnode;
wire            w_PRE_en_start;
wire            o_EN_SETUPNODE;
wire            o_EN_START;
wire            o_EN_WRITE_RAM0;
wire            o_EN_WRITE_RAM1;
wire    [1:0]   o_MODE;
wire            o_EN_READ_RAM0;
wire            o_EN_READ_RAM1;
wire            o_EN_READ_RAM2;
wire    [1:0]   w_P_SELECT_OUT;
wire    [1:0]   o_SELECT_OUT;
reg             r_start_delay0;
reg             r_start_delay1;
reg             r_MODE_test;
reg             r_MODE_learn;
reg     [1:0]   r_delay_select_0;
//_____________________________ Setup Node & Start
assign  w_PRE_en_setupnode  = (
~r_start_delay0&&i_EN_WRITE&&(i_ADDR_WRITE[14:13]==2'b00
)&&(i_ADDR_WRITE[12:0]==1) );
assign  w_PRE_en_start      = (
~r_start_delay0&&i_EN_WRITE&&(i_ADDR_WRITE[14:13]==2'b00
)&&(i_ADDR_WRITE[12:0]==0) );
assign o_EN_SETUPNODE = w_PRE_en_setupnode;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_MODE_test <= 0;
else if (i_DONE)
r_MODE_test <= 0;
else if (w_PRE_en_start)
r_MODE_test <= ~i_DATA_WRITE[57];
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_MODE_learn <= 0;
else if (i_DONE)
r_MODE_learn <= 0;
else if (w_PRE_en_start)
r_MODE_learn <= i_DATA_WRITE[57];
end
assign o_MODE = {r_MODE_test,r_MODE_learn};
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_start_delay0  <= 0;
else if (i_DONE)
r_start_delay0 <= 0;
else if (w_PRE_en_start)
r_start_delay0  <= i_DATA_WRITE[56];
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_start_delay1  <= 0;
else
r_start_delay1  <= r_start_delay0;
end
assign o_EN_START = r_start_delay0&&(~r_start_delay1);
//_____________________________ Ram Selecter
assign o_EN_WRITE_RAM0 = i_ADDR_WRITE[14:13]==2'b01;
assign o_EN_WRITE_RAM1 = i_ADDR_WRITE[14:13]==2'b10;
//_____________________________________ Reader
assign o_EN_READ_RAM0 = i_ADDR_READ[14:13]==2'b01;
assign o_EN_READ_RAM1 = i_ADDR_READ[14:13]==2'b10;
assign o_EN_READ_RAM2 = i_ADDR_READ[14:13]==2'b11;
assign w_P_SELECT_OUT = i_ADDR_READ[14:13];
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_delay_select_0 <= 0;
else
r_delay_select_0 <= w_P_SELECT_OUT;
end
assign o_SELECT_OUT = r_delay_select_0;
endmodule
```

Config

```
module Config_NN(
i_CLK,
i_RSTn,
i_EN_SETUPNODE,
i_DATA_WRITE,
INPUTnode,
HIDDENnode,
OUTPUTnode,
inputDIV4,
hiddenDIV4,
outputDIV4,
numberHIDDEN,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
numberOUTPUT,
dont_round_DIV4_output,
eta,
cycle
);
input          i_CLK;
input          i_RSTn;
input          i_EN_SETUPNODE;
input   [63:0] i_DATA_WRITE;
output  [9:0]  INPUTnode;
output  [5:0]  HIDDENnode;
output  [5:0]  OUTPUTnode;
output  [7:0]  inputDIV4;
output  [3:0]  hiddenDIV4;
output  [3:0]  outputDIV4;
output  [4:0]  numberHIDDEN;
output  [4:0]  numberOUTPUT;
output  [4:0]  dont_round_DIV4_output;
output  [11:0] eta;
output  [7:0]  cycle;
wire    [9:0]  INPUTnode;
wire    [5:0]  HIDDENnode;
wire    [5:0]  OUTPUTnode;
wire    [7:0]  inputDIV4;
wire    [3:0]  hiddenDIV4;
wire    [3:0]  outputDIV4;
wire    [4:0]  numberHIDDEN;
wire    [4:0]  numberOUTPUT;
wire    [4:0]  dont_round_DIV4_output;
wire    [11:0] eta;
wire    [7:0]  cycle;
reg     [63:0] r_hold_config;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_hold_config <= 0;
else if (i_EN_SETUPNODE==1)
r_hold_config <= i_DATA_WRITE;
end
assign INPUTnode  = r_hold_config[15:0];
assign HIDDENnode = r_hold_config[23:16];
assign OUTPUTnode = r_hold_config[31:24];
assign inputDIV4  = ((r_hold_config[15:0]+3)>>2)-1;   //195
assign hiddenDIV4 = ((r_hold_config[23:16]+3)>>2)-1;  //3
assign outputDIV4 = ((r_hold_config[31:24]+3)>>2)-1;  //2
assign numberHIDDEN = r_hold_config[23:16]-1;         //15
assign numberOUTPUT = r_hold_config[31:24]-1;         //9
assign dont_round_DIV4_output = (r_hold_config[31:24]>>2)-1;
assign eta        = r_hold_config[63:40];
assign cycle      = r_hold_config[39:32]-1;
endmodule
```

Counter Address

```
`timescale 1ns / 1ps
`define LAYER0   800
`define LAYER1   32
`define LAYER2   32
`define LAYER0_W 0       // Layer 0 Weight Start Address
`define LAYER1_W `LAYER0_W+(`LAYER0*`LAYER1)/4  // Layer 1
Weight Start Address
`define LAYER1_B `LAYER1_W+(`LAYER1*`LAYER2)/4  // Layer 1
Bias Start Address
`define LAYER2_B `LAYER1_B+(`LAYER1/4)          // Layer 2 Bias
Start Address
module Counter_Addr(
i_CLK,
i_RSTn,
i_NEXT_LAYER,
i_EN_START,
i_inputDIV4,              //195
i_hiddenDIV4,             //3
i_outputDIV4,             //2
i_numberHIDDEN,           //15
i_numberOUTPUT,           //9
i_dont_round_DIV4_output, //1
i_numberCYCLE,
i_MODE,
i_DONE,
i_CYCLE,
o_ADDR_INPUT_RAM,
o_ADDR_WnB_RAM,
o_ADDR_TEMP,
o_LOADBIAS,
o_STARTPULSE,
o_EN_SYSTEM,
o_NUMBER_REPLAY,
o_STATE,
o_NEW_STATE,
o_ACTIVE_TARGET,
o_EN_PROCESS,
o_POSITION
);
input          i_CLK,i_RSTn;
input          i_NEXT_LAYER;
input          i_EN_START;
input   [7:0]  i_inputDIV4;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
input   [3:0]  i_hiddenDIV4;
input   [3:0]  i_outputDIV4;
input   [4:0]  i_numberHIDDEN;
input   [4:0]  i_numberOUTPUT;
input   [4:0]  i_dont_round_DIV4_output;
input   [7:0]  i_numberCYCLE;
input   [1:0]  i_MODE;
input          i_DONE;
input   [3:0]  i_CYCLE;
output  [12:0] o_ADDR_INPUT_RAM;
output  [12:0] o_ADDR_WnB_RAM;
output  [12:0] o_ADDR_TEMP;
output         o_LOADBIAS;
output         o_STARTPULSE;
output         o_EN_SYSTEM;
output  [7:0]  o_NUMBER_REPLAY;
output  [3:0]  o_STATE;
output         o_NEW_STATE;
output         o_ACTIVE_TARGET;
output         o_EN_PROCESS;
output  [4:0]  o_POSITION;
reg            en_sys;
reg            en_sys_delay;
reg            startpulse_delay0;
reg            startpulse_delay1;
reg            void_bias;
reg     [3:0]  state;
reg     [7:0]  cnt_0;
reg     [12:0] cnt_1;
reg     [12:0] cnt_2;
reg     [12:0] cnt_3;
reg     [12:0] cnt_4;
reg     [5:0]  cnt_5;
reg     [12:0] cnt_6;
reg     [12:0] mux_addr_input;
reg     [12:0] mux_addr_weight;
reg     [12:0] mux_addr_bias;
reg     [12:0] mux_addr_target;
reg            r_RSTn_delay;
reg            load_bias_delay0;
reg            load_bias_delay1;
reg     [7:0]  r_number_replay;
reg     [12:0] mux_addr_RAM2;
wire           b_stop_0;
wire           b_stop_1;
wire           b_stop_2;
wire           b_stop_3;
wire           b_stop_4;
wire           b_stop_5;
wire           b_stop_6;
wire           b_stop_7;
wire           b_stop_8;
wire           b_stop_9;
wire           b_stop_10;
wire           b_stop_11;
wire           b_stop_12;
wire           b_stop_13;
wire           dis_cnt;
wire           o_EN_PROCESS;
wire           b_startpulse;
wire           o_STARTPULSE;
wire           load_bias;
wire           instant_low;
wire           instant_high;
wire           o_EN_SYSTEM;
wire           re_state;
wire           en_cnt_0;
wire           en_cnt_2;
wire           re_cnt_0;
wire           re_cnt_1;
wire           re_cnt_2;
wire           re_cnt_3;
wire           re_cnt_4;
wire           re_cnt_5;
wire           re_cnt_6;
wire           add_cnt_0;
wire           add_cnt_1;
wire           add_cnt_2;
wire           add_cnt_3;
wire           add_cnt_4;
wire           add_cnt_5;
wire           add_cnt_6;
wire    [12:0] b_target_addr;
wire    [12:0] o_ADDR_INPUT_RAM;
wire    [12:0] b_weight_addr;
wire    [12:0] b_bias_addr;
wire    [12:0] o_ADDR_WnB_RAM;
wire           b_load_bias;
wire           o_LOADBIAS;
wire           o_LOAD_2_REG;
wire    [7:0]  o_NUMBER_REPLAY;
wire    [21:0] w_config_NN;
wire    [12:0] b_input_addr;
wire           load_Target;
wire    [3:0]  o_STATE;
wire           o_NEW_STATE;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
wire          o_ACTIVE_TARGET;
wire          en_addr_ram2;
wire   [12:0] o_ADDR_TEMP;
wire   [3:0]  jump_value;
wire   [4:0]  o_POSITION;
wire   [12:0] r_cnt_state5;
wire   [12:0] mux_state_5;
//_____________________ Stop Countering Signal
assign b_stop_0  = (state==0) && (cnt_0==i_inputDIV4)   &&
(cnt_5==i_numberHIDDEN);
assign b_stop_1  = (state==1) && (cnt_0==i_hiddenDIV4)  &&
(cnt_5==i_numberOUTPUT);
assign b_stop_2  = (state==2) && (cnt_0==i_hiddenDIV4);
assign b_stop_3  = (state==3) && (cnt_0==i_outputDIV4);
assign b_stop_4  = (state==4) && (cnt_0==i_outputDIV4);
assign b_stop_5  = (state==5) && (cnt_1==i_numberOUTPUT)
&& (cnt_4==i_hiddenDIV4);
assign b_stop_6  = (state==6) && (cnt_0==i_hiddenDIV4);
assign b_stop_7  = (state==7) && (cnt_0==i_inputDIV4)   &&
(cnt_5==i_numberHIDDEN);
assign b_stop_8  = (state==8) && (cnt_0==i_outputDIV4);
assign b_stop_9  = (state==9) && (cnt_0==i_hiddenDIV4)  &&
(cnt_5==i_numberOUTPUT);
assign b_stop_10 = (state==10) && (cnt_1==i_outputDIV4);
assign b_stop_11 = (state==11) && (cnt_1==i_hiddenDIV4);
assign b_stop_12 = (state==12) && (cnt_0==i_inputDIV4)   &&
(cnt_3==i_hiddenDIV4 && (cnt_5==i_numberHIDDEN));
assign b_stop_13 = (state==13) && (cnt_0==i_hiddenDIV4)  &&
(cnt_3==i_dont_round_DIV4_output) &&
(cnt_5==i_outputDIV4);
assign dis_cnt = b_stop_0 || b_stop_1 || b_stop_2  || b_stop_3
|| b_stop_4  || b_stop_5  || b_stop_6 || b_stop_7 ||
   b_stop_8 || b_stop_9 || b_stop_10 || b_stop_11 || b_stop_12
|| b_stop_13 || i_DONE;
//_____________________________________ Start Signal
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
en_sys <= 0;
else if (i_EN_START||i_NEXT_LAYER)
en_sys <= 1;
else if ( dis_cnt )
en_sys <= 0;
end
assign o_EN_PROCESS = en_sys;
//_________________ Signal for Bias Processing
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
en_sys_delay <= 0;
else
en_sys_delay <= en_sys;
end
assign b_startpulse = (en_sys)&&(~en_sys_delay);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
startpulse_delay0 <= 0;
else
startpulse_delay0 <= b_startpulse;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
startpulse_delay1 <= 0;
else
startpulse_delay1 <= startpulse_delay0;
end
assign o_STARTPULSE = startpulse_delay1;
//_____________________ Dont Put BIAS at this time
assign load_bias = ( (state==0) &&
((cnt_6==3&&cnt_0==i_inputDIV4)||(dis_cnt)) )||
( (state==1) && ((cnt_6==3&&cnt_0==i_hiddenDIV4)||(dis_cnt)) )
||(i_EN_START==1);
assign instant_low = (i_EN_START==1)||(i_DONE==1);
assign instant_high = (i_NEXT_LAYER)&&(state==1);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
void_bias <= 0;
else if( instant_low )
void_bias <= 0;
else if( instant_high )
void_bias <= 1;
else if (en_sys)
begin
if(load_bias)
void_bias <= 0;
else
void_bias <= 1;
end
end
assign o_EN_SYSTEM = void_bias;
//_____________________ Layer Management

assign re_state = (i_CYCLE!=i_numberCYCLE&&i_DONE)||
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
(i_CYCLE==i_numberCYCLE&&state==13&&i_DONE);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
state <= 0;
else if (re_state)
state <= 0;
else if ( (i_NEXT_LAYER) )
state <= state + 1;
end
//_____________________________ Input Counter
assign en_cnt_0 = (en_sys)&&(void_bias);
assign re_cnt_0 =( (state==0)  && (cnt_0==i_inputDIV4) )||
( (state==1)  && (cnt_0==i_hiddenDIV4) )||
( (state==2)  && (cnt_0==i_hiddenDIV4) )||
( (state==3)  && (cnt_0==i_outputDIV4) )||
( (state==4)  && (cnt_0==i_outputDIV4) )||
( (state==5)  && (cnt_1==i_numberOUTPUT) || dis_cnt )||
( (state==6)  && (cnt_0==i_hiddenDIV4) )||
( (state==7)  && (cnt_0==i_inputDIV4) )||
( (state==8)  && (cnt_0==i_outputDIV4) )||
( (state==9)  && (cnt_0==i_hiddenDIV4) )||
( (state==12) && (cnt_0==i_inputDIV4) )||
( (state==13) && (cnt_0==i_hiddenDIV4) );
assign add_cnt_0 =  (state==0)||(state==1)||(state==2)||
(state==3)||(state==4)||( state==6) ||(state==7)||( state==8)||(
state==9)||( state==12)||( state==13)||
( ((state==5)&&(cnt_3==3||(cnt_1==i_numberOUTPUT&&cnt_3
==i_dont_round_DIV4_output)) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_0 <= 0;
else if( en_cnt_0==1 )
begin
if( re_cnt_0 )
cnt_0 <= 0;
else if ( add_cnt_0 )
cnt_0 <= cnt_0 + 1;
end
end
//_____________________________
assign re_cnt_1 =  (state==5)&&(cnt_1==i_numberOUTPUT)
||(state==10)&&(dis_cnt)||state==11)&&(dis_cnt);
assign add_cnt_1 =  (state==5)||(state==10)||(state==11);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_1 <= 0;
else if (en_sys)
begin
if (re_cnt_1)
cnt_1 <= 0;
else if(add_cnt_1)
cnt_1 <= cnt_1 + 1;
end
else
cnt_1 <= 0;
end
always @(state)
begin
case (state)
4     : mux_addr_RAM2 <= 0;
6     : mux_addr_RAM2 <= 8;
7     : mux_addr_RAM2 <= 16;
9     : mux_addr_RAM2 <= 6416;
10    : mux_addr_RAM2 <= 0;
11    : mux_addr_RAM2 <= 8;
12    : mux_addr_RAM2 <= 16;
13    : mux_addr_RAM2 <= 6416;
default : mux_addr_RAM2 <= 0;
endcase
end
assign en_addr_ram2 = (state==4)||(state==6)||(state==7)||
(state==9)||(state==10)||(state==11)||(state==12)||(state==13);
assign o_ADDR_TEMP = (en_addr_ram2)? cnt_1 +
mux_addr_RAM2 : 0;
//_____________________________ Weight Counter
assign en_cnt_2 = en_cnt_0;
assign re_cnt_2 =   ( (state==0)  && (dis_cnt==1) )          ||
   ( (state==1)  && (dis_cnt==1) )          ||
   ( (state==2)  && (dis_cnt==1) )          ||
   ( (state==3)  && (dis_cnt==1) )          ||
   ( (state==4)  && (dis_cnt==1) )          ||
   ( (state==5)  && (cnt_1==i_numberOUTPUT) ) ||
   ( (state==6)  && (dis_cnt==1) )          ||
   ( (state==7)  && (dis_cnt==1) )          ||
      (state==8)                            ||
   ( (state==9)  && (dis_cnt==1) )          ||
   ( (state==10) && (dis_cnt==1) )          ||
   ( (state==11) && (dis_cnt==1) )          ||
   ( (state==12) && (dis_cnt==1) )          ||
   ( (state==13) && (dis_cnt==1) );
assign add_cnt_2 =  (state==0)                              ||
   (state==1)                               ||
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    (state==2)                                  ||
    (state==3)                                  ||
    (state==4)                                  ||
    (state==5)                                  ||
    (state==6)                                  ||
    ( (state==7) && ( cnt_0==i_inputDIV4  && cnt_3==3 ) ) ||
    ( (state==9) && ( cnt_0==i_hiddenDIV4 && cnt_3==3 ) ) ||
    (state==10)                                 ||
    (state==11)                                 ||
    (state==12)                                 ||
    (state==13);
assign jump_value = (state==5)? (i_hiddenDIV4+1) : 1;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_2 <= 0;
else if( en_cnt_2==1 )
begin
if( re_cnt_2==1 )
cnt_2 <= 0;
else if(add_cnt_2)
cnt_2 <= cnt_2 + jump_value;
end
end
//_____________________ Bias Counter && mini_state at 5 9 cycle
assign re_cnt_3 =   ( (state==0)  && (dis_cnt==1) )||
( (state==1)  && (dis_cnt==1) )||
( (state==5)  && (
(cnt_3==3)||(cnt_1==i_numberOUTPUT&&cnt_3==i_dont_roun
d_DIV4_output) ) )||
( (state==7)  && ( (dis_cnt==1) ||
(cnt_0==i_inputDIV4&&cnt_3==3) ) )||
( (state==9)  && ( (dis_cnt==1) ||
(cnt_0==i_hiddenDIV4&&cnt_3==3) ) )||
( (state==12) && (cnt_0==i_inputDIV4&&cnt_3==i_hiddenDIV4)
)||
( (state==13) && ( (dis_cnt==1) ||
(cnt_0==i_hiddenDIV4&&cnt_3==i_hiddenDIV4) ) );
assign add_cnt_3 =  ( (state==0)  && (void_bias==0) )||
( (state==1)  && (void_bias==0) )|| (state==5)||
( (state==7)  && (cnt_0==i_inputDIV4) )||
( (state==9)  && (cnt_0==i_hiddenDIV4) )||
( (state==12) && (cnt_0==i_inputDIV4) )||
( (state==13) && (cnt_0==i_hiddenDIV4) );

always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_3 <= 0;
else if(en_sys)
begin
if( re_cnt_3==1 )
cnt_3 <= 0;
else if(add_cnt_3==1)
cnt_3 <= cnt_3 + 1;
end
end
assign o_POSITION = cnt_3;
//_____________________________ Target Counter
assign re_cnt_4 =   ( (state==0) && (dis_cnt==1) )||
( (state==1) && (dis_cnt==1) )||
( (state==5) &&
(cnt_1==i_numberOUTPUT&&cnt_4==i_hiddenDIV4) );
assign add_cnt_4 =  ( (state==1) && (void_bias==0) )||
( (state==5) && (cnt_1==i_numberOUTPUT) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_4 <= 0;
else if(en_sys)
begin
if(re_cnt_4==1)
cnt_4 <= 0;
else if(add_cnt_4==1)
cnt_4 <= cnt_4 + 1;
end
end
assign r_cnt_state5 = cnt_2 + cnt_4;
//_____________________________ Cycle Counter
assign re_cnt_5 =   ( (state==0)  &&  dis_cnt==1 )||
( (state==1)  &&  dis_cnt==1 )||(state==2)||(state==3)||
(state==4)||( (state==5)  &&  dis_cnt==1 )||(state==6)||
( (state==7)  &&  dis_cnt==1 )||(state==8)||
( (state==9)  &&  dis_cnt==1 )||
( (state==12) && (dis_cnt==1 ||
(cnt_0==i_inputDIV4&&cnt_5==i_numberHIDDEN) ) )||
( (state==13) && (dis_cnt==1 ||
(cnt_0==i_hiddenDIV4&&cnt_5==i_hiddenDIV4) ) );
assign add_cnt_5 = ( (state==0)  && (cnt_0==i_inputDIV4) )||
( (state==1)&&(cnt_0==i_hiddenDIV4) )||
( (state==7)&&(cnt_0==i_inputDIV4) )||
( (state==9)&&(cnt_0==i_hiddenDIV4) )||
( (state==12)&&(cnt_0==i_inputDIV4) )||
( (state==13)&&cnt_0==i_hiddenDIV4&&cnt_3==i_hiddenDIV4));
always @(posedge i_CLK or negedge i_RSTn)
begin
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
if(~i_RSTn)
cnt_5 <= 0;
else if(re_cnt_5==1)
cnt_5 <= 0;
else if(add_cnt_5==1)
cnt_5 <= cnt_5 + 1;
end
//_____ Counter Number next node
assign re_cnt_6 =  (dis_cnt==1)||
( (state==0) && (cnt_0==i_inputDIV4&&cnt_6==3) )||
( (state==1) && (cnt_0==i_hiddenDIV4&&cnt_6==3) );
assign add_cnt_6 =  ( (state==0)&&(cnt_0==i_inputDIV4) )||
( (state==1)&&(cnt_0==i_hiddenDIV4) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
cnt_6 <= 0;
else if( re_cnt_6 )
cnt_6 <= 0;
else if( add_cnt_6 )
cnt_6 <= cnt_6 + 1;
end
//_____ Mux Address RAM 0
always @(state)
begin
case(state)
0     : mux_addr_input <= 0;
1     : mux_addr_input <= 208;
2     : mux_addr_input <= 208;
3     : mux_addr_input <= 216;
4     : mux_addr_input <= 224;
5     : mux_addr_input <= 240;
6     : mux_addr_input <= 232;
7     : mux_addr_input <= 0;
8     : mux_addr_input <= 240;
9     : mux_addr_input <= 208;
default : mux_addr_input <= 0;
endcase
end
always @(state)
begin
case(state)
0     : mux_addr_target <= 0;
1     : mux_addr_target <= 200;
2     : mux_addr_target <= 0;
3     : mux_addr_target <= 0;
default : mux_addr_target <= 0;
endcase
end
assign b_input_addr = (state==12||state==13)? 0 :
mux_addr_input + cnt_0;
assign b_target_addr = mux_addr_target + cnt_4;
assign load_Target = (state==1)&&(void_bias==0);
assign o_ADDR_INPUT_RAM = (load_Target==1)? b_target_addr
: b_input_addr;
//_____ Mux Address RAM 1
always @(state)
begin
case(state)
0     : mux_addr_weight <= `LAYER0_W;
1     : mux_addr_weight <= `LAYER1_W;
2     : mux_addr_weight <= 6672;
3     : mux_addr_weight <= 6680;
4     : mux_addr_weight <= 6688;
5     : mux_addr_weight <= 6400;
6     : mux_addr_weight <= 6696;
7     : mux_addr_weight <= 6704;
8     : mux_addr_weight <= 0;
9     : mux_addr_weight <= 6712;
10    : mux_addr_weight <= 6664;
11    : mux_addr_weight <= 6656;
12    : mux_addr_weight <= 0;
13    : mux_addr_weight <= 6400;
default : mux_addr_weight <= 0;
endcase
end
always @(state)
begin
case(state)
0     : mux_addr_bias <= `LAYER1_B;
1     : mux_addr_bias <= `LAYER2_B;
2     : mux_addr_bias <= 0;
3     : mux_addr_bias <= 0;
default : mux_addr_bias <= 0;
endcase
end
assign mux_state_5 = (state==5)? r_cnt_state5 : cnt_2;
assign b_weight_addr = mux_addr_weight + mux_state_5;
assign b_bias_addr = mux_addr_bias + cnt_3;
assign o_ADDR_WnB_RAM = (void_bias==1)? b_weight_addr :
b_bias_addr;
//_____ Load Bias Signal
assign b_load_bias = (~void_bias)&&(en_sys);
always @(posedge i_CLK or negedge i_RSTn)
begin
if (~i_RSTn)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
load_bias_delay0 <= 0;
else
load_bias_delay0 <= b_load_bias;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if (~i_RSTn)
load_bias_delay1 <= 0;
else
load_bias_delay1 <= load_bias_delay0;
end
assign o_LOADBIAS = load_bias_delay1;
//_____ en_REG Before into Sigmoid
always @(state or i_inputDIV4 or i_hiddenDIV4)
begin
case(state)
0       : r_number_replay <= i_inputDIV4;
1       : r_number_replay <= i_hiddenDIV4;
default : r_number_replay <= 0;
endcase
end
assign o_NUMBER_REPLAY = r_number_replay;
assign o_STATE = state;
assign o_NEW_STATE = b_startpulse;
assign o_ACTIVE_TARGET = (state == 1);
endmodule
```

Calculator0

```
module Calcu_0(
i_CLK,
i_RSTn,
i_numberOUTPUT,
i_OP1,
i_OP2,
i_OP3,
i_EN_PROCESS,
i_STATE,
i_POSITION,
i_CYCLE,
i_ETA,
o_STATE_01_OUTPUT,
o_STATE_2up_OUTPUT,
o_LOAD2REG_STATE_5,
o_TEMP_DATA
);
input                 i_CLK;
input                 i_RSTn;
input         [4:0]   i_numberOUTPUT;
input signed  [63:0]  i_OP1;
input signed  [63:0]  i_OP2;
input signed  [63:0]  i_OP3;
input                 i_EN_PROCESS;
input         [3:0]   i_STATE;
input         [4:0]   i_POSITION;
input         [3:0]   i_CYCLE;
input         [11:0]  i_ETA;
output signed [127:0] o_STATE_01_OUTPUT;
output signed [63:0]  o_STATE_2up_OUTPUT;
output                o_LOAD2REG_STATE_5;
output signed [63:0]  o_TEMP_DATA;
wire signed   [31:0]  w_ANS_multiply_0;
wire signed   [31:0]  w_ANS_multiply_1;
wire signed   [31:0]  w_ANS_multiply_2;
wire signed   [31:0]  w_ANS_multiply_3;
wire signed   [127:0] o_STATE_01_OUTPUT;
wire signed   [15:0]  w_ANS_rounded_0;
wire signed   [15:0]  w_ANS_rounded_1;
wire signed   [15:0]  w_ANS_rounded_2;
wire signed   [15:0]  w_ANS_rounded_3;
wire signed   [63:0]  w_ANS_state_2up;
wire          [127:0] w_INPUT_Stack;
wire          [63:0]  w_OUTPUT_stack;
wire                  w_load2reg_S5;
wire          [63:0]  o_TEMP_DATA;
wire signed   [15:0]  w_TEMP0;
wire signed   [15:0]  w_TEMP1;
wire signed   [15:0]  w_TEMP2;
wire signed   [15:0]  w_TEMP3;
wire          [63:0]  w_CAL_ETA;
wire          [15:0]  w_inv_0;
wire          [15:0]  w_inv_1;
wire          [15:0]  w_inv_2;
wire          [15:0]  w_inv_3;
wire          [63:0]  w_inv_DATA;
wire          [15:0]  w_POSITION_eta_0;
wire          [15:0]  w_POSITION_eta_1;
wire          [15:0]  w_POSITION_eta_2;
wire          [15:0]  w_POSITION_eta_3;
wire signed   [31:0]  w_P_rounded_0;
wire signed   [31:0]  w_P_rounded_1;
wire signed   [31:0]  w_P_rounded_2;
wire signed   [31:0]  w_P_rounded_3;
reg           [63:0]  o_STATE_2up_OUTPUT;
reg           [4:0]   r_POSITION_delay0;
reg           [4:0]   r_POSITION_delay1;
reg           [63:0]  mux_COPY_OP0;
reg           [63:0]  mux_COPY_OP1;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
reg signed      [63:0]  mux_REAL_OP0;
reg signed      [63:0]  mux_REAL_OP1;
reg             [12:0]  mux_roundup;
reg             [3:0]   mux_shifter;
reg             [63:0]  mux_X_OP0;
reg             [63:0]  mux_X_OP1;
//_____________________________ Position delay
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_POSITION_delay0 <= 0;
else
r_POSITION_delay0 <= i_POSITION;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_POSITION_delay1 <= 0;
else
r_POSITION_delay1 <= r_POSITION_delay0;
end
//_____________________________ MUX Operation
//_____________________________ Data Management
always @(r_POSITION_delay1 or i_OP1)
begin
case (r_POSITION_delay1)
0       : mux_COPY_OP0 <=
{i_OP1[63:48],i_OP1[63:48],i_OP1[63:48],i_OP1[63:48]};
1       : mux_COPY_OP0 <=
{i_OP1[47:32],i_OP1[47:32],i_OP1[47:32],i_OP1[47:32]};
2       : mux_COPY_OP0 <=
{i_OP1[31:16],i_OP1[31:16],i_OP1[31:16],i_OP1[31:16]};
3       : mux_COPY_OP0 <=
{i_OP1[15:0],i_OP1[15:0],i_OP1[15:0],i_OP1[15:0]};
default : mux_COPY_OP0 <= 0;
endcase
end
always @(r_POSITION_delay1 or i_OP2)
begin
case (r_POSITION_delay1)
0       : mux_COPY_OP1 <=
{i_OP2[63:48],i_OP2[63:48],i_OP2[63:48],i_OP2[63:48]};
1       : mux_COPY_OP1 <=
{i_OP2[47:32],i_OP2[47:32],i_OP2[47:32],i_OP2[47:32]};
2       : mux_COPY_OP1 <=
{i_OP2[31:16],i_OP2[31:16],i_OP2[31:16],i_OP2[31:16]};
3       : mux_COPY_OP1 <=
{i_OP2[15:0],i_OP2[15:0],i_OP2[15:0],i_OP2[15:0]};
default : mux_COPY_OP1 <= 0;
endcase
end
assign w_POSITION_eta_0 = i_ETA;
assign w_POSITION_eta_1 = i_ETA;
assign w_POSITION_eta_2 = i_ETA;
assign w_POSITION_eta_3 = i_ETA;
assign w_CAL_ETA = {w_POSITION_eta_0, w_POSITION_eta_1,
w_POSITION_eta_2, w_POSITION_eta_3};
//_____________________ Mux. before it going to multipier
always @(i_STATE or i_OP1 or mux_COPY_OP0 or w_CAL_ETA)
begin
case(i_STATE)
0       : mux_REAL_OP0 <= i_OP1;
1       : mux_REAL_OP0 <= i_OP1;
2       : mux_REAL_OP0 <= i_OP1;
3       : mux_REAL_OP0 <= i_OP1;
4       : mux_REAL_OP0 <= i_OP1;
5       : mux_REAL_OP0 <= mux_COPY_OP0;
6       : mux_REAL_OP0 <= i_OP1;
7       : mux_REAL_OP0 <= i_OP1;
8       : mux_REAL_OP0 <= i_OP1;
9       : mux_REAL_OP0 <= i_OP1;
10      : mux_REAL_OP0 <= w_CAL_ETA;
11      : mux_REAL_OP0 <= w_CAL_ETA;
12      : mux_REAL_OP0 <= w_CAL_ETA;
13      : mux_REAL_OP0 <= w_CAL_ETA;
default : mux_REAL_OP0 <= 0;
endcase
end
always @(i_STATE or i_OP2 or mux_COPY_OP1 or i_OP3)
begin
case(i_STATE)
0       : mux_REAL_OP1 <= i_OP2;
1       : mux_REAL_OP1 <= i_OP2;
2       : mux_REAL_OP1 <= i_OP2;
3       : mux_REAL_OP1 <= i_OP2;
4       : mux_REAL_OP1 <= i_OP2;
5       : mux_REAL_OP1 <= i_OP2;
6       : mux_REAL_OP1 <= i_OP2;
7       : mux_REAL_OP1 <= mux_COPY_OP1;
8       : mux_REAL_OP1 <= i_OP2;
9       : mux_REAL_OP1 <= mux_COPY_OP1;
10      : mux_REAL_OP1 <= i_OP3;
11      : mux_REAL_OP1 <= i_OP3;
12      : mux_REAL_OP1 <= i_OP3;
13      : mux_REAL_OP1 <= i_OP3;
default : mux_REAL_OP1 <= 0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
endcase
end
//_____ Multiplier
assign w_ANS_multiply_0 = $signed(mux_REAL_OP0[63:48]) *
$signed(mux_REAL_OP1[63:48]);
assign w_ANS_multiply_1 = $signed(mux_REAL_OP0[47:32]) *
$signed(mux_REAL_OP1[47:32]);
assign w_ANS_multiply_2 = $signed(mux_REAL_OP0[31:16]) *
$signed(mux_REAL_OP1[31:16]);
assign w_ANS_multiply_3 = $signed(mux_REAL_OP0[15:0])  *
$signed(mux_REAL_OP1[15:0]);
assign o_STATE_01_OUTPUT = {w_ANS_multiply_0,
w_ANS_multiply_1, w_ANS_multiply_2, w_ANS_multiply_3};
assign w_INPUT_Stack = o_STATE_01_OUTPUT;
//_____ Reducer 32 bit to 16 bit
always @(i_STATE)
begin
case(i_STATE)
0      : mux_roundup <= 0;
1      : mux_roundup <= 0;
2      : mux_roundup <= 2048;
3      : mux_roundup <= 2048;
4      : mux_roundup <= 2048;
5      : mux_roundup <= 2048;
6      : mux_roundup <= 2048;
7      : mux_roundup <= 2048;
8      : mux_roundup <= 2048;
9      : mux_roundup <= 2048;
10     : mux_roundup <= 4096;
11     : mux_roundup <= 4096;
12     : mux_roundup <= 4096;
13     : mux_roundup <= 4096;
default : mux_roundup <= 0;
endcase
end
always @(i_STATE)
begin
case(i_STATE)
0      : mux_shifter <= 0;
1      : mux_shifter <= 0;
2      : mux_shifter <= 12;
3      : mux_shifter <= 12;
4      : mux_shifter <= 12;
5      : mux_shifter <= 12;
6      : mux_shifter <= 12;
7      : mux_shifter <= 12;
8      : mux_shifter <= 12;
9      : mux_shifter <= 12;
10     : mux_shifter <= 13;
11     : mux_shifter <= 13;
12     : mux_shifter <= 13;
13     : mux_shifter <= 13;
default : mux_shifter <= 0;
endcase
end
// Rounder , Answer for statement 2 upper and Inverse data;
assign w_P_rounded_0 =  $signed(w_ANS_multiply_0) +
mux_roundup;
assign w_P_rounded_1 =  $signed(w_ANS_multiply_1) +
mux_roundup;
assign w_P_rounded_2 =  $signed(w_ANS_multiply_2) +
mux_roundup;
assign w_P_rounded_3 =  $signed(w_ANS_multiply_3) +
mux_roundup;
assign w_ANS_rounded_0 = w_P_rounded_0 >> mux_shifter;
assign w_ANS_rounded_1 = w_P_rounded_1 >> mux_shifter;
assign w_ANS_rounded_2 = w_P_rounded_2 >> mux_shifter;
assign w_ANS_rounded_3 = w_P_rounded_3 >> mux_shifter;
assign w_ANS_state_2up = {w_ANS_rounded_0,
w_ANS_rounded_1, w_ANS_rounded_2, w_ANS_rounded_3};
assign w_inv_0 = ~w_ANS_rounded_0+1;
assign w_inv_1 = ~w_ANS_rounded_1+1;
assign w_inv_2 = ~w_ANS_rounded_2+1;
assign w_inv_3 = ~w_ANS_rounded_3+1;
assign w_inv_DATA = {w_inv_0,w_inv_1,w_inv_2,w_inv_3};
//__ Stack_Summation ( Stage 5 ) _____
Stack_Summation Stack_Summation(
.i_CLK            (i_CLK),
.i_RSTn           (i_RSTn),
.i_STATE          (i_STATE),
.i_EN_PROCESS     (i_EN_PROCESS),
.i_numberOUTPUT   (i_numberOUTPUT),
.i_INPUT          (w_INPUT_Stack),
.o_OUTPUT         (w_OUTPUT_stack),
.o_LOAD2REG       (w_load2reg_S5)
);
assign o_LOAD2REG_STATE_5 = w_load2reg_S5;
//_____ END _____
always @(i_STATE or w_ANS_state_2up or w_OUTPUT_stack or
i_OP1)
begin
case(i_STATE)
0      : o_STATE_2up_OUTPUT <= 0;
1      : o_STATE_2up_OUTPUT <= 0;
2      : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
3      : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
4     : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
5     : o_STATE_2up_OUTPUT <= w_OUTPUT_stack;
6     : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
7     : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
8     : o_STATE_2up_OUTPUT <= i_OP1;
9     : o_STATE_2up_OUTPUT <= w_ANS_state_2up;
default : o_STATE_2up_OUTPUT <= 0;
endcase
end
//_____________________ TO TEMP RAM
always @(i_STATE or i_OP2 or w_ANS_state_2up)
begin
case(i_STATE)
0     : mux_X_OP0 <= w_ANS_state_2up;
1     : mux_X_OP0 <= w_ANS_state_2up;
2     : mux_X_OP0 <= w_ANS_state_2up;
3     : mux_X_OP0 <= w_ANS_state_2up;
4     : mux_X_OP0 <= w_ANS_state_2up;
5     : mux_X_OP0 <= w_ANS_state_2up;
6     : mux_X_OP0 <= w_ANS_state_2up;
7     : mux_X_OP0 <= w_ANS_state_2up;
8     : mux_X_OP0 <= w_ANS_state_2up;
9     : mux_X_OP0 <= w_ANS_state_2up;
10    : mux_X_OP0 <= i_OP2;
11    : mux_X_OP0 <= i_OP2;
12    : mux_X_OP0 <= i_OP2;
13    : mux_X_OP0 <= i_OP2;
default : mux_X_OP0 <= 0;
endcase
end
wire [3:0] x_STATE;
assign x_STATE = (i_CYCLE!=0)? i_STATE : 0;
always @(x_STATE or i_OP3 or w_inv_DATA)
begin
case(x_STATE)
0     : mux_X_OP1 <= 0;
1     : mux_X_OP1 <= 0;
2     : mux_X_OP1 <= 0;
3     : mux_X_OP1 <= 0;
4     : mux_X_OP1 <= i_OP3;
5     : mux_X_OP1 <= 0;
6     : mux_X_OP1 <= i_OP3;
7     : mux_X_OP1 <= i_OP3;
8     : mux_X_OP1 <= 0;
9     : mux_X_OP1 <= i_OP3;
10    : mux_X_OP1 <= w_inv_DATA;
11    : mux_X_OP1 <= w_inv_DATA;
12    : mux_X_OP1 <= w_inv_DATA;
13    : mux_X_OP1 <= w_inv_DATA;
default : mux_X_OP1 <= 0;
endcase
end
assign w_TEMP0 = $signed(mux_X_OP0[63:48]) +
$signed(mux_X_OP1[63:48]);
assign w_TEMP1 = $signed(mux_X_OP0[47:32]) +
$signed(mux_X_OP1[47:32]);
assign w_TEMP2 = $signed(mux_X_OP0[31:16]) +
$signed(mux_X_OP1[31:16]);
assign w_TEMP3 = $signed(mux_X_OP0[15:0]) +
$signed(mux_X_OP1[15:0]);
assign o_TEMP_DATA = {w_TEMP0, w_TEMP1, w_TEMP2,
w_TEMP3};
endmodule
```

Stacksummation

```
module Stack_Summation(
i_CLK,
i_RSTn,
i_EN_PROCESS,
i_numberOUTPUT,
i_INPUT,
i_STATE,
o_OUTPUT,
o_LOAD2REG
);
input             i_CLK;
input             i_RSTn;
input             i_EN_PROCESS;
input      [4:0]  i_numberOUTPUT;
input      [127:0] i_INPUT;
input      [3:0]  i_STATE;
output     [63:0] o_OUTPUT;
output            o_LOAD2REG;
reg               r_EN_PROCESS_delay0;
reg               r_EN_PROCESS_delay1;
reg        [4:0]  r_count_data;
reg signed [37:0] r_summation0;
reg signed [37:0] r_summation1;
reg signed [37:0] r_summation2;
reg signed [37:0] r_summation3;
wire              o_LOAD2REG;
wire signed [37:0] w_Sum_0;
wire signed [37:0] w_Sum_1;
wire signed [37:0] w_Sum_2;
wire signed [37:0] w_Sum_3;
wire signed [37:0] w_round_00;
wire signed [37:0] w_round_10;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
wire signed [37:0]  w_round_20;
wire signed [37:0]  w_round_30;
//_____________________________________ Enable Process Delay
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_EN_PROCESS_delay0 <= 0;
else
r_EN_PROCESS_delay0 <= i_EN_PROCESS;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_EN_PROCESS_delay1 <= 0;
else
r_EN_PROCESS_delay1 <= r_EN_PROCESS_delay0;
end
//_____________________________________ Data Counter
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_count_data <= 0;
else if (r_EN_PROCESS_delay1&&i_STATE==5)
begin
if(r_count_data==i_numberOUTPUT)
r_count_data <= 0;
else
r_count_data <= r_count_data + 1;
end
else
r_count_data <= 0;
end
assign o_LOAD2REG = (r_count_data==i_numberOUTPUT)? 1 :
0;
//_____________________________________ Summation Stack 0
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_summation0 <= 0;
else if (r_EN_PROCESS_delay1&&i_STATE==5)
begin
if(r_count_data==i_numberOUTPUT)
r_summation0 <= 0;
else
r_summation0 <= $signed(r_summation0) +
$signed(i_INPUT[127:96]);
end
else
r_summation0 <= 0;
end
//_____________________________________ Summation Stack 1
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_summation1 <= 0;
else if (r_EN_PROCESS_delay1&&i_STATE==5)
begin
if(r_count_data==i_numberOUTPUT)
r_summation1 <= 0;
else
r_summation1 <= $signed(r_summation1) +
$signed(i_INPUT[95:64]);
end
else
r_summation1 <= 0;
end
//_____________________________________ Summation Stack 2
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_summation2 <= 0;
else if (r_EN_PROCESS_delay1&&i_STATE==5)
begin
if(r_count_data==i_numberOUTPUT)
r_summation2 <= 0;
else
r_summation2 <= $signed(r_summation2) +
$signed(i_INPUT[63:32]);
end
else
r_summation2 <= 0;
end
//_____________________________________ Summation Stack 3
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_summation3 <= 0;
else if (r_EN_PROCESS_delay1&&i_STATE==5)
begin
if(r_count_data==i_numberOUTPUT)
r_summation3 <= 0;
else
r_summation3 <= $signed(r_summation3) +
$signed(i_INPUT[31:0]);
end
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
else
r_summation3 <= 0;
end
//-------------------------------- Rounder
assign w_Sum_0 = $signed(r_summation0) +
$signed(i_INPUT[127:96]);
assign w_Sum_1 = $signed(r_summation1) +
$signed(i_INPUT[95:64]);
assign w_Sum_2 = $signed(r_summation2) +
$signed(i_INPUT[63:32]);
assign w_Sum_3 = $signed(r_summation3) +
$signed(i_INPUT[31:0]);
assign w_round_00 = $signed(w_Sum_0)+1024;
assign w_round_10 = $signed(w_Sum_1)+1024;
assign w_round_20 = $signed(w_Sum_2)+1024;
assign w_round_30 = $signed(w_Sum_3)+1024;
assign o_OUTPUT =
{w_round_00[26:11],w_round_10[26:11],w_round_20[26:11],w_
round_30[26:11]};
endmodule
```

Calculator1

```
module Calcu_1(
i_CLK,
i_RSTn,
i_STATE,
i_EN_PROCESS,
i_CYCLE_NUMBER,
i_INPUT,
o_OUTPUT,
o_EN_NEXTREG
);
input                i_CLK;
input                i_RSTn;
input                i_EN_PROCESS;
input signed   [127:0] i_INPUT;
input          [7:0]   i_CYCLE_NUMBER;
input          [3:0]   i_STATE;
output         [31:0]  o_OUTPUT;
output                 o_EN_NEXTREG;
reg            [127:0] r_hold_data;
reg signed     [7:0]   r_count_INPUT_DIV_4;
reg signed     [42:0]  Pre_Sum_Hidden;
wire signed    [42:0]  w_Summation_dot
wire                   w_en_CALCU_1;
wire signed    [42:0]  Pre_Result_1;          // 43 bit
wire signed    [42:0]  round_0;
wire signed    [37:0]  round_1;
wire           [31:0]  o_OUTPUT;
wire                   o_EN_NEXTREG;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_hold_data <= 0;
else
r_hold_data <= i_INPUT;
end
DELAY_en_CALCU_1 DELAY_en_CALCU_1(
.i_CLK(i_CLK),
.i_RSTn(i_RSTn),
.i_DELAY_INPUT(i_EN_PROCESS),
.o_DELAY_OUTPUT(w_en_CALCU_1)
);
assign w_Summation_dot = $signed(r_hold_data[127:96]) +
$signed(r_hold_data[95:64]) + $signed(r_hold_data[63:32]) +
$signed(r_hold_data[31:0]);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_count_INPUT_DIV_4 <= 0;
else if (w_en_CALCU_1)
begin
if(r_count_INPUT_DIV_4 == i_CYCLE_NUMBER )
r_count_INPUT_DIV_4 <= 0;
else
r_count_INPUT_DIV_4 <= r_count_INPUT_DIV_4 + 1;
end
else
r_count_INPUT_DIV_4 <= 0;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
Pre_Sum_Hidden <= 0;
else if (w_en_CALCU_1)
begin
if(r_count_INPUT_DIV_4 == i_CYCLE_NUMBER )
Pre_Sum_Hidden <= 0;
else
Pre_Sum_Hidden <= $signed(Pre_Sum_Hidden) +
$signed(w_Summation_dot);
end
else
Pre_Sum_Hidden <= 0;
end
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
assign Pre_Result_1 = $signed(Pre_Sum_Hidden) +
$signed(w_Summation_dot);
assign round_0 = $signed(Pre_Result_1)+1024;
assign o_OUTPUT = round_0[42:11];
assign o_EN_NEXTREG = ((r_count_INPUT_DIV_4 ==
i_CYCLE_NUMBER)&&(i_STATE==0||i_STATE==1))? 1:0;
endmodule
```

Calculator2

```
module Calcu_2(
i_CLK,
i_RSTn,
i_BIAS,
i_TARGET,
i_LOADBIAS,
i_INPUT,
i_EN_REG,
i_STARTPULSE,
i_ACTIVE_TARGET,
i_STATE,
o_FF_ANSWER,
o_FF_PRESIG,
o_FF_DELTA,
o_EN_NEXTREG
);
input              i_CLK;
input              i_RSTn;
input              i_LOADBIAS;
input              i_EN_REG;
input              i_STARTPULSE;
input              i_ACTIVE_TARGET;
input       [63:0] i_BIAS;
input       [63:0] i_TARGET;
input       [31:0] i_INPUT;
input       [3:0]  i_STATE;
output      [15:0] o_FF_ANSWER;
output      [15:0] o_FF_PRESIG;
output      [15:0] o_FF_DELTA;
output             o_EN_NEXTREG;
reg         [31:0] r_hold_data;
reg         [79:0] r_Bias;
reg         [79:0] r_Target;
reg signed  [15:0] mux_INPUT_sigmoid;
wire               w_Shifted;
wire        [16:0] w_Shifted_Bias;
wire        [31:0] w_New_Bias;
wire signed [32:0] w_Bias_Plus_Weight;
wire               w_en_MAX;
wire               w_en_MIN;
wire        [1:0]  w_SELECT_input;
wire        [15:0] w_Sigmoid_Result;
wire        [15:0] o_FF_DELTA;
wire        [15:0] o_FF_ANSWER;
wire        [15:0] o_FF_PRESIG;
wire               o_EN_NEXTREG;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_hold_data <= 0;
else if (i_EN_REG)
r_hold_data <= i_INPUT;
else
r_hold_data <= 0;
end
DELAY_0 DELAY_0(
.i_CLK(i_CLK),
.i_RSTn(i_RSTn),
.i_DELAY_INPUT(i_EN_REG),
.o_DELAY_OUTPUT(w_Shifted)
);
//_____________________________ Bias & Shifter
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Bias <= 0;
else if ((i_LOADBIAS)&&(i_STARTPULSE))
r_Bias[79:16] <= i_BIAS;
else if (i_LOADBIAS)
r_Bias[63:0] <= i_BIAS;
else if (w_Shifted)
r_Bias <= {r_Bias[63:0],r_Bias[79:64]};
end
assign w_Shifted_Bias = {r_Bias[79:64],1'b0};    //17bit
assign w_New_Bias = (w_Shifted_Bias[16]==1)?
{15'b11111_11111_11111,w_Shifted_Bias}:{15'b00000_00000_0
0000,w_Shifted_Bias};
assign w_Bias_Plus_Weight = $signed(w_New_Bias) +
$signed(r_hold_data);
//_____________________________ INPUT Sigmoid
assign w_en_MAX = $signed(w_Bias_Plus_Weight>32767);
assign w_en_MIN = $signed(w_Bias_Plus_Weight<-32767);
assign w_SELECT_input = {w_en_MAX, w_en_MIN};
always @(w_SELECT_input or w_Bias_Plus_Weight)
begin
case (w_SELECT_input)
0 : mux_INPUT_sigmoid <= w_Bias_Plus_Weight[15:0];
1 : mux_INPUT_sigmoid <= -32767;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
2 : mux_INPUT_sigmoid <= 32767;
default : mux_INPUT_sigmoid <= 0;
endcase
end
//_____________________________ Sigmoid
Top_Sig Top_Sig(
.i_CLK          (i_CLK),
.i_RSTn         (i_RSTn),
.i_A            (mux_INPUT_sigmoid),
.o_A            (w_Sigmoid_Result)
);
//_____________________________ TARGET
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Target <= 0;
else if (i_ACTIVE_TARGET)
begin
if ((i_LOADBIAS)&&(i_STARTPULSE))
r_Target[79:16] <= i_TARGET;
else if ((i_LOADBIAS))
r_Target[63:0] <= i_TARGET;
else if ((w_Shifted))
r_Target <= {r_Target[63:0],r_Target[79:64]};
end
end
//_____________ Answer Sigmoid, delta and Presig
assign o_FF_DELTA = w_Sigmoid_Result - r_Target[79:64];
assign o_FF_ANSWER = w_Sigmoid_Result;
assign o_FF_PRESIG = 4096 - w_Sigmoid_Result;
assign o_EN_NEXTREG = (i_STATE==0||i_STATE==1)? w_Shifted
: 0;
endmodule
```

TopSigmoid

```
module Top_Sig(
i_CLK,
i_RSTn,
i_A,
o_A
);
input          i_CLK;
input          i_RSTn;
input   [15:0] i_A;
output  [15:0] o_A;
wire           w_Sign_Bit;
wire    [9:0]  w_Addr_Reg;
wire    [8:0]  w_Addr;
wire    [9:0]  two_Comp;
wire    [5:0]  w_Apx_Data;
wire    [16:0] w_O_ROM;
wire    [11:0] w_Data;
wire    [4:0]  w_deltaY;
wire    [15:0] Pos_delta;
wire    [15:0] Neg_delta;
wire    [15:0] Ans_Sigmoid;
wire    [15:0] o_A;
wire    [15:0] mux_input;
wire    [5:0]  w_Apx;
reg            Sign_delay_1;
reg     [5:0]  Apx_delay_1;
reg     [15:0] Ans_Sig;
assign mux_input = (i_A[15]==1)? {1'b1,((~i_A[14:0])+1)} :
{1'b0,i_A[14:0]};
assign w_Sign_Bit = i_A[15];
assign w_Addr = mux_input[14:6];
assign w_Apx = mux_input[5:0];
Sig_Memo Sig_Memo(
.addra(w_Addr),
.clka(i_CLK),
.ena(1'b1),
.douta(w_O_ROM)
);
assign w_Data = w_O_ROM[11:0];
assign w_deltaY = w_O_ROM[16:12];
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
Sign_delay_1 <= 0;
else
Sign_delay_1 <= w_Sign_Bit;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
Apx_delay_1 <= 0;
else
Apx_delay_1 <= w_Apx;
end
assign Pos_delta = (((w_deltaY*Apx_delay_1)+32)>>6)+w_Data;
assign Neg_delta = (4096-Pos_delta);
assign Ans_Sigmoid = (Sign_delay_1)? Neg_delta : Pos_delta;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
Ans_Sig <= 0;
else
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
Ans_Sig <= Ans_Sigmoid;
end
assign o_A = Ans_Sig;
endmodule
```

Writeback

```
module WRITE_BACK(
i_CLK,
i_RSTn,
i_EN_PROCESS,
i_STATE,
i_HIDDENnode,     //16
i_OUTPUTnode,     //10
i_NEW_STATE,
i_LOAD2REG,
i_LOAD2REG_STATE_5,
i_FF_ANSWER,
i_FF_PRESIG,
i_FF_DELTA,
i_BACKWARD,
i_TEMP_DATA,
i_numberCYCLE,
i_CYCLE,
o_OUTPUT_RAM0,
o_OUTPUT_RAM1,
o_OUTPUT_RAM2,
o_ADDRESS_RAM0,
o_ADDRESS_RAM1,
o_ADDRESS_RAM2,
o_EN_WRITE_RAM0,
o_EN_WRITE_RAM1,
o_EN_WRITE_RAM2,
o_NEXT_Layer,
i_MODE,
test_done
);
input          i_CLK;
input          i_RSTn;
input          i_EN_PROCESS;
input          i_LOAD2REG;
input          i_LOAD2REG_STATE_5;
input   [3:0]  i_STATE;
input          i_NEW_STATE;
input   [5:0]  i_HIDDENnode;
input   [5:0]  i_OUTPUTnode;
input   [15:0] i_FF_ANSWER;
input   [15:0] i_FF_PRESIG;
input   [15:0] i_FF_DELTA;
input   [63:0] i_BACKWARD;
input   [63:0] i_TEMP_DATA;
input   [1:0]  i_MODE;
input   [7:0]  i_numberCYCLE;
input   [3:0]  i_CYCLE;
output  [63:0] o_OUTPUT_RAM0;
output  [63:0] o_OUTPUT_RAM1;
output  [63:0] o_OUTPUT_RAM2;
output  [12:0] o_ADDRESS_RAM0;
output  [12:0] o_ADDRESS_RAM1;
output  [12:0] o_ADDRESS_RAM2;
output         o_EN_WRITE_RAM0;
output         o_EN_WRITE_RAM1;
output         o_EN_WRITE_RAM2;
output         o_NEXT_Layer;
output         test_done;
reg            r_EN_PROCESS_Delay_0;
reg            r_EN_PROCESS_Delay_1;
reg            r_EN_PROCESS_Delay_2;
reg            r_LOAD2REG_Delay_0;
reg    [14:0]  r_count_data;
reg    [2:0]   r_count_slot;
reg    [63:0]  r_Hold_Data_RAM_0;
reg    [63:0]  r_Reserve_Data;
reg    [12:0]  r_count_ADDRESS_0;
reg    [2:0]   r_delay_CNT_ADDR_0;
reg    [12:0]  mux_Addr_Ram_0;
reg            r_WRen_State_1;
reg    [63:0]  r_Hold_Data_RAM_1;
reg    [12:0]  mux_Addr_Ram_1;
reg    [12:0]  r_count_ADDRESS_1;
reg            r_delay_Load2Reg_5;
reg            o_EN_WRITE_RAM1;
reg            o_EN_WRITE_RAM0;
reg    [63:0]  o_OUTPUT_RAM1;
reg    [63:0]  r_Hold_Data_RAM_2;
reg    [63:0]  o_OUTPUT_RAM2;
reg    [12:0]  mux_counter;
reg    [12:0]  mux_Addr_Ram_2;
reg            test_done;
reg            o_NEXT_Layer;
wire   [63:0]  FF_OUTPUT;
wire   [63:0]  o_OUTPUT_RAM0;
wire           en_Address_Counter;
wire   [12:0]  Pre_Addr_RAM_1;
wire   [12:0]  Pre_Stack_Addr;
wire   [12:0]  o_ADDRESS_RAM0;
wire           w_ready_2_write;
wire           PRE_en_WRITE;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
wire   [63:0]  PRE_OUTPUT_RAM_1;
wire   [12:0]  o_ADDRESS_RAM1;
wire           w_en_data_counter;
wire           w_dis_data_counter;
wire           w_en_slot_counter;
wire           w_dis_slot_counter;
wire           w_en_HoldData_RAM0;
wire           w_Rapid_HoldData_RAM0;
wire           Pre_WRen;
wire           w_en_HoldData_RAM1;
wire           w_Rapid_HoldData_RAM1;
wire           en_Address_Counter_1;
wire           Pre_NEXT_0;
wire           Pre_NEXT_1;
wire           Pre_NEXT_2;
wire           Pre_NEXT_3;
wire           Pre_NEXT_4;
wire           Pre_NEXT_5;
wire           Pre_NEXT_6;
wire           Pre_NEXT_7;
wire           Pre_NEXT_8;
wire           Pre_NEXT_9;
wire           Pre_NEXT_10;
wire           Pre_NEXT_11;
wire           Pre_NEXT_12;
wire           Pre_NEXT_13;
wire           o_EN_WRITE_RAM2;
wire   [12:0]  o_ADDRESS_RAM2;
wire           w_LEARN_NEXT_Layer;
wire           w_UPDATE_NEXT_Layer;
wire           mux_NEXT_Layer;
wire           w_TEST_NEXT_Layer;
wire           mux_done;
//_____________________ Delay en_process 3 clk
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_EN_PROCESS_Delay_0 <= 0;
else
r_EN_PROCESS_Delay_0 <= i_EN_PROCESS;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_EN_PROCESS_Delay_1 <= 0;
else
r_EN_PROCESS_Delay_1 <= r_EN_PROCESS_Delay_0;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_EN_PROCESS_Delay_2 <= 0;
else
r_EN_PROCESS_Delay_2 <= r_EN_PROCESS_Delay_1;
end
//_____________________________________ Data Counter
assign w_en_data_counter = ( (i_STATE==0)&&(i_LOAD2REG) )
||
( (i_STATE==1)&&(i_LOAD2REG) );
assign w_dis_data_counter = (i_NEW_STATE==1)
||
( (i_STATE==0)&&(r_count_data==i_HIDDENnode) ) ||
( (i_STATE==1)&&(r_count_data==i_OUTPUTnode) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if (~i_RSTn)
r_count_data <= 0;
else if(w_dis_data_counter)
r_count_data <= 0;
else if(w_en_data_counter)
r_count_data <= r_count_data + 1;
end
//_____________________________________ Reg Slot Counter
assign w_en_slot_counter = i_LOAD2REG;
assign w_dis_slot_counter =
(i_NEW_STATE==1)||(r_count_slot==4);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_count_slot <= 0;
else if(w_dis_slot_counter)
r_count_slot <= 0;
else if(w_en_slot_counter)
r_count_slot <= r_count_slot + 1;
end
//_____________________________ Reg Slot & Output
assign w_ready_2_write = (r_count_slot==4)
||
( (i_STATE==0)&&(r_count_data==i_HIDDENnode) ) ||
( (i_STATE==1)&&(r_count_data==i_OUTPUTnode) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_WRen_State_1 <= 0;
else if (i_STATE==1)
r_WRen_State_1 <= w_ready_2_write;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
else
r_WRen_State_1 <=0;
end
assign PRE_en_WRITE = (w_ready_2_write)||(r_WRen_State_1);
//_____________________________________ Holder Reg 0
assign w_en_HoldData_RAM0 = (i_STATE==0)||(i_STATE==1);
assign w_Rapid_HoldData_RAM0 = (i_STATE==2)  ||
(i_STATE==4)  ||
(i_STATE==7)  ||
(i_STATE==9);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Hold_Data_RAM_0 <= 0;
else if (w_en_HoldData_RAM0)
begin
if (o_EN_WRITE_RAM0)
r_Hold_Data_RAM_0 <= 0;
else if ( (i_LOAD2REG)&&(r_count_slot==0) )
r_Hold_Data_RAM_0[63:48] <= i_FF_ANSWER;
else if ( (i_LOAD2REG)&&(r_count_slot==1) )
r_Hold_Data_RAM_0[47:32] <= i_FF_ANSWER;
else if ( (i_LOAD2REG)&&(r_count_slot==2) )
r_Hold_Data_RAM_0[31:16] <= i_FF_ANSWER;
else if ( (i_LOAD2REG)&&(r_count_slot==3) )
r_Hold_Data_RAM_0[15:0] <= i_FF_ANSWER;
end
else if (w_Rapid_HoldData_RAM0)
r_Hold_Data_RAM_0 <= i_BACKWARD;
end
//_____________________________ Reserve Reg slot
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Reserve_Data <= 0;
else if (r_WRen_State_1)
r_Reserve_Data <= 0;
else if ( (i_LOAD2REG)&&(r_count_slot==0) )
r_Reserve_Data[63:48] <= i_FF_DELTA;
else if ( (i_LOAD2REG)&&(r_count_slot==1) )
r_Reserve_Data[47:32] <= i_FF_DELTA;
else if ( (i_LOAD2REG)&&(r_count_slot==2) )
r_Reserve_Data[31:16] <= i_FF_DELTA;
else if ( (i_LOAD2REG)&&(r_count_slot==3) )
r_Reserve_Data[15:0] <= i_FF_DELTA;
end
assign o_OUTPUT_RAM0 = (r_WRen_State_1==1)?
r_Reserve_Data : r_Hold_Data_RAM_0
//_____________________________ Address Ram0
assign en_Address_Counter = (r_count_slot==4)          ||
( (r_EN_PROCESS_Delay_2==1) &&
((i_STATE==2)          ||
(i_STATE==4)          ||
(i_STATE==7)          ||
(i_STATE==9)) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_count_ADDRESS_0 <= 0;
else if (o_NEXT_Layer||test_done)
r_count_ADDRESS_0 <= 0;
else if(en_Address_Counter)
r_count_ADDRESS_0 <= r_count_ADDRESS_0 + 1;
end
always @(i_STATE)
begin
case (i_STATE)
0     : mux_Addr_Ram_0 <= 208;
1     : mux_Addr_Ram_0 <= 216;
2     : mux_Addr_Ram_0 <= 232;
3     : mux_Addr_Ram_0 <= 0;
4     : mux_Addr_Ram_0 <= 240;
5     : mux_Addr_Ram_0 <= 0;
6     : mux_Addr_Ram_0 <= 0;
7     : mux_Addr_Ram_0 <= 504;
8     : mux_Addr_Ram_0 <= 6904;
9     : mux_Addr_Ram_0 <= 248;
default : mux_Addr_Ram_0 <= 0;
endcase
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_delay_CNT_ADDR_0 <= 0;
else
r_delay_CNT_ADDR_0 <= r_count_ADDRESS_0;
end
assign Pre_Addr_RAM_1 = r_count_ADDRESS_0 +
mux_Addr_Ram_0;
assign Pre_Stack_Addr = r_delay_CNT_ADDR_0 + 224;
//addr 224
assign o_ADDRESS_RAM0 = (r_WRen_State_1==1)?
Pre_Stack_Addr : Pre_Addr_RAM_1;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
//_____________________________ Write Enable Ram0
always @(i_STATE or PRE_en_WRITE or
r_EN_PROCESS_Delay_2)
begin
case (i_STATE)
0       : o_EN_WRITE_RAM0 <= PRE_en_WRITE;
1       : o_EN_WRITE_RAM0 <= PRE_en_WRITE;
2       : o_EN_WRITE_RAM0 <= r_EN_PROCESS_Delay_2;
3       : o_EN_WRITE_RAM0 <= 0;
4       : o_EN_WRITE_RAM0 <= r_EN_PROCESS_Delay_2;
5       : o_EN_WRITE_RAM0 <= 0;
6       : o_EN_WRITE_RAM0 <= 0;
7       : o_EN_WRITE_RAM0 <= 0;  //r_EN_PROCESS_Delay_2
8       : o_EN_WRITE_RAM0 <= 0;
9       : o_EN_WRITE_RAM0 <= 0;  //r_EN_PROCESS_Delay_2
default : o_EN_WRITE_RAM0 <= 0;
endcase
end
//_____________________________ Writer 2
assign w_en_HoldData_RAM1 = w_en_HoldData_RAM0;
assign w_Rapid_HoldData_RAM1 = (i_STATE==3) ||
(i_STATE==5) ||
(i_STATE==6) ||
(i_STATE==8);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Hold_Data_RAM_1 <= 0;
else if (w_en_HoldData_RAM1)
begin
if (o_EN_WRITE_RAM0)
r_Hold_Data_RAM_1 <= 0;
else if ( (i_LOAD2REG)&&(r_count_slot==0) )
r_Hold_Data_RAM_1[63:48] <= i_FF_PRESIG;
else if ( (i_LOAD2REG)&&(r_count_slot==1) )
r_Hold_Data_RAM_1[47:32] <= i_FF_PRESIG;
else if ( (i_LOAD2REG)&&(r_count_slot==2) )
r_Hold_Data_RAM_1[31:16] <= i_FF_PRESIG;
else if ( (i_LOAD2REG)&&(r_count_slot==3) )
r_Hold_Data_RAM_1[15:0] <= i_FF_PRESIG;
end
else if (w_Rapid_HoldData_RAM1)
r_Hold_Data_RAM_1 <= i_BACKWARD;
end
always @(i_STATE)
begin
case (i_STATE)
0       : mux_Addr_Ram_1 <= 6672;
1       : mux_Addr_Ram_1 <= 6680;
2       : mux_Addr_Ram_1 <= 6688;
3       : mux_Addr_Ram_1 <= 6688;
4       : mux_Addr_Ram_1 <= 0;
5       : mux_Addr_Ram_1 <= 6696;
6       : mux_Addr_Ram_1 <= 6704;
7       : mux_Addr_Ram_1 <= 0;
8       : mux_Addr_Ram_1 <= 6712;
9       : mux_Addr_Ram_1 <= 0;
10      : mux_Addr_Ram_1 <= 6664;
11      : mux_Addr_Ram_1 <= 6656;
12      : mux_Addr_Ram_1 <= 0;
13      : mux_Addr_Ram_1 <= 6400;
default : mux_Addr_Ram_1 <= 0;
endcase
end
assign en_Address_Counter_1 = (r_count_slot==4)
||
( (r_EN_PROCESS_Delay_2)            &&
( (i_STATE==3)                  ||
(i_STATE==6)                    ||
(i_STATE==8)                    ||
(i_STATE==10)                   ||
(i_STATE==11)                   ||
(i_STATE==12)                   ||
(i_STATE==13) ) )               ||
( (i_STATE==5)&&(r_delay_Load2Reg_5) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_delay_Load2Reg_5 <= 0;
else
r_delay_Load2Reg_5 <= i_LOAD2REG_STATE_5;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_count_ADDRESS_1 <= 0;
else if(o_NEXT_Layer)
r_count_ADDRESS_1 <= 0;
else if(en_Address_Counter_1)
r_count_ADDRESS_1 <= r_count_ADDRESS_1 + 1;
end
assign o_ADDRESS_RAM1 = r_count_ADDRESS_1 +
mux_Addr_Ram_1;
always @(i_STATE or r_Hold_Data_RAM_1 or
r_Hold_Data_RAM_2)
begin
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
case(i_STATE)
0      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
1      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
2      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
3      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
4      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
5      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
6      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
7      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
8      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
9      : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_1;
10     : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_2;
11     : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_2;
12     : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_2;
13     : o_OUTPUT_RAM1 <= r_Hold_Data_RAM_2;
default : o_OUTPUT_RAM1 <= 0;
endcase
end
always @(i_STATE or r_EN_PROCESS_Delay_2 or
w_ready_2_write or r_delay_Load2Reg_5)
begin
case(i_STATE)
0      : o_EN_WRITE_RAM1 <= w_ready_2_write;
1      : o_EN_WRITE_RAM1 <= w_ready_2_write;
2      : o_EN_WRITE_RAM1 <= w_ready_2_write;
3      : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
4      : o_EN_WRITE_RAM1 <= 0;
5      : o_EN_WRITE_RAM1 <= r_delay_Load2Reg_5;
6      : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
7      : o_EN_WRITE_RAM1 <= 0;
8      : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
9      : o_EN_WRITE_RAM1 <= 0;
10     : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
11     : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
12     : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
13     : o_EN_WRITE_RAM1 <= r_EN_PROCESS_Delay_2;
default : o_EN_WRITE_RAM1 <= 0;
endcase
end
//_____________________ Copy Writer
wire w_Rapid_HoldData_RAM2;
assign w_Rapid_HoldData_RAM2 = (i_STATE==10)||
(i_STATE==11)||
(i_STATE==12)||
(i_STATE==13);
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_Hold_Data_RAM_2 <= 0;
else if
(w_Rapid_HoldData_RAM0||w_Rapid_HoldData_RAM1||w_Rapid
_HoldData_RAM2)
r_Hold_Data_RAM_2 <= i_TEMP_DATA;
end
always @(i_STATE or r_Hold_Data_RAM_2)
begin
case(i_STATE)
4      : o_OUTPUT_RAM2 <= r_Hold_Data_RAM_2;
6      : o_OUTPUT_RAM2 <= r_Hold_Data_RAM_2;
7      : o_OUTPUT_RAM2 <= r_Hold_Data_RAM_2;
9      : o_OUTPUT_RAM2 <= r_Hold_Data_RAM_2;
default : o_OUTPUT_RAM2 <= 0;
endcase
end
assign o_EN_WRITE_RAM2 =
(i_STATE==4||i_STATE==6||i_STATE==7||i_STATE==9)?
r_EN_PROCESS_Delay_2 : 0;
always @(i_STATE or r_count_ADDRESS_0 or
r_count_ADDRESS_1)
begin
case(i_STATE)
4      : mux_counter <= r_count_ADDRESS_0;
6      : mux_counter <= r_count_ADDRESS_1;
7      : mux_counter <= r_count_ADDRESS_0;
9      : mux_counter <= r_count_ADDRESS_0;
default : mux_counter <= 0;
endcase
end
always @(i_STATE)
begin
case (i_STATE)
4      : mux_Addr_Ram_2 <= 0;
6      : mux_Addr_Ram_2 <= 8;
7      : mux_Addr_Ram_2 <= 16;
9      : mux_Addr_Ram_2 <= 6416;
default : mux_Addr_Ram_2 <= 0;
endcase
end
assign o_ADDRESS_RAM2 = mux_Addr_Ram_2 + mux_counter;
assign Pre_NEXT_0 =
(i_STATE==0)&&(r_count_data==i_HIDDENnode)&&(o_EN_WRIT
E_RAM0);
assign Pre_NEXT_1 =
(i_STATE==1)&&(r_count_data==0)&&(r_WRen_State_1);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
assign Pre_NEXT_2 =
(i_STATE==2)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_3 =
(i_STATE==3)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_4 =
(i_STATE==4)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_5 =
(i_STATE==5)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_6 =
(i_STATE==6)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_7 =
(i_STATE==7)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_8 =
(i_STATE==8)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_9 =
(i_STATE==9)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCESS
_Delay_2==1);
assign Pre_NEXT_10 =
(i_STATE==10)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCES
S_Delay_2==1);
assign Pre_NEXT_11 =
(i_STATE==11)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCES
S_Delay_2==1);
assign Pre_NEXT_12 =
(i_STATE==12)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCES
S_Delay_2==1);
assign Pre_NEXT_13 =
(i_STATE==13)&&(r_EN_PROCESS_Delay_1==0)&&(r_EN_PROCES
S_Delay_2==1);
assign w_LEARN_NEXT_Layer =
Pre_NEXT_0||Pre_NEXT_1||Pre_NEXT_2||Pre_NEXT_3||Pre_NEXT
_4||Pre_NEXT_5||Pre_NEXT_6||Pre_NEXT_7||Pre_NEXT_8;
assign w_UPDATE_NEXT_Layer =
Pre_NEXT_0||Pre_NEXT_1||Pre_NEXT_2||
Pre_NEXT_3||Pre_NEXT_4||Pre_NEXT_5||
Pre_NEXT_6||Pre_NEXT_7||Pre_NEXT_8||
Pre_NEXT_9||Pre_NEXT_10||Pre_NEXT_11||
Pre_NEXT_12;
assign mux_NEXT_Layer = (i_CYCLE==i_numberCYCLE)?
w_UPDATE_NEXT_Layer : w_LEARN_NEXT_Layer;
assign w_TEST_NEXT_Layer = Pre_NEXT_0;
always @(i_MODE or mux_NEXT_Layer or
w_TEST_NEXT_Layer)
begin
case (i_MODE)
1 : o_NEXT_Layer <= mux_NEXT_Layer;
2 : o_NEXT_Layer <= w_TEST_NEXT_Layer;
default : o_NEXT_Layer <= 0;
endcase
end
assign mux_done = (i_CYCLE==i_numberCYCLE)? Pre_NEXT_13
: Pre_NEXT_9;
always @(i_MODE or mux_done or Pre_NEXT_1)
begin
case (i_MODE)
1 : test_done <= mux_done;
2 : test_done <= Pre_NEXT_1;
default : test_done <= 0;
endcase
end
endmodule
```

Comparator

```
module Comparator(
i_CLK,
i_RSTn,
i_STATE,
i_DONE,
i_LOAD2REG,
i_FF_ANSWER,
o_OUTPUT_NODE
);
input          i_CLK;
input          i_RSTn;
input   [3:0]  i_STATE;
input          i_DONE;
input          i_LOAD2REG;
input   [15:0] i_FF_ANSWER;
output  [5:0]  o_OUTPUT_NODE;
reg     [31:0] r_hold_data;
reg     [5:0]  r_position;
reg            r_delay_load_0;
reg     [5:0]  r_ANS_NODE;
reg     [5:0]  r_ANS;
wire w_more_compare;
wire w_start_pulse;
wire    [5:0]  o_OUTPUT_NODE;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
assign w_more_compare = (r_hold_data[31:16] <
r_hold_data[15:0])? 1 : 0;
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_delay_load_0 <= 0;
else if (i_DONE)
r_delay_load_0 <= 0;
else if (i_STATE==1&&i_LOAD2REG)
r_delay_load_0 <= i_LOAD2REG;
end
assign w_start_pulse = ( (i_LOAD2REG)&&(~r_delay_load_0) );
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_hold_data <= 0;
else if(i_STATE==1)
begin
if (w_start_pulse)
r_hold_data[31:16]  <= i_FF_ANSWER;
else if (i_LOAD2REG)
r_hold_data[15:0]   <= i_FF_ANSWER;
else if (w_more_compare)
r_hold_data[31:16]  <= r_hold_data[15:0];
end
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_ANS_NODE <= 0;
else if (w_start_pulse)
r_ANS_NODE <= 0;
else if (i_STATE==1&&i_LOAD2REG)
r_ANS_NODE <= r_ANS_NODE + 1;
end
always @(posedge i_CLK or negedge i_RSTn)
begin
if(~i_RSTn)
r_ANS <= 0;
else if (w_start_pulse)
r_ANS <= 0;
else if (i_STATE==1&&w_more_compare)
r_ANS <= r_ANS_NODE;
end
assign o_OUTPUT_NODE = r_ANS;
endmodule
```

เอกสารนี้เป็นเอกสารที่สงวนไว้ สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้