โครงงานการพัฒนาโปรแกรมบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐาน H.264 บน Blackfin DSP

Development of H.264 Video Compression Standard on Blackfin DSP Board

โดย

นายอนุชา หิรัญพานิช

นายอมรเทพ เผือกพิบูลย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

โครงงานการพัฒนาโปรแกรมบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐาน H.264 บน Blackfin DSP

Development of H.264 Video Compression Standard on Blackfin DSP Board

โดย

นายอนุชา หิรัญพานิช 46010912

นายอมรเทพ เผือกพิบูลย์ 46010925

อาจารย์ที่ปรึกษา

ผศ.ดร.อรฉัตร จิตต์โสภักดร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **โครงงานการพัฒนาโปรแกรมบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐานH.264 บน Blackfin DSP**

Development of Video Compression in H.264 Standard on Blackfin DSP

ผู้จัดทำ

1. **นายอนุชา หิรัญพานิช** **46010912**
2. **นายอมรเทพ เผือกพิบูลย์** **46010925**

อรฉัตร จิตต์โสภักตร์ อาจารย์ที่ปรึกษา

(ผศ.ดร.อรฉัตร จิตต์โสภักตร์)

หัวข้อ โครงงานการพัฒนาโปรแกรมบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐานH.264 บน Blackfin DSP

จัดทำโดย นายอนุชา หิรัญพานิช รหัสประจำตัว 46010912
นายอมรเทพ เผือกพิบูลย์ รหัสประจำตัว 46010925

อาจารย์ที่ปรึกษา ผศ.ดร.อรฉัตร จิตต์โสภักตร์

บทคัดย่อ

เนื่องด้วยปัจจุบันเทคโนโลยีด้านภาพเคลื่อนไหวได้เข้ามามีบทบาทกับชีวิตประจำวันของเรามากทั้งด้านความบันเทิง ทั้งด้านการทำงาน ทั้งด้านการประกอบธุรกิจ ดังนั้นจึงมีการพัฒนาความก้าวหน้าอยู่เรื่อยมาซึ่งมาตรฐานในการพัฒนานี้ได้มีผู้คิดค้นและพัฒนามากมาย ซึ่งปัจจัยสำคัญที่เป็นสิ่งที่เป็นที่ต้องการของผู้ที่จะเลือกใช้ก็คือขนาดของข้อมูลที่ได้หลังจากบีบอัดและคุณภาพของข้อมูลที่ได้หลังจากบีบอัดมาตรฐาน H.264 เป็นมาตรฐานบีบอัดภาพเคลื่อนไหวมาตรฐานใหม่ที่มีประสิทธิภาพมาก ทั้งด้านขนาดภาพเคลื่อนไหวและคุณภาพของภาพเคลื่อนไหวหลังทำการบีบอัด จึงเป็นที่คาดหวังโดยทั่วกันว่า มาตรฐานนี้จะกลายเป็นมาตรฐานที่สำคัญมาตรฐานหนึ่งในการพัฒนากระบวนการบีบอัดภาพเคลื่อนไหวในอนาคตต่อไป

โครงงานนี้จัดทำขึ้นมาเพื่อทำการศึกษาข้อมูลและการทำงานของมาตรฐานนี้โดยละเอียดพร้อมทั้งวิเคราะห์หาข้อดีข้อเสียของมาตรฐานนี้และทำการศึกษาค้นคว้าเกี่ยวกับโปรแกรมที่รองรับมาตรฐาน H.264 ทำการทดสอบและศึกษาการทำงานของโปรแกรมบนVisual Studioและทำการพัฒนาโปรแกรมที่รองรับการทำงานของมาตรฐานH.264นี้ลงบนBlackfin DSP เพื่อให้โปรแกรมสามารถทำงานได้อย่างมีประสิทธิภาพมากขึ้นโดยมีโปรแกรม Blackfin Visual DSP เป็นเครื่องมือช่วยในการพัฒนา

Project Title Development of Video Compression in H.264 Standard on Blackfin DSP

Author Anucha Hirunpanich Student ID 46010912

Amonthep Peukpiboon Student ID 46010925

Advisor Asst. Prof. Dr. Orachat Chitsobhuk

ABSTRACT

At present, video technology becomes an important part of our lives not only for work but also for entertainment and business. Consequently, there were several standards proposed in the literature. The important objectives for the users are the size of compressed video files and the quality of the decompressed video stream. H.264 video compression standard is proposed to efficiently compress the video data. The size of compressed video file is small while the quality is remarkable. Therefore, the H.264 standard is expected to be one of the most important standards for video compression in the recent future.

This project presents a study of the H.264 video compression standard and also discusses the benefit and weakness of this standard. The developed program implementing H.264 standard is revised in order to effectively run on both Visual Studio on PC and VisualDSP on Blackfin DSP board.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดีด้วยการได้รับความช่วยเหลือและชี้แนะจากหลายท่าน ผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษา ผศ.ดร.อรฉัตร จิตต์โสภักตร์ ที่ให้คำปรึกษาและความช่วยเหลือในด้านต่างๆ และขอขอบคุณ บริษัท ดีไซน์ เกทเวย์ จำกัด ที่ให้ความอนุเคราะห์ ในส่วนของตัวอย่างโปรแกรม ผู้จัดทำพึงระลึกอยู่เสมอว่าปริญญานิพนธ์ฉบับนี้จะไม่สามารถสำเร็จลงได้เลย หากขาดความช่วยเหลือจากทุกท่านจึงขอขอบพระคุณอย่างสูงมา ณ ที่นี้

อนุชา หิรัญพานิช

อมรเทพ เผือกพิบูลย์

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

เนื่องด้วยปัจจุบันเทคโนโลยีด้านภาพเคลื่อนไหวได้เข้ามามีบทบาทกับชีวิตประจำวันของเรา ทั้งด้านความบันเทิง ทั้งด้านการทำงาน ทั้งด้านการประกอบธุรกิจ ดังนั้นจึงมีการพัฒนาความก้าวหน้าอยู่เรื่องซึ่งมาตรฐานในการพัฒนานี้ได้มีผู้คิดค้นและพัฒนามากมายซึ่งปัจจัยสำคัญที่เป็นที่ต้องการของผู้ที่จะเลือกใช้คือขนาดของข้อมูลที่ได้หลังจากบีบอัดและคุณภาพของข้อมูลที่ได้หลังจากบีบอัดมาตรฐาน H.264ซึ่งเป็นมาตรฐานบีบอัดภาพเคลื่อนไหวมาตรฐานใหม่ที่มีประสิทธิภาพมากทั้งทางด้านขนาดและคุณภาพของภาพเคลื่อนไหวหลังทำการบีบอัด จึงเป็นที่คาดหวังโดยทั่วกันว่ามาตรฐานนี้จะกลายเป็นมาตรฐานที่สำคัญมาตรฐานหนึ่งในการพัฒนากระบวนการบีบอัดภาพเคลื่อนไหวในอนาคตต่อไป

ด้วยเหตุผลข้างต้นนี้ ผู้จัดทำจึงมีแนวคิดที่จะศึกษาวิธีการ เข้ารหัส / ถอดรหัส สัญญาณภาพเคลื่อนไหวตามมาตรฐาน H.264 และศึกษาการโปรแกรมลงบนบอร์ด ดี เอส พี ตระกูล Blackfin เพื่อที่จะได้ทำการสร้างไลบรารีในการ เข้ารหัส/ ถอดรหัส สัญญาณภาพเคลื่อนไหวตามมาตรฐาน H.264

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 เพื่อศึกษาและทำความเข้าใจกระบวนการบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐาน H.264

1.2.2 เพื่อศึกษาวิธีการโปรแกรมระบบฝังตัวบน DSP Platform ตระกูล Blackfin

1.2.3 เพื่อนำความรู้ที่ได้มาดัดแปลงและออพติไมซ์ (Optimize) เพื่อให้สามารถลงโปรแกรมบีบอัดสัญญาณวิดีโอตามมาตรฐาน H.264 บนบอร์ด DSP ตระกูล Blackfin ได้

1.3 ผลที่คาดว่าจะได้รับ

1.3.1 ได้รับความรู้ความเข้าใจเกี่ยวกับการบีบอัดข้อมูลสัญญาณวิดีโอตามมาตรฐาน H.264

1.3.2 สามารถสร้างไลบรารี (Library) ที่อ้างอิงมาตรฐานการบีบอัด สัญญาณภาพเคลื่อนไหวตามมาตรฐาน H.264 บน Blackfin DSP Board ได้

1.4 ขอบเขตของการพัฒนา

สำหรับโครงงานที่ได้จัดทำนี้จะใช้ภาษา C ในการพัฒนาอัลกอริทึมการบีบอัดภาพเคลื่อนไหวตามมาตรฐาน H.264 โดยใช้เครื่องมือในการพัฒนาคือ Visual DSP++

1.4.1 สามารถบีบอัดสัญญาณภาพเคลื่อนไหวตามมาตรฐาน H.264 บนบอร์ด DSP ตระกูล Blackfin

1.4.2 สามารถแสดงผลภาพเคลื่อนไหวขนาด 320*240 ที่ 0.9 เฟรมต่อวินาที ได้

1.5 ขั้นตอนการดำเนินงาน

โครงงานนี้แบ่งการดำเนินงานออกเป็น 3 ขั้นตอน

1.5.1 เป็นการศึกษาข้อมูลที่จำเป็นในการทำโครงงานนี้ ประกอบด้วย

1.5.1.1 ศึกษาการเข้ารหัส / ถอดรหัส สัญญาณภาพเคลื่อนไหวตามมาตรฐาน H.264

1.5.1.2 ศึกษามาตรฐานการเขียนโปรแกรมบน ดี เอส พี แพลตฟอร์ม(DSP Platform) ของ Blackfin และวิธีการใช้งาน Visual DSP++

1.5.1.3 ศึกษาวิธีการออพติไมซ์อัลกอริทึม

1.5.2 เป็นขั้นตอนในการพอร์ต (Port) อัลกอริทึม (Algorithmm) ลงบน ดี เอส พี แพลตฟอร์ม

1.5.2.1 ทำการวิเคราะห์อัลกอริทึมในการเข้ารหัส / ถอดรหัส ว่าประกอบด้วยโมดูลอะไรบ้าง

1.5.2.2 ทำการอิมพลีเมนต์ (Implement) อัลกอริทึมให้สามารถประมวลผลแบบเวลาจริง (real time) ได้

1.5.2.3 ทำการอิมพลีเมนต์ แต่ละโมดูลลงบน ดี เอส พี แพลตฟอร์ม ลงบนบอร์ด Blackfin

1.5.3 เป็นขั้นตอนในการออพติไมซ์โปรแกรม

1.5.3.1 เริ่มต้นจากการวัดการทำงานของโมดูลย่อยของโปรแกรมว่า โมดูลใดมีความซับซ้อน และใช้เวลาในการประมวลผลนานที่สุด

1.5.3.2 หาหัวข้อวิจัยที่เกี่ยวข้องกับโมดูลนั้น เพื่อนำมาศึกษาและประยุกต์กับขั้นตอนเดิมที่มีอยู่

1.5.3.3 ศึกษาการทำงานของ ดี เอส พี แพลตฟอร์มที่ใช้ ว่ามีลักษณะอย่างไร เช่นการจัดการกับหน่วยความจำหรือฟังก์ชันเฉพาะต่างๆที่ดีเอสพี แพลตฟอร์มนั้นสนับสนุนอยู่

1.5.3.4 ทำการพัฒนาโปรแกรมให้สอดคล้องกันทั้งมาตรฐาน H.264และมาตรฐาน ดีเอสพี แพลตฟอร์มที่ใช้

1.6 แผนการดำเนินงานในการทำโครงงาน

การดำเนินการในการทำโครงงานนี้จะเริ่มจากการทำงานตามขั้นตอนการดำเนินงานในช่วงที่ 1 นั่นคือ การศึกษาข้อมูลที่จำเป็นในการทำโครงงาน ซึ่งการศึกษาการเข้ารหัส / ถอดรหัส ภาพเคลื่อนไหวตามมาตรฐาน H.264 และ การศึกษาการเขียนโปรแกรมบน ดี เอส พี แพลตฟอร์ม (DSP Platform) ของ Blackfin และวิธีการใช้งาน Visual DSP++ นี้ได้เริ่มทำการมาตั้งแต่การฝึกงานภาคฤดูร้อนที่บริษัท ดี ไซน์ เกทเวย์ จำกัด แล้ว ดังนั้นในช่วงเวลาแรกในการทำโครงงานนี้จึงเป็นการทำต่อเนื่องจากเดิมนั่น

คือ การพัฒนาโปรแกรมให้มีความเร็วในการประมวลผลมากขึ้น เพื่อให้ตรงกับจุดประสงค์ของการทำโปรแกรมแบบเวลาจริง

ในช่วงแรกนี้จะทำการศึกษาโปรแกรมเพื่อที่จะได้รู้ว่าฟังก์ชันใดในโปรแกรมที่มีการทำงานที่ซับซ้อน และ ใช้เวลาในการประมวลผลมากที่สุด โดยเมื่อวัดค่าได้แล้ว จะทำการวิเคราะห์ในฟังก์ชันนั้นๆว่า การทำงานที่ซับซ้อน และ การประมวลผลที่ช้านั้น มีสาเหตุมาจากอะไร และมีแนวทางในการแก้ไขได้อย่างไร ซึ่งการวิเคราะห์การทำงานของฟังก์ชันนั้นจะทำได้โดยการเขียนแผนภาพขั้นตอนการทำงานโดยรวมของโปรแกรม และ ขั้นตอนย่อยของการทำงานในฟังก์ชันนั้นว่ามีการทำงานอย่างไรบ้าง เมื่อได้ทราบถึงการทำงานของฟังก์ชันนั้นแล้ว จึงทำการหาแนวทางในการพัฒนาโปรแกรมในส่วนของฟังก์ชันนั้นต่อไป

1.7 ส่วนประกอบของรายงาน

ในรายงานนี้ประกอบด้วยเนื้อหา 5 บทด้วยกัน

เนื้อหาในบทที่ 1 กล่าวถึงความเป็นมาของปัญหา วัตถุประสงค์ของโครงงาน ประโยชน์ที่คาดว่าจะได้รับ ขอบเขตของโครงงาน และส่วนประกอบของรายงานฉบับนี้

เนื้อหาในบทที่ 2 จะกล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงการ โดยจะมีโครงสร้างอัลกอริทึมในการบีบอัด และ ถอดการบีบอัดข้อมูล มีการจำแนกและแยกแยะความแตกต่างระหว่าง Profile ต่างๆที่ใช้ใน H.264 มีการอธิบายถึงโครงสร้างพื้นฐานของส่วนประกอบต่างๆในการเข้ารหัสข้อมูล โดยจะมีการอธิบายถึงกลุ่มของสไลซ์ , สไลซ์ และมาโครบล็อคทั้งแบบ Inter Macroblock และ Intra Macroblock อธิบาย Motion Estimation และ Motion Compensation ซึ่งภายในจะมีกระบวนการทำ Motion Search Algorithm และขั้นตอนการหา Motion Vector ที่เหมาะสมที่สุด มีการทำ Motion Compensated Prediction of Macroblock เพื่อการทำนายการชดเชยการเคลื่อนที่ของมาโครบล็อค, มีการทำ Discrete cosine transform (DCT) มีการทำ Transform and Quantization และการทำ Entropy coding (CAVLC)มีการเปรียบเทียบข้อดีข้อเสียของการทำมาตรฐาน MPEG-4 part 10 ที่เทียบกับมาตรฐานอื่นๆหรือเทียบกับมาตรฐาน MPEG ด้วยกันเองในมาตรฐานก่อนหน้านี้

เนื้อหาบทที่ 3 เป็นการออกแบบและพัฒนา โดยจะอธิบายถึงรายละเอียดและการทำงานของโปรแกรม รวมถึงแนวคิดในการพัฒนาโปรแกรมด้วย

เนื้อหาบทที่ 4 เป็นการทดลองและผลการทดลอง โดยจะอธิบายถึงปัญหาที่เกิดเนื่องจากการพัฒนาโปรแกรม และ แนวทางในการแก้ปัญหาเพื่อให้โปรแกรมสามารถทำงานได้ตรงตามความต้องการมากที่สุด

เนื้อหาในบทที่ 5 เป็นบทวิจารณ์และสรุป

บทที่2
ทฤษฎีที่เกี่ยวข้อง

2.1 การแนะนำมาตรฐาน H.264

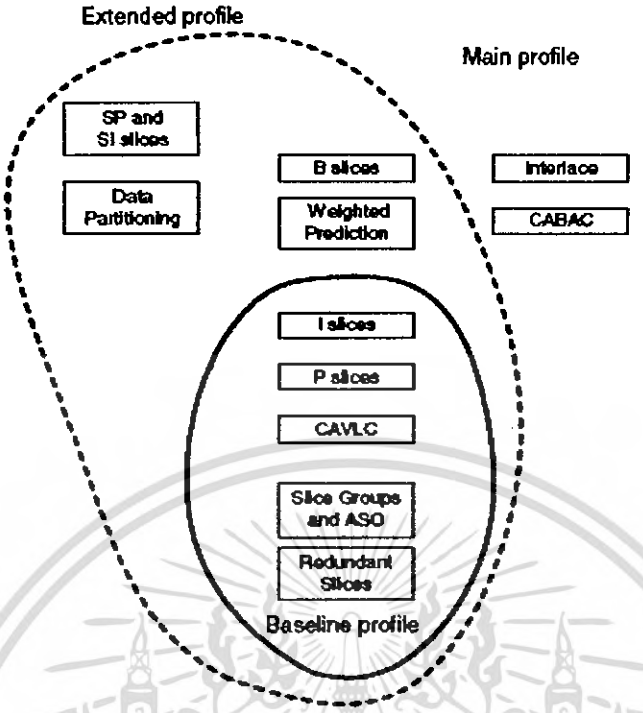
มาตรฐาน H.264 เป็นมาตรฐานร่วมของสองกลุ่มผู้เชี่ยวชาญ Moving Picture Expert Group (MPEG) และ Video Coding Expert Group (VCEG) ได้รับการพัฒนาเพื่อเป็นมาตรฐานใหม่สำหรับการเข้ารหัสวีดิทัศน์ โดยตั้งชื่อว่า "Advanced Video Coding (AVC)" และได้เป็นมาตรฐานการเข้ารหัสวีดิทัศน์ในชื่อ "ISO/IEC MPEG-4 Part 10" และ "ITU Recommendation H.264" โดยมีจุดมุ่งหมายเช่นเดียวกันมาตรฐาน MPEG-4 คือ เพื่อพัฒนามาตรฐานการเข้ารหัสวีดิทัศน์ให้ได้อัตราการบีบอัดสูง สำหรับใช้ในงานสื่อสารข้อมูลที่อัตราบิตต่ำ

ข้อมูลวีดิทัศน์ประกอบด้วยลำดับของภาพหรือเฟรม (Frame) โดยแต่ละเฟรมจะถูกดึงค่าตัวอย่าง (Sampling) ด้วยความละเอียดบางค่า และแต่ละจุดภาพจะถูกแทนด้วยจำนวนบิตจำกัด พื้นฐานของการบีบข้อมูลวีดิทัศน์คือ การลดความซ้ำซ้อน (Redundancy) คือในเฟรมมักจะมีบริเวณที่มีโครงสร้างไม่เป็นแบบสุ่ม และการเปลี่ยนแปลงที่เกิดขึ้นระหว่างเฟรมที่ต่อเนื่องกันมักจะน้อย จุดภาพที่อยู่ใกล้กันทั้งทางพื้นที่หรือทางเวลามักจะมี Correlation สูง

แต่ละเฟรมถูกแบ่งเป็นบล็อก (Block) ขนาด 16x16 จุดภาพเรียกว่า แมโครบล็อก (Macroblock) สำหรับแต่ละแมโครบล็อกการทำนายชดเชยการเคลื่อนที่จะทำโดยประมาณค่าการเคลื่อนที่จากเฟรมที่ผ่านการถอดรหัสก่อนหน้า ทำให้ข้อมูลที่ต้องใช้ในการถอดรหัสเป็นแค่เพียงเฟรมผลต่างส่วนเหลือ (Residue error) ซึ่งคือผลต่างระหว่างเฟรมที่ได้จากการทำนายกับเฟรมปัจจุบัน ถ้าการทำนายชดเชยการเคลื่อนที่ดี พลังงานของผลต่างส่วนเหลือนี้จะน้อย เฟรมที่ใช้ในการทำนายมักถูกเรียกว่าเฟรมอ้างอิง และเพื่อให้ลดความซ้ำซ้อนได้มากขึ้นจะใช้การ Transform ระดับบล็อกกับเฟรมส่วนต่างๆ จากนั้นจึงใช้การQuantization สัมประสิทธิ์การแปลง และสุดท้ายเข้ารหัสเอนโทรปี (Entropy Coding) สัมประสิทธิ์ที่ได้ จึงเสร็จสิ้นกระบวนการบีบข้อมูล

2.2 H.264/MPEG-4 part 10 profile

ในมาตรฐาน H.264 (MPEG-4 part 10) นี้มีการแบ่งProfile ออกเป็น3ประเภทด้วยกัน นั่นคือ Baseline Profile , Main Profile , Extension Profile และ High Profile ดังรูปข้างล่างนี้



รูปที่ 2.1 แสดง Profileต่างๆ

2.2.1 Baseline Profile

Baseline Profile นี้เป็นโพรไฟล์พื้นฐานในการ implement โดยมีส่วนประกอบในโพรไฟล์คือ สไลซ์ I และ สไลซ์ P โดย สไลซ์ I นั้นจะเป็น สไลซ์ที่ไม่มีการอ้างถึงสไลซ์อันอื่น จะทำการเข้ารหัสตัวเองเมื่อทำการเข้ารหัสเสร็จแล้วจะเก็บไว้เป็นสไลซ์ที่เอาไว้ใช้อ้างอิงต่อไป ส่วนสไลซ์ P นั้นจะมีการอ้างถึงสไลซ์ที่เข้ารหัสก่อนหน้านี้ ซึ่งจะเป็นการทำให้ลดขนาดของบิทเรทลงได้ เนื่องจากไม่ต้องทำการเข้ารหัสในทุกๆบิทนั่นเอง ส่วนประกอบต่อมาคือการเข้ารหัสเอนโทรปีของ Baseline Profile นี้จะเข้ารหัสแบบ CAVLC (Context-based Adaptive Variable Length Coding) ซึ่งจะช่วยในการลดขนาดบิทลงได้

ลักษณะเด่นของ Baseline Profile คือ

1. Flexible Macroblock Order นั่นคือว่ามาโครบล็อคที่ทำไม่จำเป็นจะต้องเรียงตามลำดับของ Raster Scan

2. Arbitrary Slice Order คือ ตำแหน่งที่อยู่ของมาโครบล็อคของมาโครบล็อคแรกของสไลซ์ของรูปภาพ อาจจะเล็กกว่า ตำแหน่งที่อยู่ของมาโครบล็อคของมาโครบล็อคแรกของสไลซ์อื่นในรูปที่เข้ารหัสเดียวกันได้

3. Redundant Slice คือ สไลซ์นี้จะเป็นของ coded data ที่ได้รับมาจาก coding rate เดียวกันหรือต่างกันได้เมื่อเทียบกับ coded data ก่อนหน้านี้ในสไลซ์เดียวกัน ซึ่ง Baseline Profile นี้จะเหมาะกับการใช้งานประเภท Video Conference และ Mobile Video

2.2.2 Main Profile

Main Profile เหมาะสำหรับการใช้งานในทางสื่อสาร เช่น สัญญาณโทรทัศน์แบบดิจิตอล และดิจิตอลวีดีโอ Main Profile เกือบจะครอบคลุม Baseline Profile ทั้งหมด ยกเว้นแต่ multiple slice- group จะไม่สนับสนุน ASO และ redundant slices (ซึ่งทั้งหมดนี้อยู่ใน Baseline Profile) เครื่องมือที่มีใน Main Profile คือ B slices (bi-predict slices เพื่อการ coding ที่มีประสิทธิภาพมากกว่า), weighted prediction (เพิ่มความยืดหยุ่นในการสร้าง motion-compensated prediction block) สนับสนุน interlaced video (การเข้ารหัส fields เช่นเดียวกับ frame) และ CABAC (an alternative entropy coding method based on Arithmetic Coding)

B slices

แต่ละ partition ของ macroblock ใน inter coded macroblock อาจจะถูกทำนายจากรูปภาพอ้างอิง 1-2 ภาพ ก่อนหรือหลังรูปภาพปัจจุบัน ขึ้นอยู่กับว่ารูปภาพอ้างอิงอยู่ในส่วน ตัวเข้ารหัส และ ตัวถอดรหัส ทำให้มีทางเลือกหลากหลายในการเลือกรูปภาพอ้างอิงที่ใช้ทำนาย

Weighted Prediction

Weight prediction เป็นวิธีการของการแก้ไข (scaling) ตัวอย่าง motion-compensated ข้อมูลการทำนายใน P หรือ B slice macroblock วิธี weighted prediction สามแบบใน H.264 คือ

1. P slice macroblock, 'explicit' weighted prediction:
2. B slice macroblock, 'explicit' weighted prediction:
3. B slice macroblock, 'implicit' weighted prediction:

2.2.3 Extended Profile

เป็นโพรไฟล์พิเศษที่ขยายออกมาจาก Main Profile ซึ่งจะประกอบไปด้วยส่วนประกอบทั้งหมดของ Baseline Profile(flexible macroblock order, arbitrary slice order, redundant slice) รวมไปถึงสไลซ์ B และ Weighted Prediction ของ Main Profileด้วย ซึ่งส่วนที่เพิ่มเติมขึ้นมาของ Extended Profile คือมี สไลซ์ SP , SI และ Data Partition

SP เป็นสไลซ์ที่ถูกเข้ารหัสแบบพิเศษซึ่งทำให้การสลับระหว่างVideo Stream นั้นมีคุณภาพมากขึ้น ซึ่งทำหน้าที่คล้ายๆกับ สไลซ์ P ส่วน SI เช่นเดียวกันเพียงแต่ทำหน้าที่คล้ายๆกับ สไลซ์ I นั่นเอง ในขณะที่ Data Partition นั้นข้อมูลที่ถูกเข้ารหัสจะถูกวางแยกไว้ใน data partition ซึ่งในแต่ละพาร์ทิชัน

สามารถใส่ไว้ในLayer Unitที่ต่างกันได้ ซึ่ง Extended Profile นี้เหมาะที่จะใช้ในการทำ Streaming Video

2.2.4 High Profile

เป็นโพรไฟล์ที่ทำการรวม Main Profile ไว้ทั้งหมดแล้วยังมีส่วนเพิ่มเติมคือ Adaptive Transform Block Size คือการที่สามารถเลือกได้ว่าจะใช้ Luma Sample ในการ Transform ขนาด 4*4 หรือ 8*8 และอีกอย่างที่เพิ่มเข้ามาคือ มี Quantization Scaling Matrices คือมีการกำหนดขนาดที่แตกต่างของความถี่ในการทำTransform สัมประสิทธิ์ ที่จะใช้ในกระบวนการทำ Quantization ให้มีประสิทธิภาพมากที่สุด ซึ่ง High Profile นี้เหมาะสำหรับการทำ Studio Distribution

ตารางที่2.1 แสดงการใช้งาน Profileต่างๆ

Application	Requirements	H.264 Profiles	MPEG-4 Profiles
Broadcast television	Coding efficiency, reliability (over a controlled distribution channel), interlace, low-complexity decoder	Main	ASP
Streaming video	Coding efficiency, reliability (over a uncontrolled packet-based network channel), scalability	Extended	ARTS or FGS
Video storage and playback	Coding efficiency, interlace, low-complexity encoder and decoder	Main	ASP
Videoconferencing	Coding efficiency, reliability, low latency, low-complexity encoder and decoder	Baseline	SP
Mobile video	Coding efficiency, reliability, low latency, low-complexity encoder and decoder, low power consumption	Baseline	SP
Studio distribution	Lossless or near-lossless, interlace, efficient transcoding	Main High Profiles	Studio

จากตารางข้างบนเป็นการเปรียบเทียบให้เห็นว่าในแต่ละ Profile นั้นมีการนำไปใช้ที่แตกต่างกันสำหรับแต่ละชนิดของProfileอย่างไร (ซึ่งในรายงานนี้ ชื่อของProfile ที่อ้างถึงจะใช้เป็นชื่อ Profile ตามแบบของ H.264 Profile)

2.3 โครงสร้างพื้นฐานของส่วนประกอบต่างๆในการเข้ารหัสข้อมูล

2.3.1 Slice Group

กลุ่มของสไลซ์นั้นเป็นสับเซตของมาโครบล็อคหลายๆมาโครบล็อคที่รวมกันอยู่ในเฟรมที่เข้ารหัส ซึ่งกลุ่มของสไลซ์คือสไลซ์หลายๆอันที่เอามาต่อกันเพื่อที่จะประกอบเข้ากลายเป็นเฟรมนั่นเอง ซึ่งใน สไลซ์แต่ละอันที่อยู่ในกลุ่มสไลซ์นั้นจะมีมาโครบล็อคที่ถูกเข้ารหัสอยู่เรียงกันตามลำดับของ Raster Scan ซึ่งถ้าการเข้ารหัสใน 1 เฟรมใช้ กลุ่มสไลซ์เพียงกลุ่มเดียวจะหมายความว่ามาโครบล็อคที่อยู่ในเฟรมนั้นๆจะถูกเข้ารหัสแบบ Raster Scan แต่ถ้าไม่ใช่ก็จะใช้ ASO ซึ่งไม่จำเป็นต้องเรียงกันเพราะจะเรียงลำดับอย่างไรก็ได้ตามตัวถอดรหัส ในขณะที่ถ้าใช้กลุ่มสไลซ์มากกว่า 1 กลุ่มสไลซ์ในการเข้า

รหัส จะเป็นการทำ FMO (Flexible Macroblock Order) ซึ่งจะสามารถทำให้จับคู่ลำดับของมาโครบล็อค ที่ถูกเข้ารหัสไปเป็นมาโครบล็อคที่ถูกถอดรหัสได้หลากหลายรูปแบบ โดยการจองตำแหน่งของมาโค รบล็อคหาได้จากแผนที่กลุ่มสไลซ์ (Slice group map) ซึ่งบอกว่าแต่ละมาโครบล็อคอยู่ในกลุ่มสไลซ์ใด ดังตัวอย่างรูปข้างล่างนี้

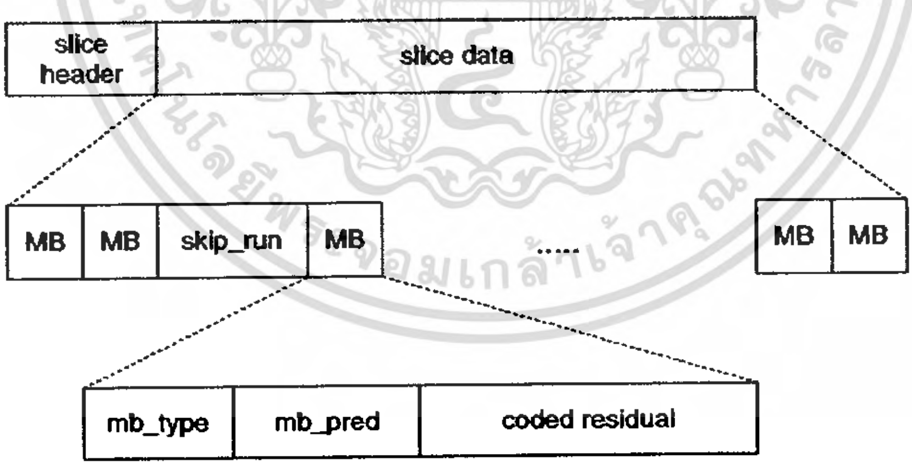
0
1
2
0
1
2
0
1
2

0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2

รูปที่ 2.2 แสดงให้เห็นระดับและการกระจายของแต่ละกลุ่มสไลซ์

2.3.2 Slice

ในเฟรมหนึ่งๆนั้นจะมีสไลซ์หนึ่งหรือหลายสไลซ์ก็ได้ ซึ่งในหนึ่งสไลซ์จะประกอบไปด้วย หลายๆมาโครบล็อคอีกที โดยในสไลซ์จะประกอบไปด้วย Slice Header จะเป็นตัวกำหนดลักษณะ ต่างๆของสไลซ์ว่าเป็นสไลซ์ประเภทไหน และ Slice Data จะเก็บข้อมูลที่จะเอามาใช้ของสไลซ์ซึ่งจะ เป็นข้อมูลต่างๆของมาโครบล็อคอีกทีหนึ่งดังจะเห็นได้จากรูปข้างล่างนี้



รูปที่ 2.3 แสดงSyntax ของ สไลซ์

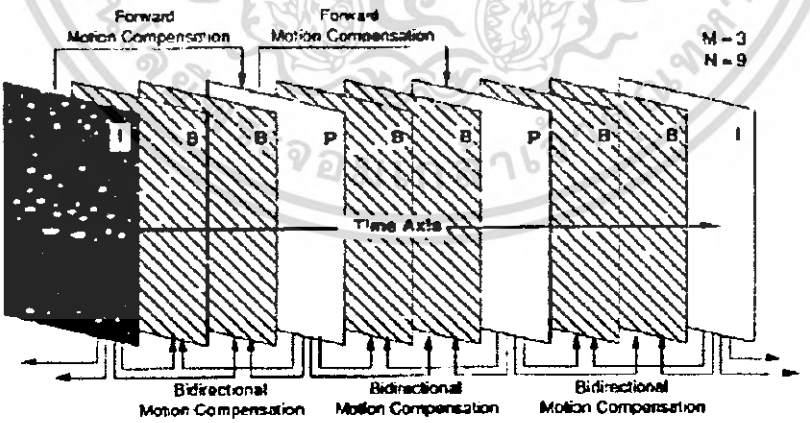
ใน มาตรฐาน H.264 นี้มีการแบ่ง Slice เป็น 5 ประเภทด้วยกันนั่นคือ สไลซ์ I , สไลซ์ P ,สไลซ์ B , สไลซ์ SP และ สไลซ์ SI

I-Slice จะเป็นการเข้ารหัสภาพโดยที่ไม่มีการอ้างอิงภาพก่อนหน้านี้เลย และจะไม่มีการทำเกี่ยวกับการเคลื่อนที่เช่น การประมาณการเคลื่อนที่ การชดเชยการเคลื่อนที่เป็นต้น เมื่อ สไลซ์ I ได้ทำการเข้ารหัสแล้วจะถูกเก็บเอาไว้เพื่อที่จะใช้เป็นภาพอ้างอิงต่อไปสำหรับสไลซ์ P และ สไลซ์ B

P-Slice จะเป็นการเข้ารหัสโดยมีการทำการประมาณการเคลื่อนที่ และการหาผลต่างของการเคลื่อนที่เพื่อใช้ทำการชดเชยการเคลื่อนที่ ซึ่งสไลซ์ P นี้จะมีการอ้างอิงถึงภาพก่อนหน้านี้ด้วยเพื่อใช้ในการคำนวณการเคลื่อนที่ได้ โดยสไลซ์ที่ใช้อ้างอิงนั้นจะมีได้ 2 ประเภทนั่นคือสไลซ์ I และ สไลซ์ P เอง เนื่องจากตัวมันเองเมื่อทำการเข้ารหัสแล้วยังสามารถเก็บไว้เพื่อใช้เป็ตัวอ้างอิงต่อไปได้อีกด้วย

B-Slice จะเป็นการเข้ารหัสโดย B-sliceนี้จะใช้ สไลซ์อ้างอิงของรูปภาพอ้างอิง ทั้งจากสไลซ์ P และ สไลซ์ I ซึ่งจะมี list ที่ทำให้สามารถเก็บรูปภาพที่ถูกเข้ารหัสก่อนหน้า และ/หรือภายหลังภาพปัจจุบันได้ เพียงแต่ สไลซ์ B นี้ไม่สามารถเอามาใช้เป็นภาพอ้างอิงเหมือนกับสไลซ์ I หรือ สไลซ์ P ได้นั่นเอง

SP-Slice และ **SP-Slice** เป็นสไลซ์ที่ถูกเข้ารหัสแบบพิเศษเพื่อที่จะสามารถสับเปลี่ยนระหว่าง Video Stream ได้อย่างมีประสิทธิภาพ ซึ่งเป็นความต้องการพื้นฐานในการส่งข้อมูลแบบ Streaming และตัวถอดรหัสวิดีโอที่จะต้องการเปลี่ยนความเร็วในการส่งข้อมูลเช่นในการส่งข้อมูลวิดีโอหนึ่งๆด้วยบิทเรทที่สูง เมื่อตัวถอดรหัสรับข้อมูลแล้วอาจจะจำเป็นที่จะต้องมีการปรับเปลี่ยนบิทเรทให้ต่ำลงถ้าความสามารถในการรับข้อมูลบิทเรทนั้นๆไม่สูงพอ เป็นต้นซึ่ง SI นั้นจะเป็นการเข้ารหัสภายในตัวของมันเองไม่มีการอ้างอิงถึงสไลซ์อื่นให้เข้ามาช่วยเข้ารหัสด้วย ซึ่งจะคล้ายกับ I-Slice มากที่เดียว ส่วนSP นั้นจะเป็นการเข้ารหัส โดยมีการอ้างอิงถึงสไลซ์ก่อนหน้านี้ไม่ว่าจะเป็น SI หรือ SP ด้วยกันก็ตามซึ่งจะมีลักษณะเหมือนกับ P-Slice นั่นเอง



รูปที่ 2.4 แสดงความสัมพันธ์แต่ละสไลซ์

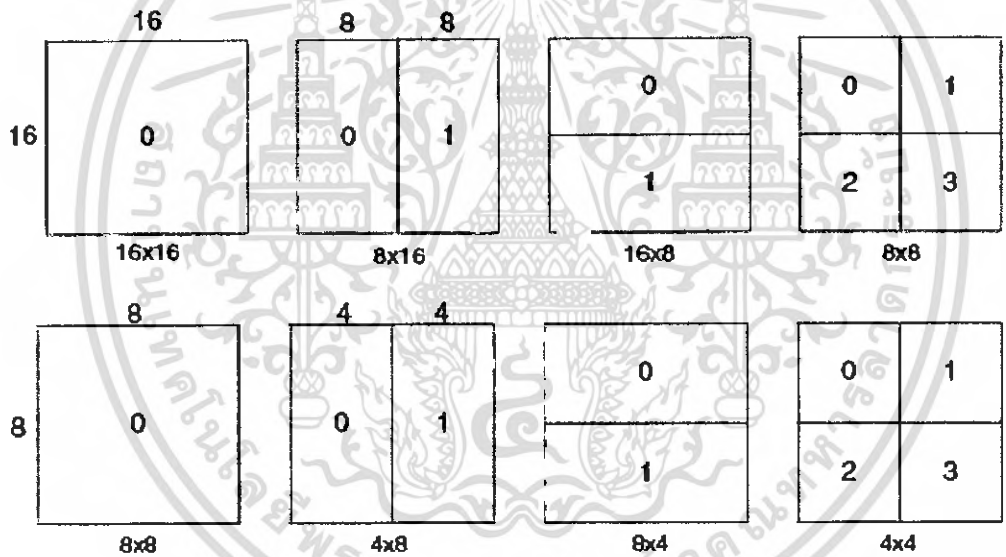
จากรูปเป็นการแสดงให้เห็นถึงความสัมพันธ์กันในการใช้งานของ I-Slice , P-Slice และ B-Slice

2.3.3 Macroblock

มาโครบล็อคนี้จะประกอบไปด้วยข้อมูล 16*16 Sample และจะมีการรวมกันเป็นกลุ่มที่มีลักษณะคล้ายกันเป็น Slice และ Frame ต่อไปซึ่งมาโครบล็อคเปรียบเหมือนหน่วยพื้นฐานในการมองภาพโดยรวมเลยทีเดียว (ไม่นับ pixel ในการ Encode Entropy) ซึ่งมาโครบล็อคนี้จะแบ่งออกเป็น 2 ประเภท นั่นคือ Inter Macroblock และ Intra Macroblock

2.3.3.1 Inter prediction

Inter prediction จะทำการสร้างแบบตัวอย่างการทำนาย จากหนึ่งหรือหลายเฟรมที่ถูกเข้ารหัสวิดีโอก่อนหน้า หรือส่วนพื้นที่การใช้บล็อก-การเคลื่อนไหวพื้นฐานแทน แตกต่างจากมาตรฐานล่าสุดโดยสนันสนุนระยะขนาดของบล็อกที่เปลี่ยนแปลงได้ (จาก16x16 ถึง4x4)
และ fine subsample เวกเตอร์การเคลื่อนไหว (หนึ่งในสี่-ความละเอียดตัวอย่างใน luma คอมโพเนนท์) ในส่วนนี้เราจะอธิบายเครื่องมือ interprediction ที่มีให้ในBaseline profile ส่วนเพิ่มเติมของเครื่องมือเหล่านี้ ในส่วน Main และ Extended profiles ประกอบด้วย B-slices และ Weighted Prediction



รูปที่ 2.5 แสดงมาโครบล็อคและซับมาโครบล็อค

Tree structured motion compensation

Luminance Component ของแต่ละ macroblock (ตัวอย่างเช่น 16x16) อาจจะแบ่งได้ 4 แบบโดยแทนการเคลื่อนไหว ของ 16x16 macroblock partition, สอง 16x8 partition, สอง 8x16 partition หรือสี่ 8x8 partition ซึ่งถ้าเลือกโหมด 8x8 แต่ละ 8x8 sub-macroblocks ภายใน macroblock อาจจะแบ่งได้อีก 4 แบบคือ 8x8 sub-macroblock partition,สอง 8x4 sub-macroblock partition,สอง 4x8 sub-macroblock partition หรือสี่ 4x4 sub-macroblock partition โดยที่ทั้ง partition และ sub-macroblockเหล่านี้ จะทำให้

มีรูปแบบการแบ่งหลากหลายมากขึ้นในแต่ละmacroblock วิธีการแบ่ง macroblocks นี้เป็นตัวแทนการเคลื่อนไหวของ sub-blocks ที่เปลี่ยนแปลงขนาดได้เรียกว่า Tree structured motion compensation

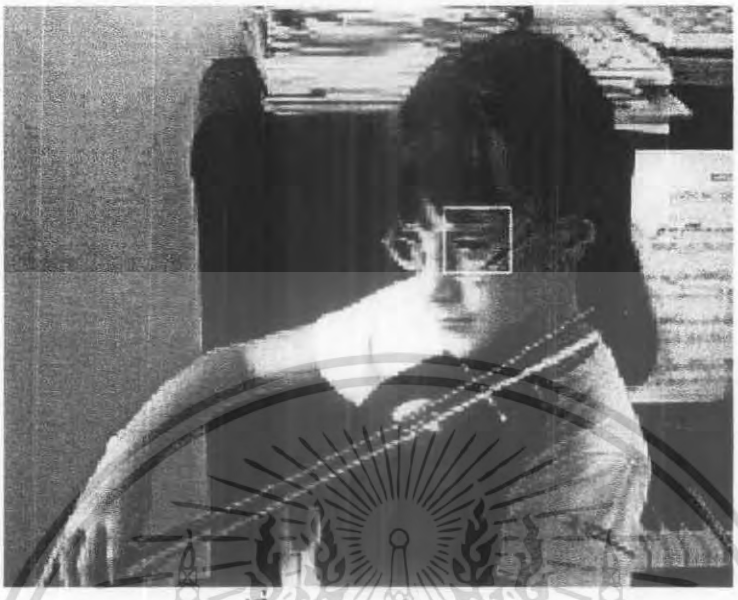
เวกเตอร์การเคลื่อนไหวที่แยกออกมานั้นมีจะต้องมีพาทิชั่นหรือ sub-macroblock โดยแต่ละเวกเตอร์การเคลื่อนไหวจำเป็นต้องถูกเข้ารหัส และตัวเลือกต่างๆ ของพาทิชั่นนั้นจำเป็นต้องเข้ารหัสใน bitstream ที่ถูกบีบอัด การเลือกขนาดพาทิชั่นที่ใหญ่ (16x16,16x8,8x16) จะทำให้จำนวนบิตน้อยลงสำหรับสัญญาณ 1 ตัวเลือก ของเวกเตอร์การเคลื่อนไหวและชนิดของพาทิชั่น ส่วนการเลือกพาทิชั่นเล็กขนาด (8x4,4x4 และอื่นๆ) จะทำให้ขนาดของข้อมูลต่ำ แต่ว่าต้องการบิตจำนวนมากสำหรับการเข้ารหัสเวกเตอร์การเคลื่อนไหวและตัวเลือกของพาทิชั่นขนาดพาทิชั่นของตัวเลือกนั้นมีผลกระทบสำคัญต่อคุณภาพการบีบอัด โดยทั่วไปพาทิชั่นขนาดใหญ่เหมาะสำหรับพื้นที่ซึ่งเป็นเนื้อเดียวกันของเฟรม และขนาดพาทิชั่นเล็กอาจจะเหมาะสำหรับพื้นที่ที่มีรายละเอียดแต่ละ chroma component ในหนึ่ง macroblock (CbและCr) มีความละเอียดของแนวนอน และแนวตั้งเป็นครึ่งหนึ่งของ luminance(luma) คอมโพแนนท์แต่ละ chroma บล็อกคือพาทิชั่นเช่นเดียวกัน luma Component ยกเว้นแต่ว่าขนาดของพาทิชั่นมีความละเอียดในแนวนอน และแนวตั้งเป็นครึ่งหนึ่งพอดี (8x16 พาทิชั่นใน luma ตรงกับ 4x8 พาทิชั่นใน chroma ; 8x4 พาทิชั่นใน luma ตรงกับ 4x2 ในchroma และต่อมา) ส่วนประกอบในแนวนอนและแนวตั้ง ของแต่ละเวกเตอร์การเคลื่อนไหว (หนึ่งต่อพาทิชั่น) คือครึ่งหนึ่งเมื่อใช้ chroma บล็อก

รูปที่ 2.6 แสดงตัวอย่าง Residual

2.3.3.2 Intra Prediction

ใน Intra Predicion modeนี้ บล็อก P จะมีรูปแบบในการอ้างถึงการเข้ารหัสและการสร้างบล็อกใหม่โดยมีการลบเพื่อหาความแตกต่างของบล็อกปัจจุบันกับบล็อกก่อนหน้านี้ ตัวอย่างเช่น สมมติให้ P

อยู่ในรูปแบบ 4*4 block หรือ 16*16 macroblock จะมี9ตัวเลือกในการทำนาย สำหรับ 4*4 luma block , 4 โหมด สำหรับ 16*16 luma block และ 4โหมดสำหรับแต่ละ chroma component



รูปที่ 2.7 แสดง เฟรม QCIF

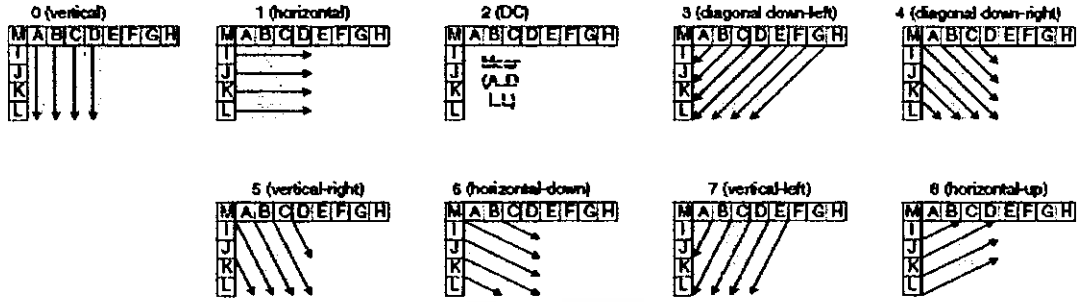
4*4 Luma Prediction Modes

จากรูปตัวอย่าง เป็น 4*4 Luma block ซึ่งเป็นส่วนที่ต้องการการทำนายโดยมีการเข้ารหัสและทำนายดังตัวอย่าง

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

รูปที่ 2.8 แสดงการ Prediction Sample ขนาด 4*4

ซึ่งจะเห็นว่ามีส่วนของsample block A-M อยู่รอบๆนั่นคือ บล็อคตัวอย่างที่ทำการ encode และ reconstructแล้วดังนั้นจึงนำมาใช้ในการเป็นตัวอ้างอิง(reference) และตัว sample a,b,c,...,p ของบล็อค P จะเป็นบล็อกที่คำนวณมาจาก sample A-Mนั่นเอง โดยรูปแบบต่างๆในการทำนายนี้จะมีทั้งหมด9โหมด



รูปที่ 2.9 ตัวอย่างสำหรับการทำ4*4 Luma Prediction

สำหรับตัวอย่างในรูปข้างบนนั้น เมื่อเอามาทำการเข้าโหมดทำนาย จะมีการหาค่าSum of Absolute Error(SAE) ในแต่ละโหมดการทำนาย ซึ่งจากตัวอย่างที่ได้นั้น ค่า SAE ออกมาแล้ว การทำนายโหมด 8 นั้นถือว่ามีความผิดพลาดที่น้อยที่สุด และผลที่ได้ออกมาถือว่าใกล้เคียงภาพจริงมากที่สุดด้วยเช่นกัน

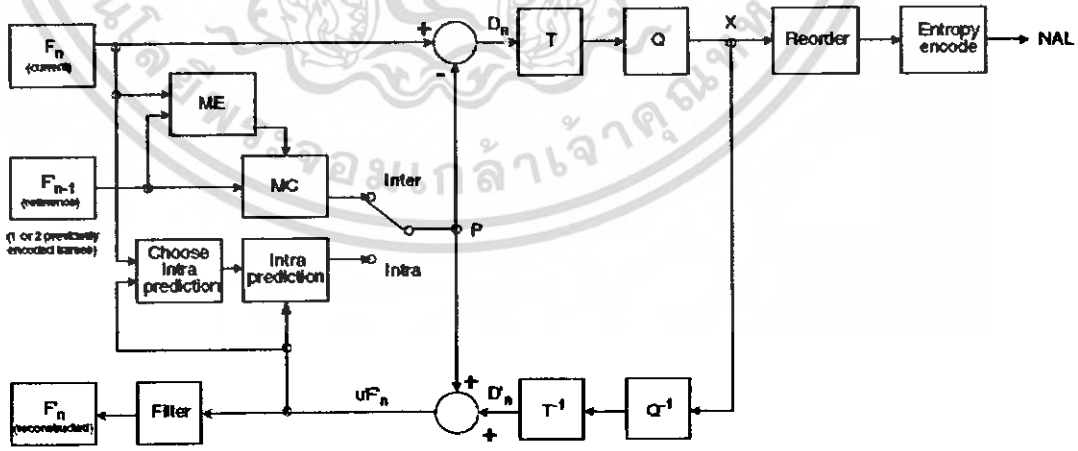
16*16 Luma Prediction Mode

ในส่วนของ 16*16 Luma Prediction Mode จะมีการทำนายเช่นเดียวกัน แต่จะใน 16*16 Luma นี้จะมีโหมดทำนายเพียงแค่4โหมดเท่านั้น

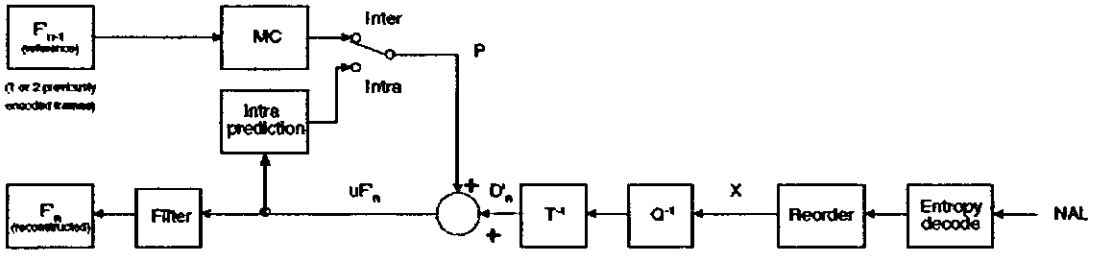
8*8 Chroma Prediction Mode

ในส่วนของ 8*8 Chroma นี้ระมีโหมดการทำนาย4โหมดเหมือนกับ 16*16 Luma Prediction เว้นแต่ว่า หมายเลขลำดับโหมดนั้นแตกต่างกัน คือ Mode 0 คือ DC , Mode 1 คือ horizontal , Mode 2 คือ Vertical และ Mode 3 คือ Plane

2.4 โครงสร้างอัลกอริทึมในการบีบอัดและถอดการบีบอัดข้อมูล



รูปที่2.10 H.264 Encoder



รูปที่2.11 H.264 Decoder

จากรูปเป็นการแสดงให้เห็นถึงการเข้ารหัสและถอดรหัสภาพ โดยที่ในส่วนของการเข้ารหัสและถอดรหัสของ H.264 นี้จะไม่ได้แตกต่างอะไรไปจากมาตรฐานเดิมมากนัก โดยในส่วนของฟังก์ชันการทำงานต่างๆไม่ว่าจะเป็น Prediction , Transform , Quantization , และการเข้ารหัส Entropy นั้น ก็ได้มีมาตั้งแต่มาตรฐานรุ่นก่อนๆ(MPEG-1,MPEG-2,MPEG-4,H.261,H.263)แล้ว เพียงแต่รายละเอียดภายในของฟังก์ชันเหล่านี้มีการเปลี่ยนแปลงโดยปรับเปลี่ยนให้ดีขึ้นกว่าเดิม

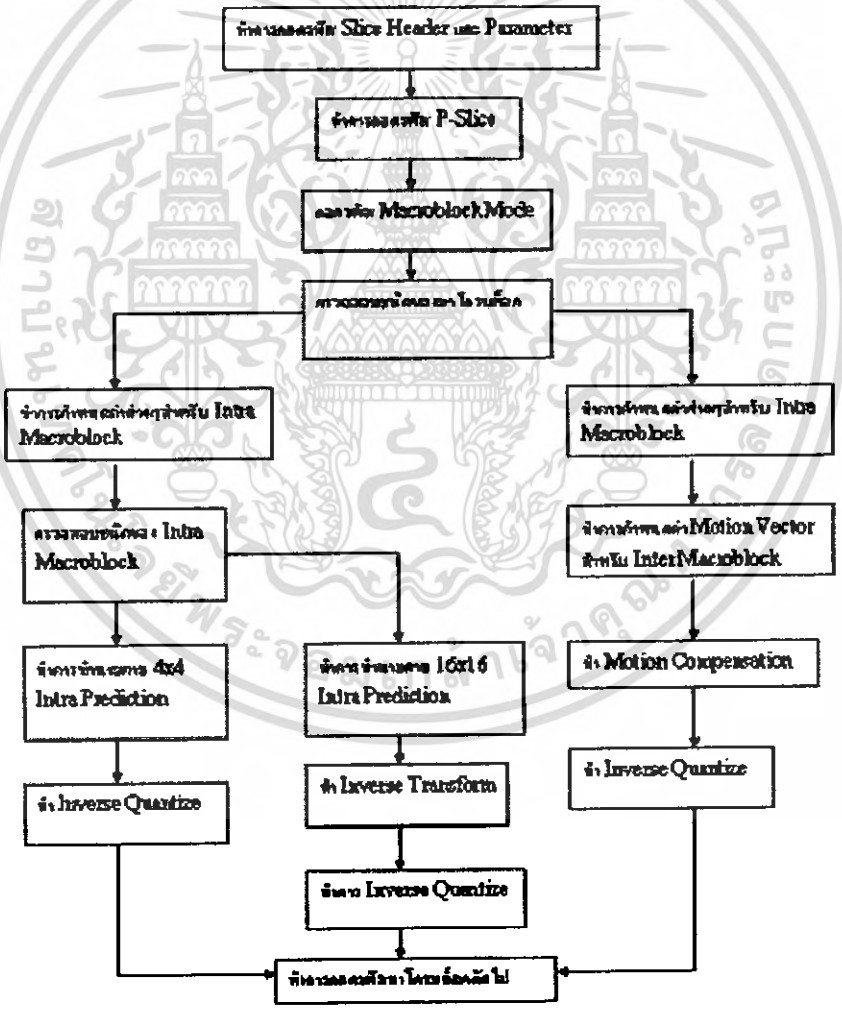
ในส่วนของการเข้ารหัสนั้นจะมีเส้นทางโดยรวมทั้งหมด 2 เส้นทางคือ "Forward Path" และ "Reconstruction Path" ซึ่งในส่วนของ Forward Path นั้นจะเป็นเส้นทางที่มีจุดประสงค์เพื่อเข้ารหัสเสร็จแล้วจึงส่งไปยัง NAL (Network Abstraction Layer) ซึ่งเริ่มแรก เมื่อมาโครบล็อคกำลังทำการเข้ารหัสนั้นจะอยู่ใน F_n ซึ่งถือเป็นสถานะปัจจุบันที่กำลังดำเนินการอยู่ซึ่งในแต่ละบล็อคของมาโครบล็อคนั้น จะผ่านไปยัง P (Prediction Mode) ซึ่งเป็นการทำนายมาโครบล็อค แบ่งออกเป็นสองทาง นั่นคือ Intra Prediction Mode , Inter Prediction Mode ซึ่งถ้าเป็น Intra Mode จะทำการจัดรูปแบบให้อยู่ในรูปแบบโดยดูจาก Sample Slice ก่อนหน้านี้ ในขณะที่ Inter Mode จะทำการอ้างอิงถึงภาพอ้างอิงที่อยู่ก่อนหน้านี้ซึ่งจากรูป ภาพอ้างอิงจะอยู่ในส่วนของ F_{n-1} ซึ่งจริงๆแล้วสามารถเลือกได้ทั้งภาพที่ผ่านมาและภาพข้างหน้าที่ทำการเข้ารหัสเรียบร้อยแล้วก็ได้เช่นกัน

เมื่อทำการทำนายแล้ว ต่อไปจะเอาภาพที่ทำนายนั้นไปลบออกจากบล็อคในปัจจุบันที่กำลังทำอยู่ เมื่อได้ส่วนต่างมานั่นจะกลายเป็น D_n ซึ่งจะเอาไปทำการ Transform และQuantize ผลที่ได้ออกมาในรูปในที่นี้ให้แทนด้วย X ซึ่งเป็นสัมประสิทธิ์ที่ได้จากกระบวนการ Transform และ Quantize แล้วจะทำการจัดเรียงใหม่ (Reorder) และเข้ารหัส Entropy เสร็จแล้วจึงจะส่งไปยัง NAL เพื่อที่จะทำการส่งต่อหรือเก็บเอาไว้ก่อนเพื่อที่จะส่งไปยังส่วนของการถอดรหัสต่อไป

ในส่วนของการเข้ารหัส โดยใช้เส้นทาง Reconstruction Path นั้นทำเช่นเดียวกันกับ Forward Path ในตอนต้น เพียงแต่เมื่อถึงในส่วนของการได้ X ที่เป็นสัมประสิทธิ์จากการTransform และ Quantizeแล้ว จะลงมายังการทำ Inverse Quantize และ Inverse Transform แทนจะได้ผลต่าง D'_n ซึ่งจะทำการสร้าง reconstructed block ซึ่งคือ uF'_n จากนั้นจะผ่าน Filter เพื่อลดความผิดเพี้ยนที่เกิดจากการ Transform และ Quantize แล้วเราจะได้รูปที่ใช้อ้างอิงที่ได้จากการ reconstruct มา

ในส่วนของการถอดรหัสนั้น(จากรูปให้ดูจากขวามาซ้ายในส่วนของการถอดรหัส) ตัวถอดรหัสจะได้รับชุดข้อมูลที่ถูกบีบอัดที่มาจาก NALในส่วนของการเข้ารหัสแล้ว จะทำการถอดรหัส Entropy แล้วทำการเรียงใหม่ (Reorder) ซึ่งจะได้ X (สัมประสิทธิ์ที่ผ่านการเข้ารหัสแล้ว) มาจากนั้นจะผ่านการ Inverse Quantize และ Inverse Transform ซึ่งจะทำให้ได้ผลต่าง (D'_n ซึ่งจะเหมือนกับ D'_n ในส่วนของการเข้ารหัสทุกประการ) แล้วจะทำการดูข้อมูลในส่วนหัว (Header) ของชุดข้อมูล เพื่อที่จะรู้ว่าการเข้ารหัสนั้น ใช้การทำนาย (Prediction) แบบใดแล้วจะทำการถอดรหัสแบบนั้น ซึ่งต่อมาจะทำการเพิ่ม D'_n เข้าไปให้กับ F'_n (ซึ่งในส่วนของการเข้ารหัสนั้นเราลบออก) ที่ถูกกรองแล้ว เราจะได้บล็อคที่ได้รับการถอดรหัส (F'_n) มา

2.5 ขั้นตอนการทำงานของโปรแกรมการเข้ารหัสของ H.264



รูปที่2.12 แสดงการทำงานโดยรวมของโปรแกรม

ในการออกแบบและพัฒนาโปรแกรมนี้ได้ใช้มาตรฐาน H.264 ในการออกแบบโปรแกรมซึ่งได้ใช้ Source Code อ้างอิงจากงานวิจัย "Implementation of a basic H.264/AVC Decoder" ของ Martin Fiedler ซึ่งได้ทำการศึกษาค้นคว้าการทำงานของมาตรฐานนี้ โดยได้ทำการพัฒนา Source Code มาจาก โปรแกรมตัวอย่างของมาตรฐาน H.264 อีกที คือ JM7.2 โดยโปรแกรมที่นำมาพัฒนานี้มีรายละเอียดการทำงานดังรูปด้านบนนี้

2.5.1 การรับอินพุท

จะเป็นการจัดการเกี่ยวกับ Input ที่รับเข้ามาโดยเริ่มจากการอ่านไฟล์อินพุท ซึ่งในที่นี้จะใช้ชื่อว่า test.264) โดยการอ่านไฟล์อินพุทนี้จะทำการอ่านทีละช่วงของบิท เนื่องจากเป็นรูปแบบของตัวเข้ารหัส ซึ่งตำแหน่งของบิทที่เรียงกันนี้จะมีความหมาย ดังนั้นการอ่านค่าจึงต้องกำหนดลำดับของตัวแปรที่ใช้รองรับอินพุทที่ถูกต้องด้วย โดยกระบวนการทำงานโดยรวมของการรับอินพุทคือ จะรับข้อมูลอินพุทจากไฟล์ทั้งหมดมาเก็บไว้ในบัฟเฟอร์ แต่จะทำการจัดแบ่งอินพุทเป็นบล็อคย่อยๆไว้ก่อน โดยจะทำการชิฟบัฟเฟอร์ไปยังบิทที่กำหนดไว้ที่เหมาะสมกับขนาดของบัฟเฟอร์นั้นๆ(ในที่นี้คือ 32 บิท) จากนั้นจะทำการแบ่งค่าที่รับมาออกเป็นบล็อกย่อยๆแบ่งอินพุทที่รับมาออกเป็นบล็อกละ 8 บิท จากนั้นจะสร้างตัวแปรมาตัวแปรหนึ่งเพื่อทำการอ้างอิงถึงตำแหน่งบิทที่รับมาโดยตัวแปรนี้จะมีขนาด 3 บิทเพื่อที่จะอ้างตำแหน่งได้ 8 ตำแหน่งของอินพุทที่รับมา($2^3 = 8$) จากนั้นเมื่อต้องใช้ข้อมูลจะทำการชิฟบิทไปยังที่ต้องการและรับข้อมูลมาจากบัฟเฟอร์

2.5.2 การกำหนดค่าการทำงานของโปรแกรมการเข้ารหัสของ H.264

เนื่องจากในส่วนของมาตรฐาน H.264 นี้มีการทำงานที่ผู้ใช้สามารถกำหนดค่าได้หลากหลาย ดังนั้นในส่วนของตัวเข้ารหัสจึงเป็นส่วนที่ทำการกำหนดค่าการทำงานต่างๆเพื่อให้ตรงตามความต้องการของผู้ใช้ และทำให้ในส่วนของตัวถอดรหัสจึงจำเป็นต้องมีส่วนของการแปลข้อมูลการทำงานให้ถูกต้องด้วย

ในส่วนของการกำหนดการทำงานจะมีการแยกออกเป็น 2 ส่วนคือการถอดรหัส Parameter และการถอดรหัส Slice Header โดยทั้งสองส่วนนี้จะเป็นตัวกำหนดฟังก์ชันการทำงานต่างๆ และ จะรับค่าอินพุทจากบัฟเฟอร์มาเพื่อทำการแปลว่าในฟังก์ชันส่วนนั้นๆจะมีการเปิดใช้งานหรือไม่ หลังจากที่ทำการแปลว่าตัวเข้ารหัสกำหนดไว้ว่าต้องการเรียกใช้ฟังก์ชันใดบ้างแล้ว ต้องทำการตรวจสอบอีกทีว่าฟังก์ชันที่ต้องใช้นั้นตัวถอดรหัสรองรับการทำงานหรือไม่ เนื่องจากในส่วนของตัวถอดรหัสที่สร้างขึ้นนี้ไม่สามารถถอดรหัสการทำงานในบางฟังก์ชันได้ เช่น long-term predition , B-Slide เป็นต้น

2.5.3 Entropy Coding

ในส่วนของ Entropy นั้นเปรียบเสมือนส่วนที่ทำการแปลข้อมูลส่วนเล็กๆ ซึ่งในโปรเจคนี้มาตรฐาน H.264 ที่ใช้นั้นใช้เป็น Baseline Profile ซึ่งใน Profile นี้การเข้ารหัส Entropy ที่ใช้จะเป็น CAVLC ซึ่งมีการใช้งานที่ไม่ซับซ้อนเท่า CABAC แต่ทำงานได้เร็วกว่า ซึ่งการถอดรหัส Entropy โดย CAVLC นี้จะใช้ตาราง Huffman เข้ามาช่วย ซึ่งในโปรแกรมจะเรียกว่า exp_golomb ซึ่งมีอยู่ 2 ประเภทคือ Signed_exp_golomb และ Unsigned_exp_golomb ซึ่งจะมีสูตรในการคำนวณต่างกันออกไป โดยในส่วนของ Unsigned_exp_golomb จะใช้สูตรการคำนวณคือ

$$2^n-1+V \qquad (2.1)$$

ในขณะที่ Signed_exp_golomb จะมีสูตรการคำนวณคือ

$$(C+1)/2 \qquad (2.2)$$

โดยในส่วนของโปรแกรมจะมีตาราง Code Table ซึ่งเป็นตาราง Huffman อยู่ด้วย ซึ่งเมื่อรวมการทำงานทั้งหมดและได้ค่า Entropy ออกมาเพื่อใช้ในการแปลข้อมูลต่อไป

2.5.3.1 Context-based Adaptive Binary Arithmetic Coding (CABAC)

เมื่อพารามิเตอร์ของรูปภาพกำหนดค่า flag entropy_coding_mode เป็น 1 ระบบ arithmetic coding ถูกใช้เข้ารหัสและถอดรหัส H.264 syntax elements Context-based Adaptive Binary Arithmetic Coding ประสิทธิภาพการบีบอัดที่ดีโดย (a) แบบจำลองความน่าจะเป็นการเลือกสำหรับแต่ละ syntax element ตาม context ของ element (b) ความน่าจะเป็น adapting ประมาณโดยมีพื้นฐานบน Local Static และ (c) การใช้ arithmetic coding มากกว่า variable-lenght coding การเข้ารหัสสัญลักษณ์ข้อมูลทำให้ต้องใช้ขั้นตอนดังต่อไปนี้:

1.Binarisation: CABAC ใช้ Binary Arithmetic Coding ที่มีความหมายว่าตัดสินใจแบบ Binary (1 หรือ 0) ถูกเข้ารหัส สัญลักษณ์ non-binary-valued (สัมประสิทธิ์การแปลง (transform coefficient) หรือMotion Vector สัญลักษณ์บางตัวมีค่าที่เป็นไปได้มากกว่าสองค่า) คือ binary หรือ แปลงเป็น binary code ก่อน arithmetic coding กระบวนการนี้เหมือนกับกระบวนการการแปลงสัญลักษณ์ข้อมูลไปเป็น variable length code แต่ binary code เป็นการเข้ารหัสต่อมา (โดยตัวเข้ารหัส arithmetic) ก่อนที่จะส่งและจะทำซ้ำสำหรับแต่ละ bit

2.การเลือกแบบจำลอง 'context model' คือแบบจำลองความน่าจะเป็นสำหรับหนึ่งหรือหลายๆ bins ของสัญลักษณ์ binary และถูกเลือกมาจากการเลือกแบบจำลองที่มีอยู่ขึ้นอยู่กับ สถิติของสัญลักษณ์ข้อมูล recently-coded แบบจำลอง context เก็บความน่าจะเป็นของแต่ละ bin เป็น '1' หรือ '0'

3.การเข้ารหัส arithmetic: ตัวเข้ารหัส arithmetic เข้ารหัสแต่ละ bin ตามแต่แบบจำลองความน่าจะเป็นถูกเลือก สังเกตว่าจะมีเพียงสอง sub-range สำหรับแต่ละ bin (ตรงกันกับ '0' และ '1')

4. Update ความน่าจะเป็น: แบบจำลอง context ที่ถูกเลือกจะถูก update อยู่บนพื้นฐานบนค่าของ coded จริง (ตัวอย่างเช่น ถ้าค่า bin เป็น '1' การนับความถี่ของ '1' จะเพิ่มขึ้น)

The Coding Process

การอธิบาย coding process จะใช้ตัวอย่างหนึ่งตัวอย่าง mvd_x (ผลต่างของเวกเตอร์ motion ในแนว x เข้ารหัสสำหรับแต่ละพาทิชั่นหรือพาทิชั่นสำหรับ sub-maroblockในInter-Macroblock)

1. Binrise ค่า mvdx . . . mvdx ตรงกับตารางข้างล่างนี้ของ uniquely-decodeable codewords สำหรับ | mvdx | < 9 (ค่าที่มากกว่าของ mvd_x ถูก binarised โดยใช้ Ex-Golomb codeword)

$\|mvd_x\|$	Binarisation (s=sign)
0	0
1	10s
2	110s
3	1110s
4	11110s
5	111110s
6	1111110s
7	11111110s
8	111111110s

ตารางที่ 2.2 แสดง mvd_x

bit ที่หนึ่งของ binarised codeword คือ bin 1 , bit ที่สองคือ bin 2 และอื่นๆต่อไปเรื่อยๆ

2. เลือกแบบจำลอง context สำหรับแต่ละ bin หนึ่งในสามของแบบจำลองถูกเลือกสำหรับ bin 1 พื้นฐานตามแบบ L1 ของค่า previously-coded mvd_x ek :

e_k	Context model for bin 1
$0 \le e_k < 3$	Model 0
$3 \le e_k < 33$	Model 1
$33 \le e_k$	Model 2

ตารางที่ 2.3 แสดง Context Modeของแต่ละ bin

$$E_k = | mvdx\ A | + | mvdxB | \tag{2.3}$$

ที่ซึ่ง A และ B คือ block ที่ติดกับด้านซ้ายและด้านบนของ block ปัจจุบัน

Bin	Context model
1	0, 1 or 2 depending on e_k
2	3
3	4
4	5
5 and higher	6
6	6

ตารางที่ 2.4 แสดง Bin และ Context Mode

ถ้า E_k มีค่าน้อยแล้วมีค่าความน่าจะเป็นค่ามาก MVD ปัจจุบันจะมีขนาดเล็ก และในทางตรงกันข้ามถ้า E_k มีค่ามากแล้วในทางเดียวกัน MVD ปัจจุบันจะมีขนาดใหญ่ด้วย ตารางความน่าจะเป็น (แบบจำลอง context) สำหรับ bin แรกจะถูกเลือกตามตาราง 6.16 ส่วน bin ที่เหลือจะถูก coded โดยใช้หนึ่งในสี่ของแบบจำลอง context ต่อไป

3. เข้ารหัสแต่ละ bin แบบจำลอง context ที่ถูกเลือกจะประมาณค่าความน่าจะเป็นได้สองค่า ความน่าจะเป็นที่ bin เก็บค่า '1' และความน่าจะเป็นที่ bin เก็บค่า '0' หาค่า sub-range สองค่าโดยการใช้ arithmetic coder เข้ารหัส bin

4. Update แบบจำลอง context ตัวอย่างเช่น ถ้าแบบจำลอง context 2 ถูกเลือกสำหรับ bin 1 และค่าของ bin 1 คือ '0' การนับจำนวนความถี่ของ '0' จะเพิ่มขึ้น ดังนั้นในครั้งต่อไปเมื่อแบบจำลองนี้ถูกเลือก ความน่าจะเป็นของ '0' จะค่อยๆสูงขึ้นทีละน้อย เมื่อจำนวนรวมของการเกิดแบบจำลองเกินค่า threshold การนับจำนวนความถี่สำหรับ '0' และ '1' จะลดลง ผลกระทบทำให้การกระทำเร็วๆนี้มีความสำคัญมากกว่า

The Context Models

แบบจำลอง context และ วิธี binarisation สำหรับแต่ละ syntax element ถูกกำหนดในมาตรฐาน มีแบบจำลอง context เกือบ 400 แบบสำหรับ syntax element ต่างๆ

ในส่วนแรกของแต่ละ code slice แบบจำลอง context ถูกกำหนดขึ้นโดยขึ้นกับค่าเริ่มของ Quantisation Parameter QP (ตั้งแต่มันมีผลกระทบกับความน่าจะเป็นของการมีอยู่ของสัญลักษณ์ข้อมูลต่างๆ) นอกจากนี้สำหรับ coded P,SP และ B slices ตัวเข้ารหัสอาจจะเลือกหนึ่งจากสามเซตของแบบจำลอง context พารามิเตอร์เริ่มต้นในส่วนเริ่มของแต่ละ slice อนุญาตให้มีการดัดแปลงชนิดของการเก็บวีดีโอเป็นแบบอื่นๆ

The Arithmetic Coding Engine

ตัวถอดรหัส arithmetic ถูกอธิบายในบางรายละเอียดในมาตรฐาน และมีคุณสมบัติที่แตกต่าง 3 ข้อ

1. การประมาณความน่าจะเป็นถูกกระทำโดยกระบวนเปลี่ยนระหว่างสถานะความน่าจะเป็น 64

สถานะ สำหรับ 'Least Probable Symbol' (LPS ความเป็นไปได้ที่น้อยที่สุดของการตัดสินใจแบบ binary '0' และ '1')

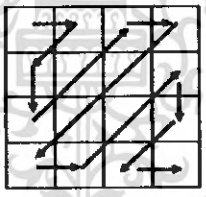
2. ระยะ R แสดงสถานะปัจจุบันของ arithmetic coder ถูกลดจำนวนเหลือระยะน้อยๆของค่า pre-set ก่อนการคำนวณระยะใหม่ที่แต่ละขั้นตอน ทำให้เป็นไปได้โดยคำนวณระยะใหม่โดยการใช้ตาราง look-up (multiplication-free)

3. การเข้ารหัสแบบง่ายและกระบวนการถอดรหัส (ในอันที่ผ่านส่วนของแบบจำลอง context ไป) ถูกกำหนดสำหรับสัญลักษณ์ข้อมูล กับการกระจายความน่าจะเป็น near-unifrom

2.5.3.2 Context-based Adaptive Variable Length Coding (CAVLC)

ใน H.264 นั้นได้มีการนำเอา CAVLC มาใช้เพื่อที่จะทำการเข้ารหัส Entropy ซึ่งเป็นการเข้ารหัส DCT นั่นเอง ขั้นตอนการทำ เริ่มแรก เมื่อเราทำการTransform และ Quantize เสร็จแล้ว เราจะได้ค่าสัมประสิทธิ์มา ซึ่งการที่เราจะเอาค่าที่ได้มาเข้ารหัสนั้น เราจะต้องทำการเอาค่าที่ได้มาเรียงกันก่อน โดยใน CAVLC นี้จะเอาค่าที่ได้มาเรียงกันแบบ Zigzag Scan นั่นคือการที่เริ่มที่มุมบนซ้ายก่อนจากนั้นจึงหาค่าถัดไปทางขวาหนึ่งช่องและหาค่าลงมาจากช่องแรกอีกหนึ่งช่อง จากนั้นจึงทำเช่นเดิมแต่เพิ่มจำนวนช่องถัดออกมาเรื่อยๆ ดังตัวอย่างภาพข้างล่างนี้

0	3	-1	0
0	-1	1	0
1	0	0	0
0	0	0	0



รูปที่ 2.13 แสดงการเรียงค่าแบบ Zigzag Scan

ซึ่งจะได้เป็น 0 3 0 1 –1 –1 0 1 0 ... 0

ต่อมาได้มีผู้สังเกตว่ารูปแบบการเรียงค่าแบบ Zigzag Scan ในการทำ CAVLC นี้มีรูปแบบที่น่าสังเกตว่ามีลักษณะซ้ำๆเดิมคือ

1. ค่าสัมประสิทธิ์ตัวท้ายสุดที่ไม่ใช่ค่า 0 มักจะเป็น 1 หรือ -1
2. บล็อคส่วนมากหลังจากทำ Quantize แล้ว มักจะมีค่าเป็น 1 หรือ -1
3. ค่าสัมบูรณ์ของสัมประสิทธิ์ที่มีค่ามากมักจะอยู่ใกล้ DC (ความถี่ต่ำ) ส่วนค่าสัมบูรณ์ของสัมประสิทธิ์ที่มีค่าน้อย มีจะเป็นค่าที่มีความถี่สูง

ขั้นตอนการเข้ารหัสหลังจากทำ Zigzag Scan เสร็จแล้ว จะมีขั้นตอนหลักๆทั้งหมด 5 ขั้นตอนด้วยกัน คือ

1. NumTrail ทำการเข้ารหัสสัมประสิทธิ์ที่ไม่ใช่ 0

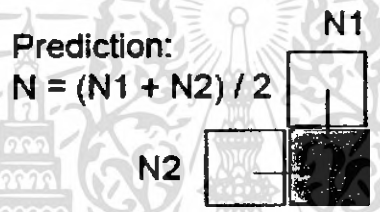
2. SignTrail ทำการเข้ารหัส sign (0 เป็น บวก , 1 เป็น ลบ)
3. Levels ทำการเข้ารหัสสัมประสิทธิ์ที่ไม่ใช่ 0 ตัวที่เหลือ
4. TotalZeros ทำการเข้ารหัสตัวที่เป็น 0 ทั้งหมด
5. Runs ทำการเรียงการกระจายของ 0 ใหม่

ขั้นตอนที่1 NumTrail

เป็นการเข้ารหัสค่าสัมประสิทธิ์ที่ไม่ใช่ 0 โดยการเข้ารหัสนั้นจะทำการเทียบกับตาราง Huffman โดยตาราง Huffman นั้นมีอยู่ด้วยกัน 3 ตาราง ซึ่งวิธีการเลือกว่าจะใช้ตารางไหนนั้นสามารถหาได้จากการหาค่าเฉลี่ยของค่าสัมประสิทธิ์ โดยการนำค่าสัมประสิทธิ์ของบล็อคที่อยู่ด้านซ้ายและบล็อคที่อยู่ด้านบนมาหาค่ากลาง ตามสมการ

$$N=(N1+N2)/2 \tag{2.4}$$

ดังรูปข้างล่างนี้



รูปที่ 2.14 แสดงการPredictหาค่าบล็อค

โดยมีหลักการเลือกคือ

1. ถ้า N มีค่ามากกว่าหรือเท่ากับ 0 แต่ไม่ถึง 2 ใช้ Table 0 (เหมาะกับสัมประสิทธิ์ที่มีค่าน้อย)
2. ถ้า N มีค่ามากกว่าหรือเท่ากับ 2 แต่ไม่ถึง 4 ใช้ Table 1(เหมาะกับสัมประสิทธิ์ที่มีค่ากลางๆ)
3. ถ้า N มีค่ามากกว่าหรือเท่ากับ 4 แต่ไม่ถึง 8 ใช้ Table 2(เหมาะกับสมประสิทธิ์ที่มีค่ามาก)
4. ถ้า N มีค่ามากกว่า 8 ขึ้นไป จะใช้ Fixed-Length code XXXXYY
    - XXXX คือ ค่าสัมประสิทธิ์
    - YY คือ ค่า T1

ตัวอย่าง Huffman Table สามารถดูได้จากรูปข้างล่างนี้

ตารางที่ 2.5 แสดงตัวอย่างตาราง Huffman

NumCoef\T1	0	1	2	3
0	1	-	-	-
1	000011	01	-	-
2	00000111	0001001	001	-
3	000001001	00000110	0001000	00011
4	000001000	000001011	000000101	000010
5	0000000111	000001010	000000100	0001011
...	...	...	...	...
16	0000000000000000	0000000000000001	0000000000000010	0000000000000010000

ตารางที่ 2.6 แสดงตัวอย่างตาราง Huffman

T1s NumCoeff	0	1	2	3
0	0011	-	-	-
1	0000011	0010	-	-
2	0000010	101110	1101	-
3	000011	101001	010110	1100
4	000010	101000	010001	1111
5	101101	101011	010000	1110
...	...	...	...	...
15	0000000010	00000000011	00000000010	00000000001
16	0000000000001	000000000001	0000000000001	00000000000000

ขั้นตอนที่ 2,3 Coding of Level Information

เนื่องจากขั้นตอนที่2 และ 3 นั้นมีความต่อเนื่องใกล้เคียงกัน ปกติจึงมักมีการรวมทั้ง 2 ขั้นตอนในการทำทีเดียวต่อเนื่องกันไปเลย โดยขั้นตอนนี้เป็นการเข้ารหัสค่าสัมประสิทธิ์ที่ไม่ใช่ 0 โดยมีการคิดเครื่องหมาย

เริ่มจากเมื่อได้ค่าที่เรียงแล้ว(จากการทำ Zigzag Scan) นั้นจะทำการเข้ารหัสโดยที่ว่า ค่าที่เป็นบวก จะใส่ 0 ส่วนค่าที่เป็น ลบ จะใส่เป็น 1 จากนั้นจึงทำการเข้ารหัสเลเวลที่เหลือในลำดับที่กลับกัน จากนั้นทำการเลือกตารางโดยเลเวลปัจจุบันจะมีผลต่อการตัดสินใจการใช้ ตาราง Huffman ในเลเวลถัดไป โดยมีหลักการว่า เริ่มแรกจะเริ่มจากVLC 0จากนั้นจะใช้ VLC ถัดไปถ้าเลเวลก่อนหน้านี้มีค่า Thresholdที่สูงกว่า และจะใช้ Golomb-Rice(N) ถ้าค่า N ที่ได้มีมากกว่าหลังจากที่ทำการเปลี่ยนแล้ว

รูปข้างล่างแสดงให้เห็นตัวอย่างตารางการเข้ารหัสเลเวล

ตารางที่ 2.7 แสดงตารางเข้ารหัสเลเวล 0 และ 1

■ Level VLC 0
(Unary code)

Level	Code
1	1
-1	01
2	001
-2	0001
3	00001
-3	000001
..	..
-7	00000000000001
±8 to ±15	000000000000001xxxx
±16 ->	000000000000001xxxxxx xxxxxx

Level VLC 1
(Golomb-Rice(2) Code)

Level	Code
1	10
-1	11
2	010
-2	011
3	0010
-3	0011
..	..
14	000000000000010
-14	000000000000011
±15 to ±22	000000000000001x xxx
±23 ->	000000000000001x xxxxxxxxxx

ตารางที่ 2.8 แสดงตารางเข้ารหัสเลเวล 2

Level VLC 2 (Golomb-Rice(4) Code)

Level	Code
1	100
-1	101
2	110
-2	111
3	0100
-3	0101
4	0110
-4	0111
5	00100
..	..
±37 ->	000000000000001xxxxxxxxxxxx

กรณีพิเศษ

- ถ้าหมายเลขของTrail น้อยกว่า 3 ค่าสัมประสิทธิ์ตัวถัดไปที่ไม่ใช่ 0 จะ มากกว่า 1 หรือ น้อยกว่า -1
- การเปลี่ยนเป็นเลเวลถัดไปสามารถช่วยลดความยาวบิทได้

  กรณีที่เลเวลถัดไปมากกว่า 0 จะทำการเข้ารหัสเหมือน Level-1

  กรณีที่เลเวลถัดไปน้อยกว่า 0 จะทำการเข้ารหัสเหมือน Level+1

เช่น สมมติให้ค่าที่ได้หลังจาก Zigzag Scan เป็น -2 4 3 -3 0 0 -1 0 ... 0

ให้ T = 1 แล้ว -3 จะถูกเข้ารหัสเหมือน -2 (-3 + 1 = -2) ซึ่งจะช่วยลดจำนวนบิทลง คือ

-3 ใน VLC0 คือ 000001 และ -2 ใน VLC0 คือ 0001 เพราะฉะนั้นจำนวนบิทจะลดลงไป 2 บิท

ขั้นตอนที่4 Coding of Total Zeros

ขั้นตอนนี้เป็นการเข้ารหัสสัมประสิทธิ์ทั้งหมดที่เป็น 0 ซึ่งจากการQuantizeจะได้ค่าทั้งหมด 16 ค่า โดยจำนวนสัมประสิทธิ์ที่เป็น 0 ทั้งหมดนั้นหาได้จากสมการ Max TotalZeros = 16-จำนวนสัมประสิทธิ์ที่ไม่ใช่0 และกรณีที่สัมประสิทธิ์จะเป็น 0 หรือไม่นั้นมี 17 กรณีคือ ในกรณีที่จำนวนค่าสัมประสิทธิ์ที่ไม่ใช่ 0 เป็น 16 นั้นจะได้ TotalZeros เป็น 0 ในขณะที่ถ้าสัมประสิทธิ์เป็น 0 อยู่แล้ว ไม่ต้องทำอะไร ในขณะที่กรณีอื่นๆที่เหลือนั้นจะมี 15 กรณี คือจำนวน สัมประสิทธิ์ที่ไม่ใช่ 0 มีค่าเท่ากับ 1 ถึง 15 ซึ่งในแต่ละกรณี ซึ่งจะมีการนำไปเทียบกับตาราง Huffman อีกทีหนึ่งนั่นเอง โดยที่ Huffman Table Size = (16 – จำนวนสัมประสิทธิ์ที่ไม่ใช่ 0) + 1

ตารางที่ 2.9 แสดงตารางTotal Zeros

Different Huffman tables

NumCoeff TotZeros	1	2	3	4	5	6	7
0	1	111	0010	111101	01000	101100	111000
1	011	101	1101	1110	01010	101101	111001
2	010	011	000	0110	01011	1010	11101
3	0011	001	010	1010	1110	001	1001
4	0010	000	1011	000	011	010	1111
5	00011	1000	1111	100	100	000	00
6	00010	0101	011	110	1111	110	01
7	000011	1001	100	1011	110	111	101
8	000010	1100	0011	010	101	100	110
9	0000011	01000	1110	001	001	011	100
10	0000010	11011	1010	0111	000	10111	-
11	00000001	11010	11000	1111	01001	-	-
12	00000000	0100010	110011	111100	-	-	-
13	00000011	01100111	110010	-	-	-	-
14	000000101	01100110	-	-	-	-	-
15	000000100	-	-	-	-	-	-

ตารางที่ 2.10 แสดงตารางTotal Zeros (ต่อ)

NumCoeff TotZeros	8	9	10	11	12	13	14	15
0	101000	111000	10000	11000	1000	100	00	0
1	101001	111001	10001	11001	1001	101	01	1
2	10101	11101	1001	1101	101	11	1	-
3	1011	1111	101	111	0	0	-	-
4	110	00	01	0	11	-	-	-
5	00	01	11	10		-	-	-
6	111	10	00	-	-	-	-	-
7	01	110	-	-	-	-	-	-
8	100	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-

ขั้นตอนที่5 Coding of Runs

จากขั้นตอนที่ผ่านมาทำให้ในตอนนี้เรารู้ NumCoeff , All Levels ,NumZeros แล้ว ซึ่งขั้นตอนต่อไปคือการ Reconstruct การเข้ารหัสเหล่านี้ซึ่งสิ่งที่เราจำเป็นต้องรู้อีกอย่างหนึ่งนั่นคือ ตำแหน่ง ของ 0 ทั้งหมดที่จะเอาไปใส่จะอยู่ตรงไหนบ้างก่อนที่จะเจอสัมประสิทธิ์ตัวสุดท้ายที่ไม่ใช่ 0 โดยจะใช้ตาราง Huffman สำหรับ Coding of Runs โดยมีวิธีการทำและตัวอย่างการทำต่อไปนี้

ตัวอย่าง ให้ลำดับสัมประสิทธิ์ที่ได้คือ 3 -2 1 **0 1 0 0 0** -1 0 ... 0

จะมี TotalZeros เป็น 4 และมี RunLeft(จำนวน 0 ที่ยังไม่ถูกเข้ารหัส)

การทำงานครั้งที่1 RunLeft = 4 , RunBefore(จำนวน 0 ที่ ตัวที่เราพิจารณาอยู่) = 3 ซึ่งจะมี 0 ที่เป็นไปได้ก่อนค่าสัมประสิทธิ์ที่ไม่ใช่ 0 คือ 0 , 1 , 2 , 3 , 4 ซึ่งทั้งหมด 5 ค่าจากนั้นจึงเอาไปเทียบกับตาราง Huffman โดยดูค่าที่เป็นไปได้กับค่า RunBefore ซึ่งในที่นี้ค่าที่ได้คือ 101(3 | 4) และค่า RunLeft ที่เหลือคือ 4 – 3 = 1

การทำงานครั้งที่2 RunLeft = 1 , RunBefore = 1 เพราะฉะนั้นค่าสัมประสิทธิ์ที่ไม่ใช่ 0จึงเป็นได้แค่ 0 , 1 เมื่อเทียบกับตาราง Huffman จะได้เป็น 0 (1 | 1) เพราะฉะนั้น ค่า RunLeft ที่เหลือคือ 1-1 = 0 ซึ่งถือเป็นการเสร็จสิ้นการทำงาน

ซึ่งรูปข้างล่างเป็นตัวอย่างของตาราง Huffman ในการทำ Coding Runs

ตารางที่ 2.11 แสดงตาราง Coding of Runs

Runs Left *Run Before*	1	2	3	4	5	6	>6
0	1	1	01	01	01	01	000
1	0	01	00	00	00	00	010
2	-	00	11	11	11	101	101
3	-	-	10	101	101	100	100
4	-	-	-	100	1001	111	111
5	-	-	-	-	1000	1101	110
6	-	-	-	-	-	1100	0011
7	-	-	-	-	-	-	0010
8		-	-	-	-	-	00011
9	-	-	-	-	-	-	00010
10	-	-	-	-	-	-	00001
11	-	-	-	-	-	-	0000011
12	-	-	-	-	-	-	0000010
13	-	-	-	-	-	-	0000001
14	-	-	-	-	-	-	00000001

2.5.4 Motion estimation and compensation

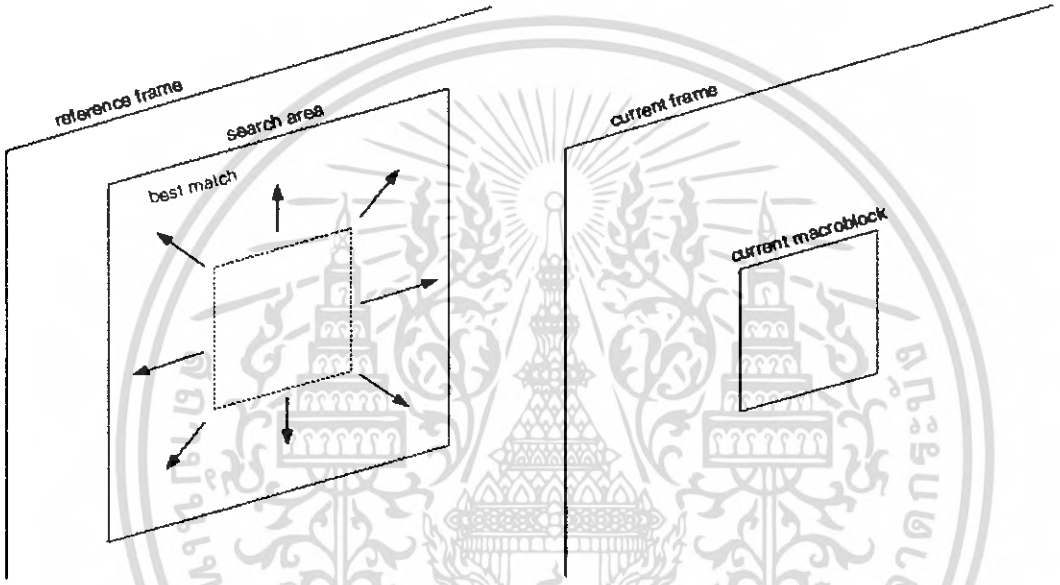
ในมาตรฐานการบีบอัดภาพแบบ H.264 นั้นได้มีการลดขนาดของ บิทเรท ลงไปมาก แต่ทั้งนี้คุณภาพของไฟล์ภาพที่ได้กลับลดลงจากเดิมน้อยมาก นั่นเป็นเพราะกระบวนการและขั้นตอนในการบีบอัดภาพที่มีประสิทธิภาพนั่นเอง โดยหลักที่ที่ใช้นั้นเป็นหลักการในการเข้ารหัสที่ดีและหลักในการชดเชยภาพเคลื่อนไหวที่ซ้ำๆกัน ซึ่งมีแนวคิดในการยึดเอาภาพๆหนึ่งเป็นภาพที่อ้างอิงไว้ แล้ว ให้ภาพในเฟรมต่อมาเทียบกับภาพที่อ้างอิง แล้วดูว่ามีส่วนไหนแตกต่าง และ ส่วนไหนที่เหมือนเดิมอย่างไรบ้าง

ขั้นแรกทำการหาความแตกต่างของภาพโดยทำการเปรียบเทียบภาพในเฟรมก่อนหน้านี้กับภาพในเฟรมปัจจุบันที่กำลังดูอยู่ ว่ามีส่วนต่างกันเท่าไหร่และอยู่ตรงส่วนไหน และส่วนที่ไม่ต่างจากเดิม (ส่วนประกอบเหมือนกับเฟรมก่อนหน้านี้) อยู่ที่ไหนบ้าง เช่นพื้นหลัง (Background) นั้นส่วนมากจะไม่มีการเปลี่ยนแปลงเท่าไหร่แล้วเลือกภาพที่เหมาะที่สุด (Best Match) โดยเลือกเอาภาพที่มีส่วนต่างน้อยที่สุดนั่นเอง วิธีการนี้เรียกว่า Motion Estimation

ต่อมาภาพที่ถูกเลือก(ภาพที่เทียบแล้วมีความแตกต่างและสามารถชดเชยได้มากสุด) ซึ่งจะเอามาเป็นตัวอ้างอิง ใช้เป็นตัวทำนายต่อไป จะเรียกว่าเป็นการทำ Motion Compensation นั่นเอง ในเฟรมผลต่างส่วนที่เหลือ (Risidual) นั้นจะถูกเข้ารหัสและส่ง ซึ่ง Offset ของตำแหน่งขอบเขตที่ใช้อ้างอิง (Motion Vector) นั้นจะถูกส่งออกไปเช่นเดียวกัน

ซึ่งต่อมา ตัวถอดรหัสที่ได้รับMotion Vector แล้วจะทำการสร้างขอบเขตที่ทำนาย (Prediction Region) ขึ้นมาใหม่ แล้วจะทำการถอดรหัสบล็อคที่เป็นผลต่างออกมาเพื่อจะเอามาทำการเพิ่มเข้าไปให้

กับบล็อคที่ต้องการ แล้วทำการReconstruct ขึ้นมาใหม่ ซึ่งจะได้ Block เดิมตามต้นแบบที่มี ซึ่งจากหลักการที่ใช้ข้างต้นนี้ เมื่อเอามาใช้เข้ารหัสจะทำให้ลดขนาดบิทเรทได้ เพราะไม่ต้องทำการเข้ารหัสภาพในทุกเฟรม แต่ถึงแม้จะดูดีและสามารถใช้ได้จริงก็ตาม แต่วิธีข้างดังกล่าวนี้ยังมีข้อเสียที่ไม่สามารถใช้ได้กับภาพที่มีลักษณะการเคลื่อนที่ที่แปลกจากปกติและซับซ้อน ยกตัวอย่างเช่น ในความเป็นจริงแล้วภาพไม่ได้มีการเคลื่อนที่ในกรอบที่เรียบแบบปกติ อย่างกรอบเป็นสีเหลี่ยมผืนผ้านั้น ตัววัตถุมักจะมีการเคลื่อนที่ด้วยจำนวนพิกเซลระหว่างเฟรมจำนวนมาก และจะต้องมีการชดเชยการเคลื่อนที่หลากหลายรูปแบบ ซึ่งได้แก่ การหมุนรอบ การวาร์ป หรือการเคลื่อนที่ที่ดูไม่ชัดเจนและซับซ้อน อย่างกลุ่มหมอกควันเป็นต้น

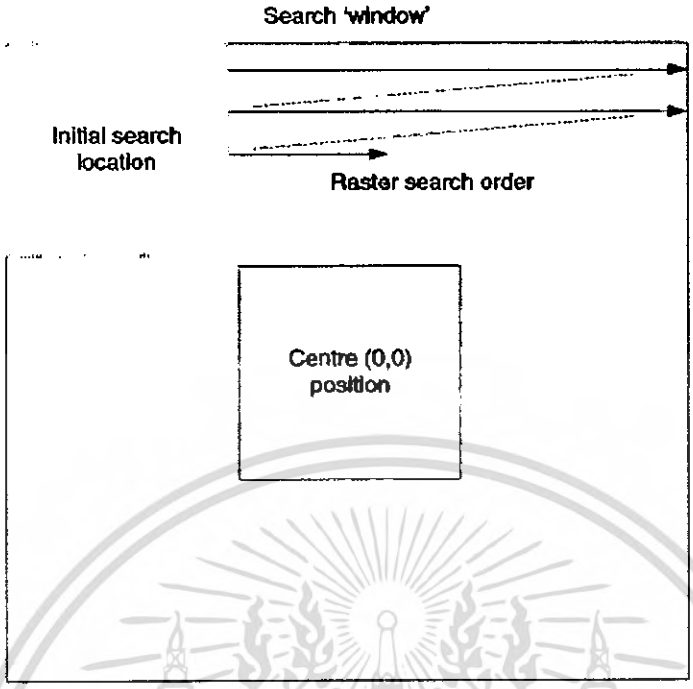


รูปที่ 2.15 แสดงการประมาณการเคลื่อนที่

2.5.4.1 Motion Estimation (Motion Search) Algorithm

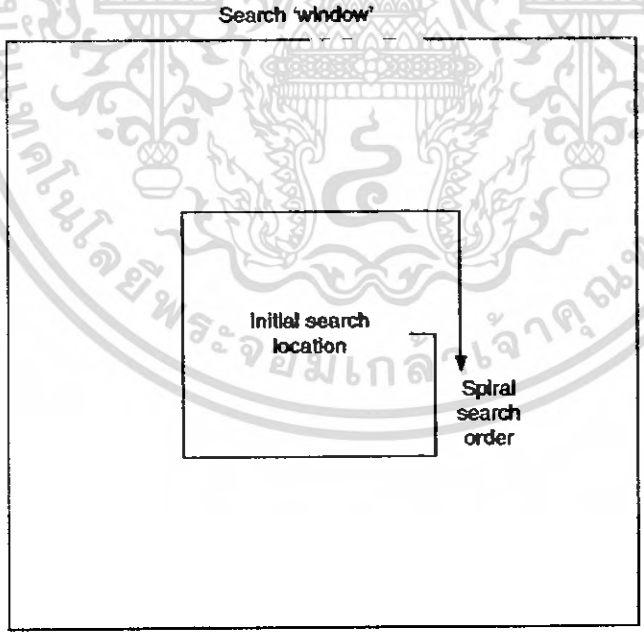
2.5.4.1.1 Full Search

Full Search เป็นการค้นหาค่า SAE ที่ต่ำที่สุด ซึ่งสามารถรับประกันได้ว่าค่าที่หาได้นั้นจะเป็น SAEที่ต่ำที่สุดจริงๆ เนื่องจากจะเป็นการหาจากทุกๆที่ที่สามารถเป็นไปได้ในSearch Window (หน้าต่างที่กำหนดขอบเขตการหาว่าค่าที่เป็นไปได้ที่จะเอามาใช้อ้างอิงนั้นจะอยู่ในขอบเขตไหน) ซึ่งการทำ Full Search นี้สามารถทำได้2วิธีด้วยกัน คือการทำFull Search แบบ Raster Scan กับแบบ Spiral Scan



รูปที่ 2.16 แสดงการทำ Full Search แบบ Raster Scan

จากรูปเป็น Full Search แบบ Raster Scanซึ่งจะเป็นการเริ่มที่จุดบนซ้ายที่สุดก่อนแล้วค่อสแกนไปทางขวาจนสุดกรอบจึงจะเริ่มต้นที่ซ้ายสุดของแถวถัดไป



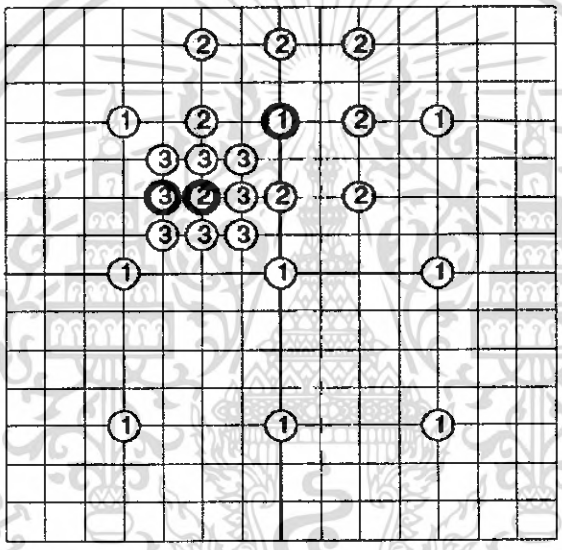
รูปที่ 2.17 แสดงการทำ Full Search แบบ Spiral Scan

จากรูปเป็นการทำ Full Search แบบ Spiral Scan ซึ่งจะเป็นการเริ่มค้นหาจากตรงกลางก่อนแล้วจึงค่อยๆ วนขยายออกไปรอบข้างข้างนอกเรื่อยๆตามเข็มนาฬิกา

อย่างไรก็ตามไม่ว่าจะเป็นวิธีไหนก็สามารถหาค่าต่ำสุดได้จริงเพียงแต่จะมีข้อเสียคือเสียเวลาในการค้นหานานมากและยังเป็นการเพิ่มความซับซ้อนในการค้นหาเพราะต้องค้นหาหลายรอบซึ่งเป็นการผิดจุดประสงค์ที่ต้องการลดความซับซ้อนในการใช้ค่าส่วนต่างเอามาช่วยคิด

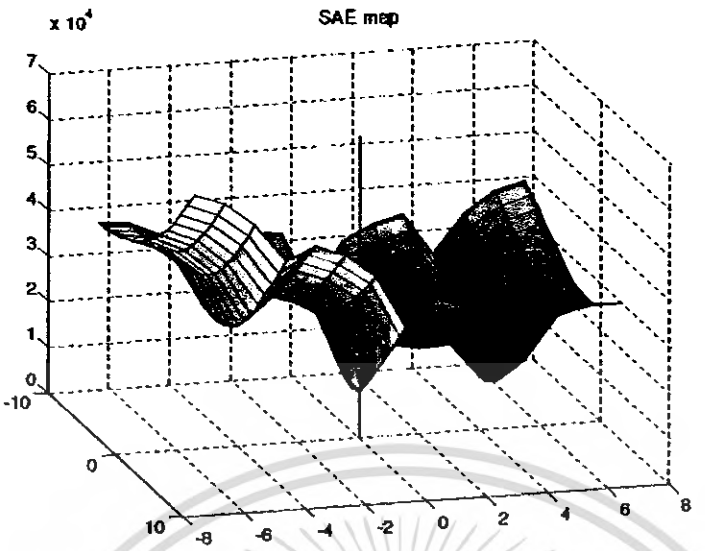
2.5.4.1.2 Fast Search

การทำงานคร่าวๆของ Fast Search คือจะเป็นการคำนวณ (มักจะใช้ SAE) โดยคิดเพียงแค่ส่วนย่อยๆของSearch Window เท่านั้น โดยปกติวิธีการค้นหาแบบรวดเร็วนี้มักจะใช้วิธีการ **Three Step Search** (แต่บางครั้งเรียกว่า N-Step Search)



รูปที่ 2.18 แสดงการทำ Fast Search แบบ Three-Step Search

จากรูปสามารถอธิบายให้เข้าใจวิธีการได้ง่ายๆนั่นคือ เริ่มจากตรงกลางสุดกำหนดให้เป็นหมายเลข 1 ต่อมาไล่ไปทางซ้าย, ขวา, บน, ล่างและแนวทแยงโดยนับไป4ช่อง จะได้ว่ามีจุดที่ถูกกำหนดด้วยหมายเลข 1 เป็น9 ช่องจากนั้นค่อยมาไล่ดูว่า ใน 9 ช่องนั้นช่องใดได้ค่าที่ต่ำสุด (ปกติใช้SAE) โดยเลือกจุดนั้นเป็นจุดศูนย์กลางใหม่ แล้วกระจายไปรอบๆทั้ง8ทิศเช่นเดิมแต่คราวนี้ระยะห่างเอาแค่2ช่องพอ แล้วกำหนดจุดรอบๆนั้นทั้ง8จุดให้กลายเป็นเลข 2 แล้วทำเช่นเดิมนั่นคือหาค่าที่คำนวณแล้วต่ำที่สุดเลือกมาเป็นศูนย์กลางใหม่อีกอันหนึ่ง แล้วทำการกระจายเช่นเดิมทั้ง8ทิศ แล้วกำหนดให้แต่ละจุดนั้นเป็นหมายเลข 3 โดยคราวนี้ไม่ต้องเว้นระยะห่างเลย แล้วเมื่อหาค่าที่ต่ำที่สุดในกลุ่มหมายเลข 3 ได้แล้วค่านั้นจะเป็นค่าที่ต่ำที่สุดใน Search Window นั้นๆ



รูปที่ 2.19 แสดงผลการหาค่าMinimal

ซึ่งเราจะเห็นว่ากระบวนการและจำนวนครั้งในการค้นหานั้นลดลงไปมากทีเดียวซึ่งถือว่าได้ผลดีมาก เนื่องจากไม่มีความซับซ้อนในการคำนวณมากเท่ากับ Full Search แต่อย่างไรก็ตามวิธีการนี้ต้องแลกมาด้วยการยอมว่าค่าที่ได้นั้นอาจจะไม่ใช่ค่าที่ดีที่สุด เนื่องจากเป็นการสมมติฐานว่ากลุ่มที่น้อยที่สุดจะอยู่ในกลุ่มที่น้อยที่สุดอีกที แต่ความจริง กลุ่มที่น้อยที่สุดที่ Three Step Search หาได้นั้นอาจจะไม่ใช่ค่าที่น้อยที่สุดก็เป็นได้ ในขณะที่Full Search นั้นถึงแม้ว่าจะมีการค้นหาที่ช้ากว่าและทำงานหนักกว่าแต่ค่าที่ได้นั้นเป็นค่าที่น้อยที่สุดอย่างแน่นอน ซึ่งการที่เราจะเลือกใช้แบบไหนนั้นเราจำเป็นที่จะต้องดูให้ดีว่าวิธีใดมีส่วนได้ส่วนเสียมากกว่ากันอย่างไร

ในความจริงแล้วอัลกอริทึม ที่ใช้ใน Fast Search นี้ยังมีอีกมาก อย่างเช่น Logarithmic Search , Hierarchical Search , Cross Search , and One at a Time Search ซึ่งแต่ละวิธีการนั้นมีวิธีและจุดเด่นจุดด้อยต่างกันออกไป ซึ่งโดยรวมแล้วมีประสิทธิภาพที่ใกล้เคียงกับ Fast Search ดังนั้นการทำ Fast Search นั้นจึงไม่เป็นที่นิยมเท่าไหร่

2.5.4.1.3 การหา Motion Vector ที่เหมาะสมกันมากที่สุด

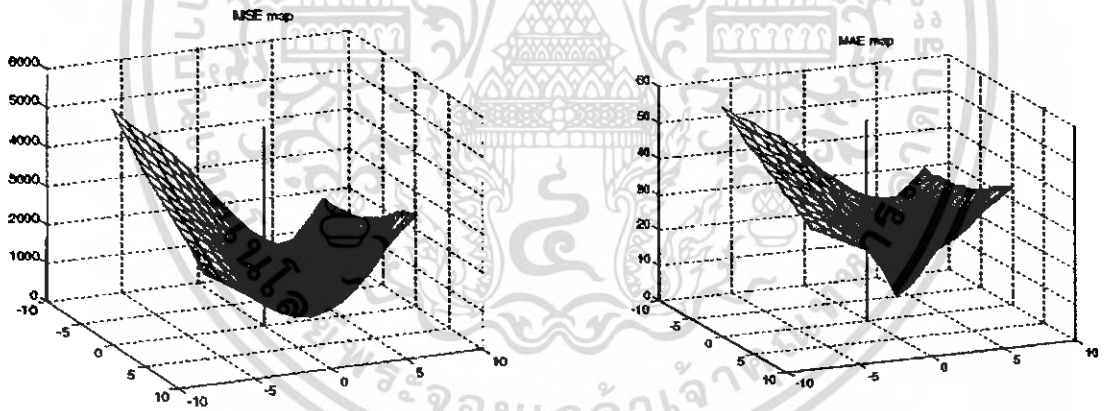
ทางเลือกที่ใช้คิดว่าจะเลือกใช้อันไหนดีนั้น จะดูจากว่าบล็อคผลต่างที่จะเอามานั้นมีความซับซ้อนในการที่จะคำนวณมากน้อยแค่ไหน และมีความแม่นยำในการประมาณการเคลื่อนที่มากน้อยแค่ไหน โดยจะมีวิธีในการวัดทั้งหมด3วิธีนั่นคือ MSE , MAE , SAE โดยในแต่ละวิธีจะมีการคำนวณดังนี้

1. Mean Squared Error: $$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(C_{ij} - R_{ij})^2 \quad (2.5)$$

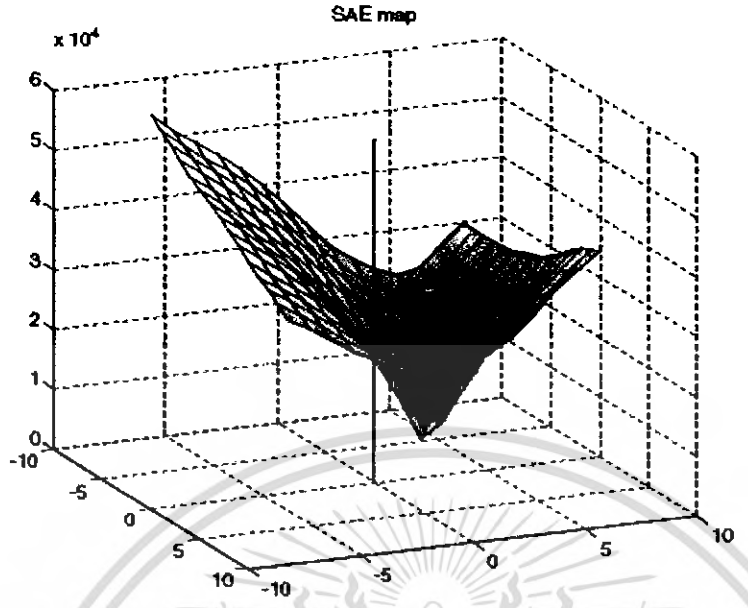
2. Mean Absolute Error: $$MAE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|C_{ij} - R_{ij}| \quad (2.6)$$

3. Sum of Absolute Errors: $$SAE = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|C_{ij} - R_{ij}| \quad (2.7)$$

ซึ่งโดยทั่วไปแล้ว คนส่วนมากมักจะใช้วิธี SAE (Sum of Absolute Errors) มากกว่าในการวัดหาว่าจะต้องประมาณผลต่างเท่าไหร่ซึ่งค่าที่ได้จากการคำนวณนั้นจะเป็นตัวบอก โดยที่ค่าที่น้อยที่สุดของ SAE นั้นจะเป็นค่าที่เหมาะสมที่สุดซึ่งจะมีความซับซ้อน และใช้พลังงานในการทำงานน้อยที่สุด (พลังงานในที่นี้หมายถึงความยุ่งยากในการคิดคำนวณครั้งต่อๆไป และความซับซ้อนในการที่ต้องอ้างถึงและเอาไปใช้คำนวณ) ในความจริงแล้วเราไม่จำเป็นจะต้องคำนวณทุกๆoffsetเพราะเมื่อเราคำนวณไปถึงจุดๆหนึ่งแล้วถ้าค่าที่ได้นั้นมากกว่าค่าของก่อนหน้านี้ เราจะเลิกคำนวณส่วนนั้นได้เลย เพราะการรวมค่าจะมากขึ้นเรื่อยๆต่อให้ยังคำนวณไม่เสร็จแต่ถ้าเกินค่าก่อนหน้านี้แล้วจึงไม่จำเป็นจะต้องคำนวณoffsetนั้นให้เสร็จอีก สามารถที่จะคำนวณส่วนต่อไปได้เลย จะทำให้การคำนวณนั้นใช้เวลาสั้นลงพอสมควร



รูปที่ 2.20 แสดงผลจากการคำนวณแบบ MSE และ MAE ตามลำดับ



รูปที่ 2.21 แสดงผลลัพธ์ในการหาคำตอบจากวิธีSAE ซึ่งเป็นวิธีที่นิยมที่สุด

2.5.4.2 Motion Compensated Prediction of Macroblock

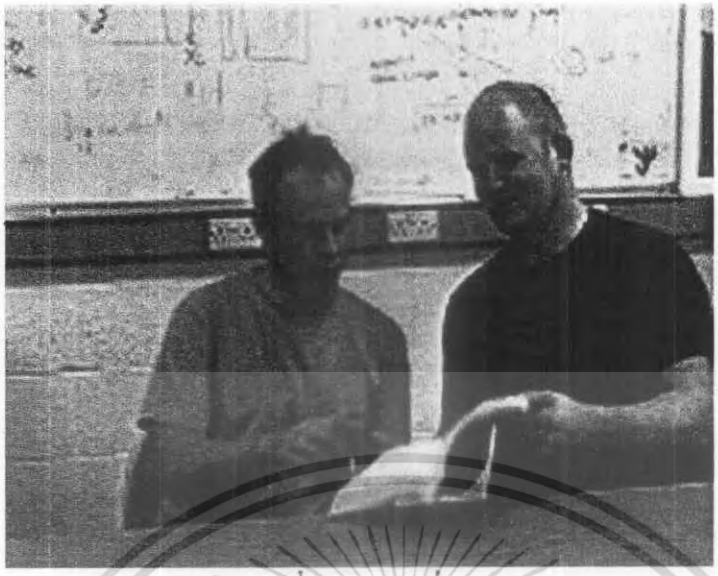
ปกติแล้วรูปแบบที่พบบ่อยๆสำหรับการทำการชดเชยการเคลื่อนที่คือ 16*16-pixel region frame ซึ่งจะมีการทำต่อไปนี้

การประมาณการเคลื่อนที่(Motion Estimation)

การประมาณการเคลื่อนที่ของมาโครบล็อคจะอ้างถึงเฟรมที่เหมาะและอยู่ใกล้กับเฟรมปัจจุบันมากที่สุดซึ่งเป็นไปได้ว่าจะอยู่ก่อหน้านี้หรือหลังจากนี้ซึ่งวิธีการหาคือเอาเฟรมปัจจุบันเป็นศูนย์กลางแล้วจึงค่อยเลือกมาโครบล็อคที่อยู่รอบๆเฟรมนั้นที่เหมาะกันมากที่สุด (Best Match)

การชดเชยการเคลื่อนที่(Motion Compensation)

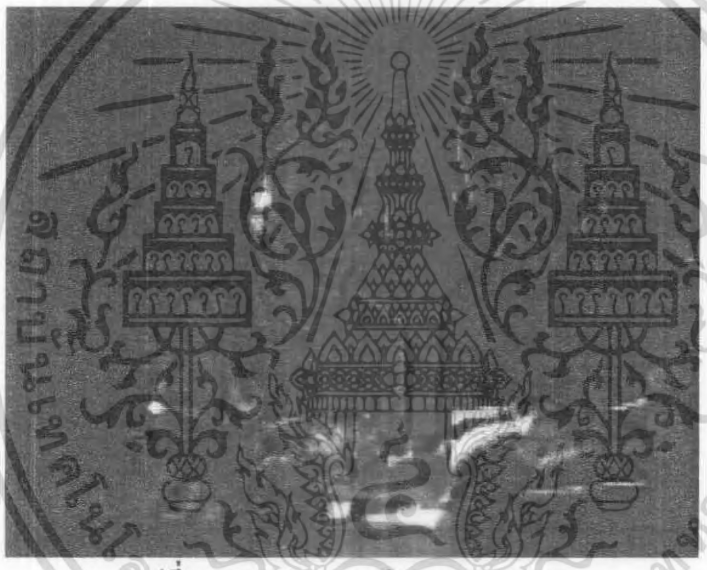
เฟรมที่ถูกเลือกนั้นจะถูกลบออกจากเฟรมปัจจุบัน ซึ่งจะกลายเป็นส่วนต่างของภาพปัจจุบัน ซึ่งจะเรียกว่า Residual macroblock ซึ่งจะถูกเข้ารหัสและถูกส่งไปพร้อมกับ Motion Vector ที่ใช้อธิบายตำแหน่งของขอบเขตที่เหมาะที่สุดนั่นเองซึ่งหลังจากนั้นตัวถอดรหัสจะทำการRoconstructมาโครบล็อคนั้นๆใหม่อีกครั้งนั่นเอง ซึ่งที่จริงแล้วการที่ไม่ทำ การชดเชยการเคลื่อนที่ในบางกรณีจะให้คุณภาพเฟรมที่ดีกว่า ซึ่งตัวเข้ารหัสจะเลือกใช้ Intra Mode เป็นการเลือกที่จะไม่ใช้การชดเชยการเคลื่อนที่ ในถ้าเลือกใช้ Inter Mode จะเป็นการเลือกที่จะใช้การชดเชยการเคลื่อนที่นั่นเอง



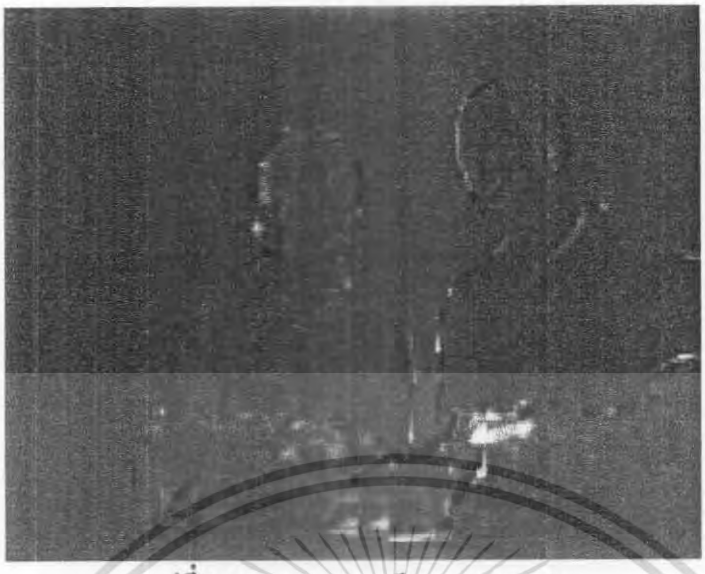
รูปที่ 2.22 เฟรมที่ 1

รูปที่ 2.23 เฟรมที่ 2

รูปที่ 2.24 Residual (แบบที่ไม่ได้ทำการชดเชยการเคลื่อนที่)



รูปที่ 2.25 Residual (บล็อกขนาด 16*16)



รูปที่ 2.26 Residual (บล็อคขนาด 8*8)

จากรูปเป็นตัวอย่างการเปรียบเทียบภาพที่ใช้การชดเชยการเคลื่อนที่และการใช้ Residual ขนาดต่างกัน โดยภาพแรกเป็นภาพอ้างอิง และภาพ2เป็นภาพปัจจุบัน ภาพที่3เป็นภาพที่ไม่มีการการชดเชยการเคลื่อนที่ ซึ่งเราจะเห็นว่าคุณภาพของผลต่างของภาพที่ได้จะไม่ค่อยดีนัก ในขณะที่ภาพที่4และ5 เป็นการเปรียบเทียบให้เห็นว่าถ้าใช้ขนาดบล็อคผลต่างที่มีขนาดไม่เท่ากัน คุณภาพของภาพที่ได้จะต่างกันด้วย ซึ่งจากภาพจะเห็นว่าภาพที่ใช้บล็อคขนาด16*16นั้นคุณภาพในการได้Residualจะไม่ดีเท่าการใช้บล็อคขนาด8*8 ซึ่งจะเห็นได้ว่ามีความละเอียดที่ดีกว่า แต่นั่นจะแลกมากับการทำงานที่จะซับซ้อนมากกว่า เพราะว่าจะมี Motion Vectorที่ต้องถูกส่งมากขึ้นตามไปด้วย ซึ่งการที่จะเลือกใช้บล็อคขนาดใดนั้นเป็นสิ่งที่จะต้องศึกษาให้ดี ในอดีตขนาดของบล็อคนี้จะไม่มีการเปลี่ยนแปลง คือใช้บล็อคขนาดไหนจะใช้ขนาดนั้นต่อไป แต่สำหรับมาตรฐานใหม่คือ H.264 แล้วขนาดของบล็อคสามารถเปลี่ยนแปลงได้ โดยภาพที่มีความละเอียดไม่มาก มีพื้นที่กว้างและไม่เปลี่ยนแปลงจะใช้ขนาดบล็อคใหญ่ๆ แต่สำหรับส่วนที่ต้องการความละเอียดมากๆและมีการเคลื่อนที่ที่ค่อนข้างซับซ้อน จะใช้ขนาดบล็อคขนาดเล็กเพื่อให้มีความละเอียดในการหาผลต่างที่สูงเพื่อที่ภาพที่ได้จะได้มีการชดเชยการเคลื่อนที่ที่ดีและจะส่งผลถึงคุณภาพของภาพที่ดีอีกด้วย

2.6 Discrete cosine transform (DCT)

DCT (Discrete Cosine Transform) เป็นอัลกอริทึมที่ใช้กันอย่างแพร่หลายในการทำ Image/Video Compression เพื่อที่จะเอาไปใช้ทำ Quantization ต่อไป โดยDCT นั้นมีวิธีการทำดังนี้ เริ่มจากเมื่อได้ค่าต่างๆในการทำในมาโครบล็อคต่างๆแล้วจะนำค่าเหล่านั้นมาใช้คำนวณซึ่งวิธี DCT นี้จะมีสมการการคำนวณดังต่อไปนี้

$$\mathbf{Y} = \mathbf{AXA}^{\mathrm{T}} \tag{2.8}$$

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos \frac{(2j+1)y\pi}{2N} \cos \frac{(2i+1)x\pi}{2N} \tag{2.9}$$

กำหนดให้ X เป็น เมตริกซ์ ของ Samples Y เป็น เมตริกซ์ของสัมประสิทธิ์ และ A เป็น N*N Transform matrix ซึ่งเราจะได้ค่า A เป็นดังนี้

$$A_{ij} = C_i \cos \frac{(2j+1)i\pi}{2N} \quad \text{โดยที่ } C_i = \sqrt{\frac{1}{N}} \ (i=0), \qquad C_i = \sqrt{\frac{2}{N}} \ (i>0) \tag{2.10}$$

ดังนั้นการทำ Inverse DCT (ซึ่งจะเป็นการทำ Inverse Transform นั่นเอง) เพียงแค่กลับค่า X,Y ซึ่งจะได้สูตรการคำนวณออกมาเป็นดังนี้

$$\mathbf{X} = \mathbf{A}^{\mathrm{T}}\mathbf{YA} \tag{2.11}$$

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos \frac{(2j+1)y\pi}{2N} \cos \frac{(2i+1)x\pi}{2N} \tag{2.12}$$

2.7 Transform and Quantization

การทำ Transform นั้นมีจุดประสงค์เพื่อที่จะเปลี่ยนภาพหรือผลต่างการเคลื่อนไหวของข้อมูลให้ไปอยู่อีกรูปแบบโดเมนหนึ่ง (ในที่นี้คือ Transform Domain) ในการทำ Transform นั้นจะมีอยู่2 ประเภท คือแบบ Block-base และ Image-based ซึ่งการทำแบบ Block-based นั้นวิธีที่ได้รับความนิยมมากที่สุดคือ DCT (Discrete Cosine Transform) ซึ่งมีข้อดีคือจะเป็นการใช้หน่วยความจำที่น้อยมากและเหมาะที่จะเอามาทำการชดเชยการเคลื่อนที่ แต่ข้อเสียคือตรงขอบจะไม่ค่อยดีเท่าไหร่ ในขณะที่ Image-based จะเป็นการทำทั้งภาพ ซึ่งวิธีที่นิยมใช้กันมากที่สุดคือDWT(Discrete Wavelet Transform) ผลที่ได้ออกมานั้นจะมีคุณภาพที่ดีกว่าเนื่องจากเป็นการทำบนภาพทั้งภาพแต่มีข้อเสียที่ใช้หน่วยความจำมากและไม่ค่อยเหมาะที่จะเอามาใช้กับผลต่างการชดเชยการเคลื่อนที่

ตัวอย่างการทำ4*4 Transform

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \quad (2.13)$$

โดยที่ $a = \frac{1}{2}, \quad b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right), \quad c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$

ให้ CXC^T เป็น Core สำหรับการทำTransform แบบ2มิติ Eเป็นเมตริกของ Scaling Factor โดยจะเอาค่า E นี้ไปคูณสำหรับทุกๆค่าใน CXC^T ซึ่งท้ายที่สุดแล้วเมื่อคำนวณเสร็จจะได้ค่า Y เป็นดังต่อไปนี้

$$Y = (CXC^T) \otimes E = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \quad (2.14)$$

Quantization

เป็นการแมพให้สัญญาณที่เป็นช่วงของค่า X เป็นสัญญาณquantize ที่ลดค่าของช่วงค่า Y ซึ่งจะแบ่งออกเป็น 2 ประเภทนั่นคือ Scalar Quantize และ Vector Quantize โดยที่ Scalar Quantize จะเป็นการแมพสัญญาณอินพุทอันเดียวเป็นค่าเอาท์พุทอันเดียว ขณะที่ Vector Quantize จะเป็นการแมพสัญญาณอินพุทแบบกลุ่มเป็นค่าเอาท์พุทแบบกลุ่ม โดยวิธีการคิด Quantize โดยเบื้องต้นนั้นจะสามารถคำนวณได้จากสูตร

$$Z_{ij} = round(Y_{ij} / Qstep) \quad (2.15)$$

โดยที่ Y_{ij} เป็นสัมประสิทธิ์ของการ Transform Qstepเป็นขนาดของ stepที่ใช้ทำ quantize และ Z_{ij} เป็นสัมประสิทธิ์การ Quantize นั่นเอง โดยปกติแล้วค่า Qstep นั้นจะมีขนาดเพิ่มขึ้นเป็น2เท่า ทุกๆQPที่เพิ่มขึ้น6 โดยที่ QP คือ Quantization Parameter สามารถดูค่าได้จากตารางข้างล่างนี้

ตารางที่ 2.12 แสดงค่า QP และ Qstep

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	...
QStep	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2	2.25	2.5	...
QP	...	18	...	24	...	30	...	36	...	42	...	48	...	51
QStep		5		10		20		40		80		160		224

ต่อมาเป็นการคิด Combining Real number Scaling to Quantization

$$Z_{ij} = round\left(W_{ij} \cdot \frac{PF}{Qstep}\right) \quad (2.16)$$

โดยที่ W_{ij} คือ CXC^T นั่นเอง ส่วน PF นั้นสามารถหาได้จากตารางข้าล่างนี้ โดยค่า PF จะขึ้นกับตำแหน่ง i , j นั่นเอง

ตารางที่ 2.13 แสดงการหาค่า PF

Position	*PF*
(0,0), (2,0), (0,2) or (2,2)	a^2
(1,1), (1,3), (3,1) or (3,3)	$b^2/4$
other	ab/2

ขณะที่การทำ Inverse Quantization นั้นจะเรียกว่าเป็นการ Rescale นั่นเอง โดยจะมีวิธีการ Rescaleเบื้องต้นคือ

$$Y'_{ij} = Z_{ij} Qstep \quad \text{และ} \quad W'_{ij} = Z_{ij} Qstep \cdot PF \cdot 64 \tag{2.17}$$

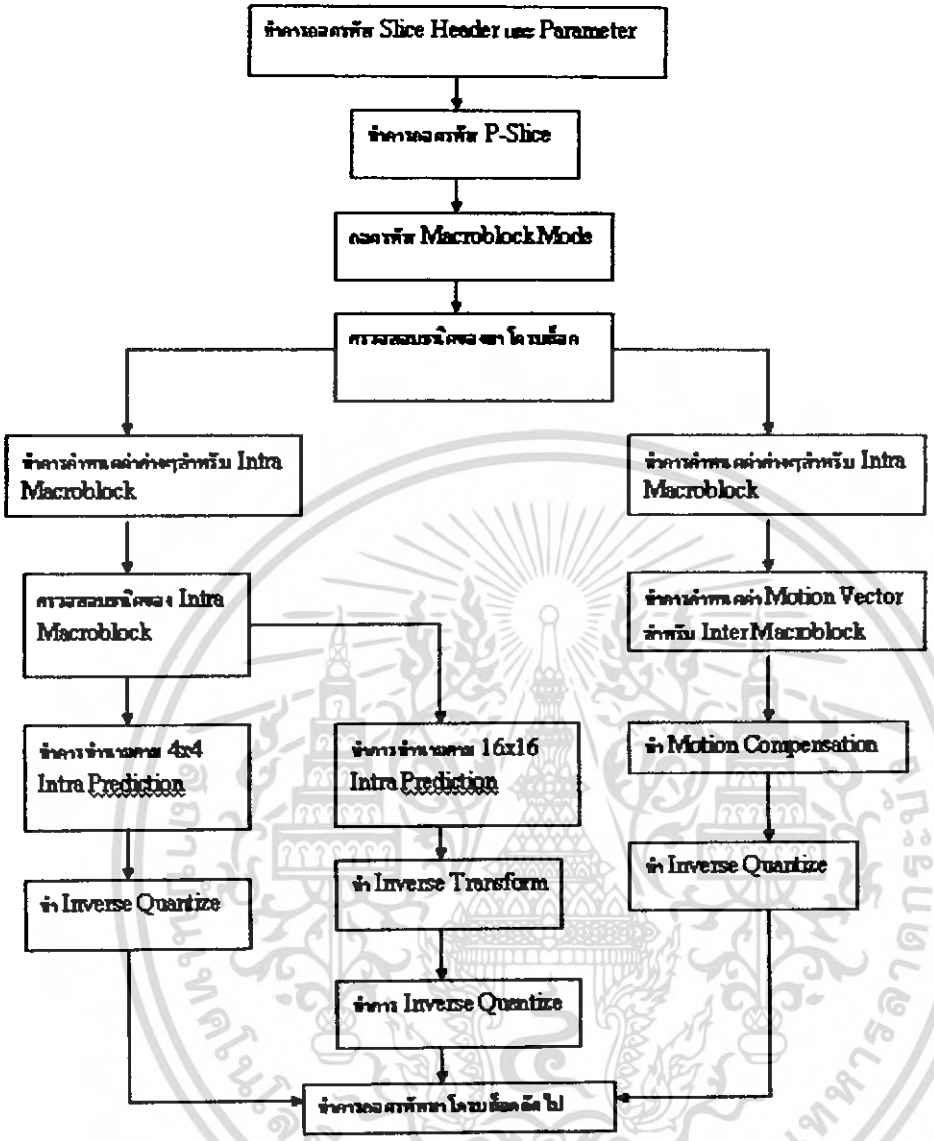
2.8 การทำงานของโปรแกรมการบีบอัดสัญญาณวิดีโอตามมาตรฐาน H.264

ในการออกแบบและพัฒนาโปรแกรมนี้ได้ใช้มาตรฐาน H.264 ในการออกแบบโปรแกรมซึ่งได้ใช้ Source Code อ้างอิงจากงานวิจัย "Implementation of a basic H.264/AVC Decoder" ของ Martin Fiedler ซึ่งได้ทำการศึกษาค้นคว้าการทำงานของมาตรฐานนี้ โดยได้ทำการพัฒนา Source Code มาจากโปรแกรมตัวอย่างของมาตรฐาน H.264 อีกทีซึ่งคือ JM7.2 โดยโปรแกรมที่นำมาพัฒนานี้มีรายละเอียดการทำงานดังนี้

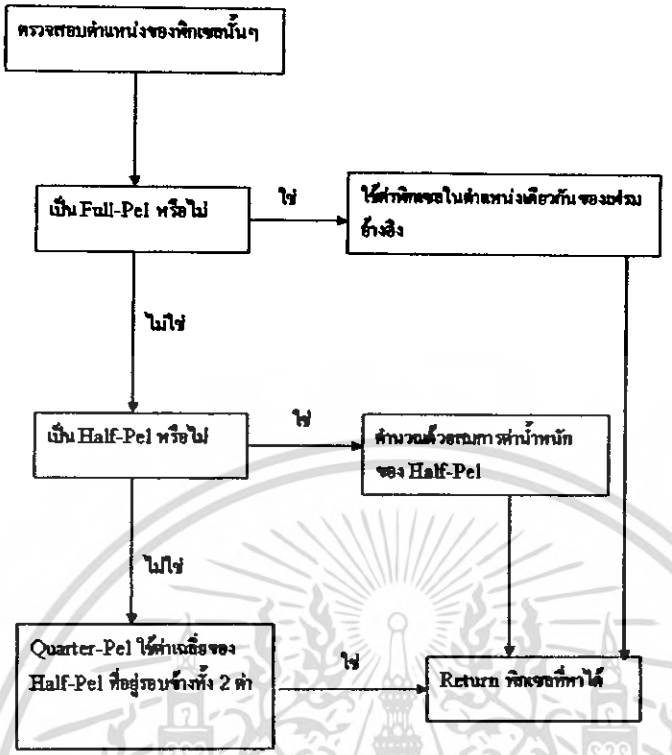
เมื่อได้ข้อมูลที่จำเป็นสำหรับการถอดรหัสแล้วจะเริ่มเข้าสู่กระบวนการการถอดรหัสจริงๆ ซึ่งการทำงานหลักๆของโปรแกรมจะอยู่ในฟังก์ชัน decode_slice_data ซึ่งเป็นฟังก์ชันที่จะทำการเรียกใช้งานฟังก์ชันย่อยๆที่จำเป็นในการถอดรหัสทั้งหมด โดยจะเริ่มจากการตั้งค่าเริ่มต้นที่จำเป็นให้กับตัวแปรต่างๆ เช่นตัวแปร mpi(มาจากฟังก์ชัน mode_predict_info ซึ่งทำหน้าที่ในการเก็บค่าของข้อมูลต่างๆของ slice ที่จำเป็นในการทำ prediction) จะทำการเรียก clear_mode_pred_info ซึ่งเป็นฟังก์ชันที่ใช้ในการเคลียร์ค่าต่างๆของ mpi ให้กลับไปเป็นค่าตั้งต้นใหม่ เนื่องจากทำการถอดรหัสใน Slice ใหม่แล้ว หลังจากนั้นจะทำการตรวจสอบว่าข้อมูลที่เหลืออยู่ยังมีอีกหรือไม่ ถ้ามีจะเริ่มทำการถอดรหัสโดยในส่วนแรกของการถอดรหัสนี้จะทำการตรวจสอบmb_skip_run() ซึ่งเป็นการทำงานของ P-Slice แต่ถ้าในช่วงเริ่มแรกของการถอดรหัสจะเป็น I-Slice จะข้ามในส่วนนี้ไป แต่ถ้าเป็น Slice ต่อๆมาเมื่อเป็น P-Slice จะเข้าฟังก์ชันนี้ซึ่งเป็นการเรียกใช้ฟังก์ชันต่างๆที่จำเป็นในการถอดรหัสของ P-Slice เริ่มจากจะทำการกำหนดค่าของ Mode Predict Info ซึ่งเป็นการกำหนดค่า Mode ของมาโครบล็อก ให้เป็นค่าที่เหมาะ

สม หลังจากนั้นจะเริ่มทำการหาค่า Motion Vector ซึ่งเป็นกระบวนการหนึ่งซึ่งสำคัญมากในการถอดรหัส ซึ่งกระบวนการนี้จะเป็นการชี้ไปยังตำแหน่งของบล็อกที่อ้างอิง เพื่อที่จะได้อ้างถึงเมื่อต้องการทำการ Prediction ซึ่งจะมีฟังก์ชันการทำงานอยู่หลายฟังก์ชันด้วยกัน โดยการทำงานหลักๆจะอยู่ที่ PredictMV ในฟังก์ชันนี้จะทำการทำนายค่า Motion Vector โดยดูจาก Motion Vector ที่อยู่รอบข้างซึ่งคือ Motion Vector จากด้านบน ด้านซ้ายและด้านขวาบน และถ้าไม่สามารถหา Motion Vector ด้านขวาบนได้จะใช้ Motion Vector ด้านซ้ายบนแทน แต่ถ้าไม่สามารถหา Motion Vector ของสองด้านใดๆที่อยู่รอบข้างได้จะใช้ Motion Vector ของด้านที่เหลือไปเลย แต่ถ้าสามารถหาค่าได้ครบทั้งสามด้านจะนำทั้งสามค่ามาหาค่ากลาง แต่ถ้า Motion Vector ชี้ไปยังนอกรูปซึ่งไม่สามารถหาค่าได้ หรือ ถ้า Motion Vector ที่จะคำนวณนั้นเป็นเป็น mode 0 Motion Vector นั้นจะมีค่าเป็นศูนย์ ต่อมาในส่วนของฟังก์ชัน FillMV จะเป็นฟังก์ชันที่ทำการเติมค่า Motion Vector ที่หาได้ใส่ในบล็อกย่อยๆทุกบล็อกในบล็อกขนาด 4x4 นั่นเอง

หลังจากทำการคำนวณ Motion Vector เสร็จแล้วจะเริ่มทำการคำนวณหา Motion Compensation ซึ่งถือว่าเป็นกระบวนการที่มีการทำงานมากที่สุดเนื่องมาจากการทำงานที่ค่อนข้างซับซ้อนและใช้จำนวนรอบและระยะเวลาที่ใช้ในการคำนวณมาก การทำงานของฟังก์ชัน Motion Compensation นี้จะเริ่มจากการสร้าง Temp Block ขึ้นมาเพื่อทำการเก็บค่าพิกเซลของเฟรมอ้างอิง ณ ตำแหน่งเดียวกัน โดยจะมี Temp Block อยู่ 2 ประเภทคือ Luma Temp block ที่ใช้เก็บค่าพิกเซลประเภท Luma ของเฟรมอ้างอิงซึ่งจะมีขนาด 9x9 และ Chroma Temp Block ที่ใช้เก็บค่าพิกเซลประเภท Chromaของเฟรมอ้างอิงซึ่งจะมีขนาด 3x3 หลังจากที่ทำการสร้าง Temp Block แล้วก็จะทำการคำนวณหาค่าพิกเซล โดยเริ่มจากการคำนวณค่าพิกเซลของ Luma 4x4 ก่อน ซึ่งค่าพิกเซลแต่ละตำแหน่งก็จะมีการคำนวณที่แตกต่างกันไป แต่ก่อนอื่นจะต้องทำการตรวจสอบความถูกต้องของค่าพิกเซลที่จะมาคำนวณก่อน เนื่องจากค่าพิกเซลที่ถูกต้องจะต้องเป็นค่าที่ไม่ต่ำกว่า 0 และจะต้องมีค่าไม่เกิน 255 โดยถ้าค่าพิกเซลผิดไปจากที่กำหนดก็จะทำการกำหนดค่าให้อยู่ในกรอบที่ถูกต้องแทนโดยถ้าค่าพิกเซลต่ำกว่า 0 ก็จะทำการเปลี่ยนค่านั้นให้กลายเป็น 0 แทน และถ้าค่าพิกเซลนั้นมีค่าเกิน 255 ก็จะทำการเปลี่ยนค่าให้กลายเป็น 255 แทน หลังจากตรวจสอบแล้วก็จะเริ่มทำการคำนวณค่าพิกเซล ณ ตำแหน่งต่างๆโดยจะมีวิธีการคำนวณที่แตกต่างกันไป



รูปที่2.27 แสดงการทำงานโดยรวมของโปรแกรม



รูปที่2.28 แสดงการทำงานของการคำนวณหาค่าพิกเซลใน Motion Compensation

1. Full Pixel จะเป็นค่าพิกเซลที่มีตำแหน่งที่อยู่ในพิกัด p(0,0) b(2,0) h(0,2) j(2,2) ซึ่งเป็นตำแหน่งที่ตรงตามเฟรมอ้างอิงและไม่ต้องทำการคำนวณ
2. Half Pixel จะเป็นค่าพิกเซลที่มีตำแหน่งที่อยู่ในพิกัดที่อยู่ระหว่าง Full Pixel ซึ่งเป็นตำแหน่งที่ไม่ตรงกับเฟรมอ้างอิง และการคำนวณหาค่าพิกเซลนี้จะทำการคำนวณหาค่าพิกเซลที่อยู่รอบข้างเอามาเป็นค่าน้ำหนัก (Weight) เอามาคำนวณตามสมการค่าน้ำหนักตามสูตร

$$(E)-5*(F)+20*(G)+20*(H)-5*(I)+(J)+16 \qquad (2.18)$$

จะได้ค่า Half Pixel มา

3. Quarter Pixel จะเป็นค่าพิกเซลที่มีตำแหน่งที่อยู่ในพิกัดระหว่าง Half Pixel อีกที ซึ่งตามความจริงแล้วควรจะคำนวณค่าน้ำหนักเช่นเดียวกันกับการคำนวณ Half Pixel แต่เนื่องจาก Quarter Pixel เป็นการคำนวณในส่วนที่ย่อยลงมา และถ้ามีการคำนวณค่าน้ำหนักอีกจะเป็นการกระทำที่ซับซ้อนและเสียเวลาเกินไป ดังนั้นการคำนวณค่า Quarter Pixel จึงเป็นแค่การหาค่าเฉลี่ยตรงกลางระหว่างค่า Half Pixel เท่านั้น

และเนื่องจากการที่ค่าพิกเซลที่อยู่ในพิกัดต่างๆใช้การคำนวณที่แตกต่างกันไป ดังนั้นจึงต้องมีการตรวจสอบตำแหน่งพิกัดของพิกเซลก่อนการคำนวณทุกครั้ง ซึ่งเป็นสาเหตุที่ทำให้เกินความล่าช้าและ เสียเวลาในการคำนวณมาก แต่จะทำให้ค่าที่คำนวณเกิดความถูกต้องมากขึ้นด้วย

ต่อมาหลังจากทำการคำนวณค่าพิกเซลของ Luma เรียบร้อยแล้วจะทำการคำนวณในส่วนของ Chroma ต่อ ซึ่งจะทำการคำนวณในส่วนของ Chroma Pixel ซึ่งโดยรวมการทำงานคล้ายๆกับการคำนวณในส่วนของ Luma Pixel คือทำการสร้าง Temp Block และโหลดค่าพิกเซลเข้ามา แต่จะต่างกันตรงขนาดจะเป็น 3x3 เท่านั้น และจะเริ่มทำการคำนวณค่าพิกเซลซึ่งการคำนวณนี้จะทำการคำนวณโดยคิดค่าพิกเซลเพียงแค่ 2x2 เท่านั้นและสมการที่ใช้จะเปลี่ยนจากสมการค่าน้ำหนักที่ใช้ใน Luma Pixel เป็นสมการ Bilinear แทน

หลังจากทำการคำนวณในส่วนของ Slice แล้วจะลงมาทำการคำนวณในส่วนของมาโครบล็อก ซึ่งจะทำการตรวจสอบว่ามาโครบล็อกที่จะทำการถอดรหัสนั้นๆ เป็นมาโครบล็อกชนิดใด ซึ่งถ้าเป็นมาโครบล็อกประเภท I_PCM จะไม่ทำการคำนวณอะไรและส่งมาโครบล็อกนั้นๆไปเลย ซึ่งมาโครบล็อกประเภทนี้ผู้ที่เข้ารหัสมักจะไม่ใช้กันมากนัก แต่อาจจะใช้ในบางกรณีเช่นในกรณีที่เกิด Error ในการคำนวณอย่างมากเกินกว่าที่จะข้ามไปได้ หรือเมื่อคำนวณแล้วค่ามีความผิดพลาดเกินไปและไม่มีความใกล้เคียงกับค่ารอบข้างเลย จะทำการเปลี่ยนมาใช้มาโครบล็อกประเภทนี้เพื่อที่จะได้ส่งข้อมูลมาโครบล็อกนี้ผ่านไปทันทีโดยไม่ต้องเข้ารหัส และเมื่อตัวถอดรหัสได้รับจะไม่ต้องทำการคำนวณอะไร แต่จะรับค่าของมาโครบล็อกนั้นๆมาตรงๆเลย

แต่ถ้ามาโครบล็อกนั้นไม่ใช่ I_PCM จะมีอยู่ 2 ประเภทนั่นคือ Inter Macroblock และ Intra Macroblock โดยในส่วนของ Inter Macroblock จะทำการถอดรหัสโหมดของมาโครบล็อกก่อน หลังจากนั้นจะทำการหาค่า Motion Vector ของมาโครบล็อกนั้นๆ ต่อมาในส่วนของ Intra Macroblack จะมีอยู่ 2 ประเภทคือ Intra 4x4 และ Intra 6x6 ซึ่งจะทำการ Predict มาโครบล็อกตามโหมดของมาโครบล็อก นั้นๆ

ต่อมาจะเป็นการทำงานในส่วนของ Residual ซึ่งจะทำการกำหนด pattern ของบล็อกเพื่อเป็นการระบุเลเวลของบล็อก เมื่อได้ข้อมูลที่จำเป็นสำหรับการถอดรหัสระดับมาโครบล็อกเรียบร้อยแล้วจะเข้าสู่กระบวนการในการถอดรหัสโดยจะแบ่งการถอดรหัสมาโครบล็อกออกเป็น 2 ประเภทเช่นเดียวกับ Slice นั่นคือมี Inter Macroblock และ Intra Macroblock ในขณะที่ Intra Macroblock นั้นจะแบ่งย่อยได้อีก 2 ประเภทคือ Intra4x4 และ Intra16x16 โดยเริ่มจาก Intra4x4 นั้นเริ่มแรกจะทำการสแกนแบบ Raster Scan เมื่อรับค่าที่ได้จากการสแกนเรียบร้อยแล้วจะนำค่าที่ได้มาทำการ Prediction โดยการทำนายของ Intra4x4นี้มีทั้งหมด 9 โหมดการทำนายซึ่งมาโครบล็อกนั้นๆใช้โหมดใดในการถอดรหัสต้องขึ้นอยู่กับว่ามาโครบล็อกนั้นๆถูกเข้ารหัสมาแบบใด ซึ่งโดยปกติแล้ว default ของโหมดการทำนายจะเป็น Intra_4x4_DC ซึ่งเป็นการหาค่ากลาง หลังจากที่ทำการทำนายเรียบร้อยแล้ว จะทำการ Inverse Quantize และต่อมาจะทำการทำนายในส่วนของIntra Chroma 4x4 ซึ่งจะมีโหมดในการทำนายอยู่ 4 โหมด

ต่อมาในส่วนของ Intra16x16 ก็จะทำงานเช่นเดียวกันกับในส่วนของ Intra4x4 นั่นคือจะทำการทำนายมาโครบล็อกเช่นเดียวกับ Intra4x4 เพียงแต่โหมดที่ใช้ทำนายจะมีเพียงแค่ 4 โหมดเท่านั้น หลังจากนั้นจะทำการ Inverse Transform โดยจะมีรูปแบบการสแกนเป็นแบบ Zig-Zag Scan Order ซึ่งรูปแบบการทำ Inverse Transform นี้จะใช้วิธี Hadamard Transform ซึ่งจะได้ค่า Inverse Transform ออกมาเสร็จแล้วจะทำการ Inverse Quantize อีกทีหนึ่ง เมื่อทำการถอดรหัสการทำนายในส่วนของ Luma เสร็จแล้วจะทำในส่วนของ Chroma อีกทีเช่นเดียวกับ Intra4x4 แล้วจะเสร็จในส่วนของการทำ Intra Macroblock ต่อมาถ้ามาโครบล็อกนั้นๆเป็นมาโครบล็อกประเภท Inter Macroblock จะมีขั้นตอนแตกต่างไปจาก Intra Macroblock ทั้ง 2 ประเภท โดยInter Macroblock จะเริ่มการทำงานจากการคำนวณ Motion Compensation จากนั้นจึงทำการInverse Quantize เสร็จแล้วจึงค่อยเริ่มทำการทำนายมาโครบล็อก หลังจากนั้นจะเสร็จสิ้นกระบวนการทำนายมาโครบล็อกในส่วนของ Luma Residual ซึ่งหลังจากนี้จะเป็นกระบวนการสุดท้ายของการถอดรหัสใน Slice หนึ่งๆ นั่นคือการถอดรหัส Chroma Residual ซึ่งไม่มีการทำงานอะไรมากมายเหมือน Luma Residual นั่นคือทำการ Inverse Quantize และ Inverse Transform สำหรับ Chroma หลังจากนั้นจะเสร็จสิ้นกระบวนการถอดรหัสของ Slice หนึ่งๆ

หลังจากนี้โปรแกรมจะกลับเข้าไปใน Main Loop เพื่อทำการถอดรหัส Slice ต่อไป เมื่อครบทุก Slice แล้ว จะทำการบันทึกค่าพิกเซลทั้งหมดลงบนไฟล์เอาท์พุท แล้วจะทำการปิดไฟล์อินพุท จากนั้นจะสิ้นสุดกระบวนการทั้งหมดในการถอดรหัสภาพเคลื่อนไหวตามมาตรฐาน H.264

2.9 บทสรุปและเปรียบเทียบการทำงานของมาตรฐาน H.264 กับมาตรฐานอื่นๆ

มาตรฐาน H.264 นี้มาความสามารถในการบีบอัดและเข้ารหัสได้ดีซึ่งสามารถลดจำนวนบิทเรทลงได้มากเมื่อเทียบกับมาตรฐานอื่นๆ ทั้งคุณภาพหลังจากผ่านกระบวนการแล้วมีคุณภาพที่ดีและใกล้เคียงกับต้นฉบับมาก ซึ่งไม่เพียงแต่มาตรฐานอื่นเท่านั้น ทั้งในมาตรฐานเดียวกัน MPEG-4 part 10 ถือว่ามีคุณภาพสูงกว่ารุ่นก่อนหน้านี้เป็นอย่างมาก โดยจะสามารถเปรียบเทียบมาตรฐาน MPEG-4 part 10 กับมาตรฐาน MPEG-4 เวอร์ชันก่อนหน้านี้ได้ตามตารางเปรียบเทียบข้างล่างนี้

ตารางที่ 2.14 แสดงการเปรียบเทียบมาตรฐาน MPEG ในเวอร์ชันต่างๆ

Feature/Standard	MPEG-2	MPEG-4 part 2	MPEG-4 part 10 /H.264
Macroblock size	16x16 (frame mode) 16x8 (field mode)	16x16	16x16
Block size	8x8	16x16, 16x8, 8x8	16x16, 8x16, 16x8, 8x8, 4x8, 8x4, 4x4
Intra prediction	No	Transform Domain	Spatial Domain
Transform	8x8 DCT	8x8 DCT/Wavelet transform	8x8, 4x4 integer DCT 4x4, 2x2 Hadamard
Quantization	Scalar quantization with step size of constant increment	Vector quantization	Scalar quantization with step size of increase at the rate of 12.5%
Entropy coding	VLC	VLC	VLC, CAVLC, CABAC
Pel accuracy	½-pel	¼-pel	¼-pel
Reference picture	One picture	One picture	Multiple pictures
Bidirectional prediction mode	forward / backward	forward / backward	forward / backward forward / forward backward / backward
Weighted prediction	No	No	Yes
Deblocking filter	No	No	Yes
Picture types	I, P, B	I, P, B	I, P, B, SI, SP
Profiles	5 profiles	8 profiles	7 profiles
Playback & Random access	Yes	Yes	Yes
Error robustness	Data partitioning, FEC for important packet transmission	Synchronization, Data partitioning, Header extension, Reversible VLCs	Data partitioning, Parameter setting, Flexible macroblock ordering, Redundant slice, SP and SI slices
Transmission rate	2-15Mbps	64kbps - 2Mbps	64kbps - 150Mbps
Encoder complexity	Medium	Medium	High
Compatibility with previous standards	Yes	Yes	No

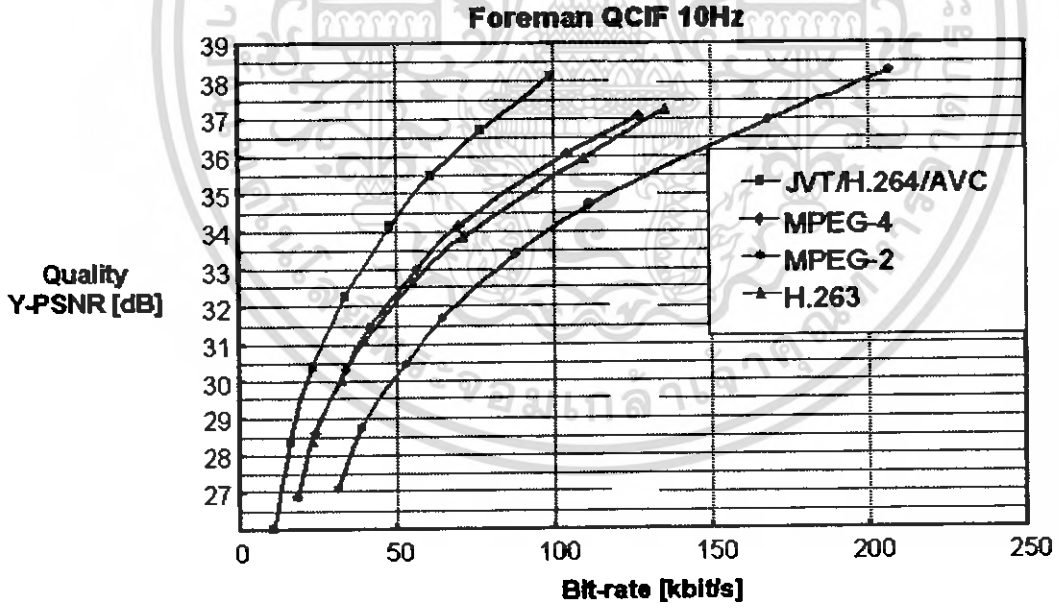
จะเห็นว่าส่วนประกอบต่างๆและขั้นตอนการทำงานของ MPEG-4 part 10 นั้นมีความยืดหยุ่นให้มีการกำหนดเองมากกว่า และขั้นตอนการทำงานนั้นมีมากกว่า สามารถทำได้ละเอียดทำให้ประสิทธิภาพในการเข้ารหัสและผลลัพธ์ที่ออกมาจึงดีกว่าด้วย

นอกจากจะดีกว่า MPEG รุ่นก่อนๆแล้วแล้ว แม้แต่มาตรฐานอื่น MPEG-4 part 10 นี้ยังมีขั้นตอนที่หลากหลายและมีส่วนประกอบที่ค่อนข้างจะละเอียดกว่ามากด้วยดังเช่นตารางเปรียบเทียบมาตรฐานอื่นๆ (WMV-9 และ AVS) ข้างล่างนี้

ตารางที่ 2.15 แสดงการเปรียบเทียบมาตรฐาน MPEG-4 part 10 กับ มาตรฐานอื่นๆ

Feature/Standard	MPEG-4 part 10 /H.264	WMV-9	AVS
Prediction block size	16x16, 8x16, 16x8, 8x8, 4x8, 8x4, 4x4	16x16, 8x8	16x16, 8x8
Intra prediction	4x4, 8x8 : 9 modes 16x16 : 4 modes	No	8x8 : 5 modes
Transform	8x8, 4x4 integer DCT 4x4, 2x2 Hadamard	8x8, 8x4, 4x8, 4x4 integer DCT	Asymmetric 8x8 integer DCT
Quantization	Scalar quantization	Dead zone, uniform scalar quantization	Scalar quantization
Entropy coding	VLC, CAVLC, CABAC	Multiple VLC tables	VLC
Sub-pel filter	1/2-pel : 6-tap 1/4-pel : 2-tap	1/2-pel : 4-tap 1/4-pel : 4-tap	1/2-pel : 4-tap 1/4-pel : 4-tap
Reference picture	Multiple pictures	One picture	Two pictures
Bidirectional prediction mode	forward / backward forward / forward backward / backward 2 motion vectors	forward / backward 2 motion vectors	forward / backward symmetric 1 motion vector
Weighted prediction	Yes	Yes	Yes
Deblocking filter	Yes	Yes	Yes

ซึ่งถ้าเราทำการเปรียบเทียบ มาตรฐาน MPEG-4 , JVT/H.264/AVC, MPEG-2 และ H.263 ปกติสิ่งที่เราคำนึงถึงในการเปรียบเทียบคือคุณภาพการบีบอัด และ จำนวนบิทเรท นั่นเอง ซึ่งถ้าเราเอาทั้ง4มาตรฐานมาเปรียบเทียบกันจะได้ดังกราฟต่อไปนี้



รูปที่ 2.29 เทียบคุณภาพกับบิทเรทในมาตรฐานต่างๆ

แต่ถึงอย่างไรก็ตามการทำงานที่ซับซ้อนเพื่อให้ได้มาซึ่งคุณภาพของการบีบอัดภาพเคลื่อนไหวที่สมบูรณ์แบบต้องแลกมาด้วยการทำงานที่ซับซ้อนตามมาด้วย ซึ่งการทำงานที่ซับซ้อนนี้ได้ส่งผลอย่างมากถึงประสิทธิภาพในการประมวลผลข้อมูลซึ่งทำให้เกิดการประมวลผลที่ล่าช้า นอกจากการ

ทำงานบน DSP จะช้าแล้ว แม้แต่การทำงานบนเครื่อง PC ปกติยังไม่สามารถทำให้เกิดการประมวลผลที่รวดเร็วได้ ทำได้แค่การประมวลผลด้วยความเร็วปานกลางคือประมาณ 12 เฟรมต่อวินาทีสำหรับภาพขนาด 640 x 480 ในขณะที่การทำงานบน Blackfin นั้นมีการทำงานที่ช้ายิ่งกว่าบน PC คือในภาพขนาด 320 x 240 นั้นจะมีความเร็วในการประมวลผลเพียงแค่ 0.9 เฟรมต่อวินาทีเท่านั้น ซึ่งแม้จะมีการทำการออปติไมเซชันเพียงใดก็ไม่สามารถทำให้มีความเร็วในการประมวลผลเพิ่มขึ้นได้อีก โดยคาดว่าอาจจะต้องมีการเปลี่ยนโครงสร้างการทำงานของตัวโปรแกรมมาตรฐานเพื่อให้มีการทำงานที่รวดเร็วกว่านี้

บทที่ 3

การออกแบบและพัฒนา

3.1 ขั้นตอนการออกแบบพัฒนา

ในส่วนของการทำการพัฒนาโปรแกรม H.264 นี้เริ่มจากการศึกษาถึงการทำงานของมาตรฐาน H.264 ว่าทำงานอย่างไร จากนั้นทำการหาตัวอย่างโปรแกรมที่เหมาะสมและตรงตามมาตรฐาน H.264 มาเพื่อทำการศึกษาและพัฒนาโปรแกรมต่อ จากนั้นทำการลงโปรแกรม H.264 ลงบน Visual DSP ของ Blackfin และทำการตรวจสอบผลของการทำงานของโปรแกรมบน Blackfin นี้ หลังจากทำการตรวจสอบแล้ว จะพบปัญหาการทำงานต่างๆในการโปรแกรมลงบน Blackfin DSP จะทำการตรวจสอบและหาสาเหตุของปัญหาการทำงานต่างๆ ไม่ว่าจะเป็นปัญหาในการประมวลผล หรือ ปัญหาทางด้านประสิทธิภาพต่างๆ เมื่อพบปัญหาและสาเหตุแล้วจึงทำการศึกษาผลงานวิจัยของต่างประเทศว่าในส่วนของปัญหาต่างๆที่พบนี้มีการแก้ปัญหาอย่างไร จากนั้นจึงทำการดัดแปลงวิธีการเพื่อให้เข้ากันได้กับโปรแกรมที่พัฒนา และทำการตรวจสอบผลการทำงานและประสิทธิภาพในการทำงานต่อไป

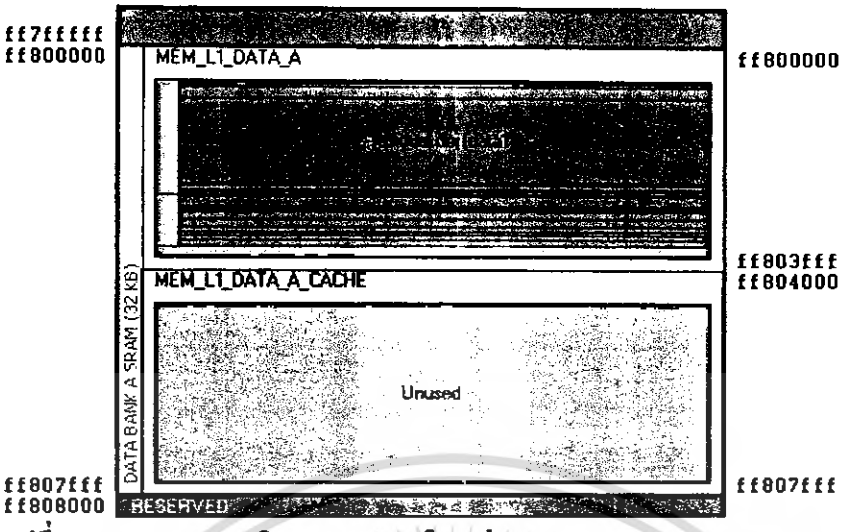
3.2 ปัญหาในการพัฒนาโปรแกรมเพื่อให้สามารถ run บน Blackfin ได้

3.2.1 ปัญหาการ Overflow ของข้อมูล

เนื่องจากในเริ่มแรกของการทำโปรแกรมลงบน Blackfin นั้นตัวโปรแกรม Visual DSP ของ Bllackfin นั้นได้แสดงข้อผิดพลาดเกิดขึ้น นั่นคือการ Overflow ของข้อมูลนั่นเอง สาเหตุเป็นเพราะเนื่องจากในการรับอินพุทเข้ามาเป็นไฟล์ที่ได้รับการบีบอัดจากโปรแกรมบีบอัดตามมาตรฐาน H.264 แล้วนั้นจำเป็นที่จะต้องใช้การแปลงข้อมูลกลับมาอยู่ในรูปแบบที่สามารถนำมอ่านได้ ซึ่งมีกระบวนการทำงานที่ค่อนข้างซับซ้อน และเนื่องจากการทำงานที่ซับซ้อน และ ฟังก์ชันการทำงานที่มากมายรวมถึงการเรียกใช้งานฟังก์ชันที่มากและซ้ำกันอยู่บ่อยครั้งจึงทำให้ตัวคอมไพเลอร์จำเป็นต้องสร้างIntermediate Code จำนวนมาก และยังมีการจองพื้นที่หน่วยความจำสำหรับรองรับอินพุทที่นำเข้ามาแปลเป็นข้อมูลเพื่อใช้ในการถอดรหัสอีกจึงทำให้ตัวโปรแกรม Visual DSP ไม่สามารถจองพื้นที่หน่วยความจำตามปกติได้เพียงพอต่อความต้องการของโปรแกรม

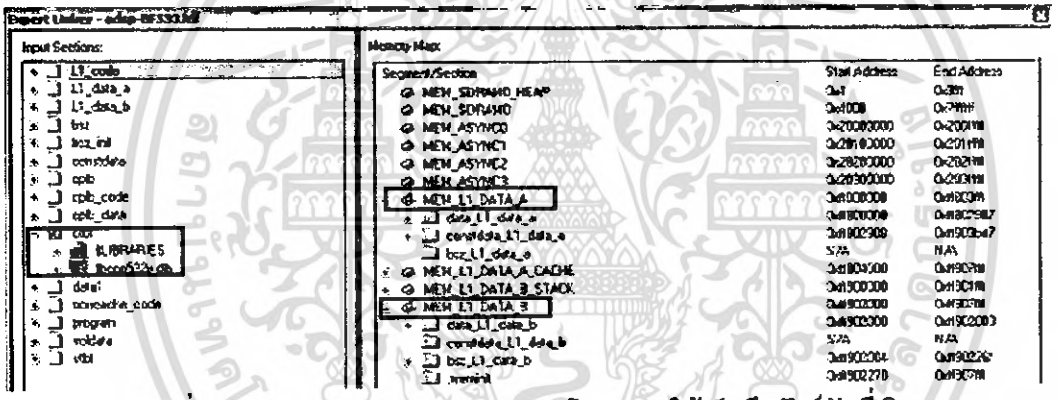
```
[Error 1i1040] "..\..\..\..\Documents and Settings\Administrator\Desktop\H264_BF_Final\Debug\adsp-BF533.ldf":487 Out of memory in output sectio
        Total of 0x1a5e4 word(s) were not mapped.
For more details, see 'linker_log.xml' in the output directory.

[Error 1i1040] "..\..\..\..\Documents and Settings\Administrator\Desktop\H264_BF_Final\Debug\adsp-BF533.ldf":493 Out of memory in output sectio
        Total of 0x1a5e4 word(s) were not mapped.
For more details, see 'linker_log.xml' in the output directory.

Linker finished with 2 errors
cc3089: fatal error: Link failed
Tool failed with exit/exception code: 1.
Build was unsuccessful.
```

รูปที่ 3.1 แสดงภาพการแจ้งข้อผิดพลาดของการคอมไพล์ เนื่องจากเกิดการ Overflow



รูปที่ 3.2 แสดงภาพการเกิด Overflow ในหน่วยความจำของ Blackfin DSP

จากรูปจะเห็นว่า MEM_L1_DATA_A นั้นที่กรอบเป็นสีแดง เป็นสิ่งที่แสดงให้เห็นว่ามีความผิดปกติในส่วนนี้ซึ่งคือ เกิดการ Overflow ในพื้นที่ Memory ในส่วนนี้นั่นเอง



รูปที่ 3.3 แสดงภาพการเกิด Overflow โดยแสดงให้เห็นถึงฟังก์ชันที่เกิด

จากรูปเป็นการแสดงปัญหาการเกิด Overflow ในส่วนต่างๆแบบ Text Mode ซึ่งจะทำให้เห็นถึงส่วนย่อยที่อยู่ในฟังก์ชันการทำงานต่างๆได้ละเอียดขึ้น ซึ่งจากรูปจะเห็นว่าปัญหาทั้ง 3 ส่วนที่เกิด Overflow นั้นอยู่ที่ Input ของ Ctor และ Output ของ MEM_L1_DATA_A และ MEM_L1_DATA_B ซึ่งเป็นส่วนที่ใช้ในการเก็บข้อมูลของฟังก์ชันต่างๆในโปรแกรม

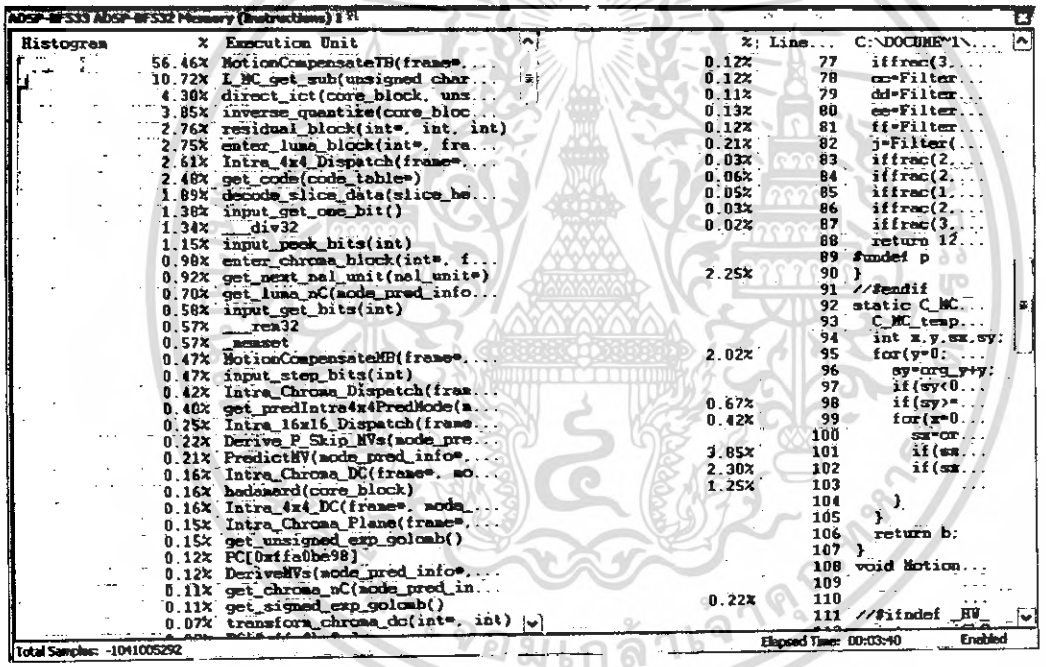
3.2.2 ปัญหาการจัดการ memory (หน่วยความจำ)

หลังจากการจัดการที่ทำให้โปรแกรมรันได้ตามปกติ ไม่เกิดการ Overflow แล้ว แต่ทางด้านประสิทธิภาพนั้นโปรแกรมยังทำงานได้ช้ามาก คือในการถอดรหัสภาพเคลื่อนไหวขนาด 640x480 นั้นใช้เวลาถึงเฟรมละ 45 นาทีขึ้นไปเลยทีเดียว ซึ่งถ้าจะทำการถอดรหัสทั้งหมดทุกเฟรมนั้น อาจจะใช้เวลาเกินครึ่งวัน ซึ่งเป็นสิ่งที่มีความผิดปกติมากสำหรับระยะเวลาที่ใช้ในการถอดรหัสที่นานขนาดนี้

จึงได้ทำการตรวจสอบการจัดเรียงหน่วยความจำอีกครั้งก็ไม่พบการแจ้งความผิดพลาดในการทำงานแต่อย่างใด

3.2.3 ปัญหาความเร็วในการถอดรหัสภาพวิดีโอ

ในส่วนของการทำงานของโปรแกรมตามมาตรฐาน H.264 นี้พบว่าโปรแกรมได้ใช้เวลาในการประมวลผลค่อนข้างนาน และส่งผลให้แสดงภาพเคลื่อนไหวไม่ได้แบบเวลาจริง (Real time) ซึ่งถือเป็นปัญหาด้านประสิทธิภาพของโปรแกรมที่ต้องได้รับการแก้ไข ดังนั้นจึงต้องมีการปรับปรุงและแก้ไขโปรแกรมเพื่อให้มีประสิทธิภาพด้านเวลาที่ดีขึ้น ซึ่งได้มีการวัดการทำงานของโปรแกรมโดยใช้โปรแกรม Visual DSP++ ซึ่งเป็นเครื่องมือของบริษัท Analog Device ในส่วนของบอร์ด Blackfin DSP ซึ่งเป็นบอร์ดที่ใช้ในการทำงานของโครงงานนี้ เมื่อวัดการทำงานของโปรแกรมออกมาจะได้ผลดังรูปข้างล่าง



รูปที่3.4 แสดงประสิทธิภาพในการประมวลผลของโปรแกรม

พบว่าการทำงานหลักที่โปรแกรมใช้การทำงานของ Microprocessor ประมวลผลจะอยู่ในฟังก์ชันการทำงานของ MotionCompensation โดยเหตุผลหลักๆที่ทำให้เกิดการคำนวณอย่างมากคือ

1. สาเหตุจากการวนตรวจสอบ และ " if statement " processor จะต้องเจอกับทางเลือก แต่ละทางเลือกต้องใช้การทำงาน 3 cycles ของ processor เมื่อทางเลือกรวมกันมากๆเข้าจะทำให้ลดประสิทธิภาพของอัลกอริธึมลง
2. เทคนิคไม่ได้ครอบคลุมกรณีที่เกิดขึ้นมากกว่า เนื่องจากกรณี half pel 'b' , 'h' และ 'j' ถูกใช้ในการ

คำนวณของกรณี quarter pel ดังนั้นการคำนวณเหล่านี้ควรจะจัดให้อยู่ในกรณีพิเศษในอัลกอริธึม อย่างไรตามในอัลกอริธึมเราให้มันอยู่ในกรณีอื่นๆ

3. การที่จำเป็นต้องทำการโหลดพิกเซลรอบข้างที่จะนำมาคำนวณทุกครั้งทั้งที่เมื่อผ่านไป 1 บล็อกย่อยๆ พิกเซลรอบข้างที่โหลดยังเป็นพิกเซลเดิม ทำให้เกิดการทำงานที่ซ้ำซ้อนโดยไม่จำเป็นและเป็นการเพิ่มภาระในการประมวลผลโดยไม่จำเป็น

3.3 การศึกษาการ Optimization

จากการศึกษาข้อมูลเพื่อทำการออปติไมเซชันโปรแกรม จึงได้มีการนำเอกสารวิชาการของต่างประเทศมาศึกษาเพื่อที่จะได้ทราบข้อมูลความก้าวหน้าว่ามีการพัฒนากันไปถึงไหนแล้ว ซึ่งจากการศึกษานี้ได้พบว่าทุกเอกสารวิชาการได้เห็นพ้องต้องกันว่า H.264 นี้เป็นมาตรฐานบีบอัดใหม่ที่มีการทำงานที่ละเอียดเพื่อประสิทธิภาพในการบีบอัดที่ดี แต่ประสิทธิภาพที่ได้นี้แลกมาด้วยค่าของเวลาที่เสียไปมากหลายเท่าตัว ซึ่งทุกเอกสารวิชาการได้พยายามที่จะพัฒนาโปรแกรมเพื่อให้ทำงานได้ดีขึ้น แต่ถึงอย่างไรก็ไม่มีวิธีการที่เป็นมาตรฐานในการพัฒนาโปรแกรมนั่นเพราะบางเอกสารวิชาการได้อ้างว่ามีการพัฒนาในบางส่วนของโปรแกรมแต่ก็ไม่ได้ผลตามที่ต้องการ บางเอกสารได้อ้างว่ามีการทำการพัฒนาบางส่วนของโปรแกรมและได้เห็นผลที่เร็วขึ้นแต่ก็ไม่ได้มีการอ้างถึงว่ามีการพัฒนาอย่างไร และไม่ได้มีการขยายความต่อไปว่าจะมีการพัฒนาอะไรต่อไป จึงเป็นสิ่งที่ค้างคามานานในการพัฒนาโปรแกรมบีบอัดภาพเคลื่อนไหวตามมาตรฐาน H.264 นี้

ในการหาข้อมูลจากเอกสารวิชาการของต่างประเทศนั้นได้พบเอกสารวิชาการหนึ่งซึ่งมีเนื้อหาที่น่าสนใจเกี่ยวกับการวัดประสิทธิภาพของการประมวลผลโปรแกรมบีบอัดข้อมูลตามมาตรฐาน H.264 โดยการพัฒนาลงบน DSP ที่ต่างชนิดกัน โดยหนึ่งในนั้นคือ Blackfin DSP ซึ่งเป็นรุ่นที่ใช้ในการทดลองของโครงงานนี้ด้วยคือ ADSP-BF533 ซึ่งเป็นผลงานของ Yair Moshe และ Nimrod Peleg โดยมีหัวข้อผลงานวิจัยคือ "Implementations of H.264/AVC Baseline Decoder on Different Digital Signal Processors" โดยมีเนื้อหาโดยรวมกล่าวถึงการนำเอามาตรฐานการบีบอัดภาพเคลื่อนไหวตามมาตรฐาน H.264 ไปพัฒนาลงบน DSP ซึ่งมีการพัฒนาลงบน DSP ทั้งสามค่ายคือ DM642, StarCore และ Blackfin ซึ่งเมื่อได้ศึกษาถึงผลงานวิจัยดังกล่าวนี้แล้วพบว่าทางผู้วิจัยพบปัญหาด้านประสิทธิภาพในการพัฒนาโปรแกรมบีบอัดเมื่อพัฒนาลงบน DSP เช่นเดียวกัน

ทางผู้วิจัยได้กล่าวถึงปัญหาที่หนักกว่าปัญหาอื่นๆในการทำโปรแกรมลงบน DSP เอาไว้คือปัญหาความซับซ้องของตัวโค้ด แม้แต่ Baseline Profile ซึ่งเป็นโพรไฟล์พื้นฐานที่ทำงานไม่ซับซ้อนที่สุดเมื่อเทียบกับโพรไฟล์อื่นๆยังมีความซับซ้อนมาก ในตัวถอดรหัสแบบ Baseline Profile นั้นถึงกับใช้การเขียนโปรแกรมถึง 13,000 บรรทัด และทางผู้วิจัยได้ระบุว่าถึงแม้ปัจจุบันทางผู้พัฒนาทรัพยากรของ DSP จะทำให้สามารถรองรับการทำงานที่หนักมากได้ แต่ยังไม่ใช่งานง่ายสำหรับการพัฒนาโปรแกรม

ตามมาตรฐาน H.264 อยู่ดี ซึ่งทางด้านผู้วิจัยได้มีการกล่าวถึงลูปที่ซับซ้อนที่มีการซ้อนลูปลึกถึง 4 ระดับขึ้นไปยิ่งทำให้โปรแกรมใช้ทรัพยากรมากขึ้นไปอีก และเป็นการเพิ่มข้อจำกัดในการพัฒนาโปรแกรมต่อไปข้างหน้าอีกแน่นอน

เนื่องจากการทำงานที่ซับซ้อนของโปรแกรมจึงทำให้ใช้ทรัพยากรทั้งทางด้านหน่วยความจำและการประมวลผลมาก ดังนั้นจากผลงานวิจัยนี้จึงได้มีการทำการออปติไมเซชันโปรแกรมลงบนDSP ชนิดต่างๆกัน โดยซึ่งจากการทำการออปติไมเซชันของผลงานวิจัยนี้ได้มีวิธีการหลักๆคือการทำการเข้าถึงหน่วยความจำโดยตรงนั่นเอง

ตารางที่ 3.1 แสดงผลการประมวลผลโปรแกรมบน DSP ต่างๆ

	DM642	StarCore	Blackfin
Non-optimized	2.1	1.9	1.9
Optimized	9	7.1	-

จากตารางจะพบว่าในส่วนของ DM462 นั้นมีการทำการออปติไมเซชันที่ดีกว่าเดิมขึ้นหลายเท่าการออปติไมเซชันของ DSP นี้จึงประสบความสำเร็จ ในส่วนของ StarCore นั้นเป็นเช่นเดียวกัน ถึงแม้ว่าความเร็วในการประมวลผลภาพเคลื่อนไหวจะต่ำกว่าของ DM642 ก็ตามแต่นับว่าสามารถพัฒนาความเร็วให้มีมากขึ้นได้ ในขณะที่ Blackfin DSP(ADSP-BF533) ที่ใช้ในโครงงานนี้ ทางเอกสารที่ศึกษานี้ได้อ้างว่าไม่สามารถทำการออปติไมเซชันได้ โดยความเร็วในการประมวลผลภาพเคลื่อนไหวในเบื้องต้นอยู่ที่ 1.9 เฟรมต่อวินาที สำหรับภาพเคลื่อนไหว QCIF (คือขนาด 176 x 144) ซึ่งในโครงงานนี้ใช้ภาพขนาด 320x240 ซึ่งเป็นภาพที่ใหญ่กว่า สามารถประมวลผลที่อัตราเร็ว 0.90 เฟรมต่อวินาที และเมื่อเปลี่ยนขนาดภาพเคลื่อนไหวเท่ากันกับของผลงานวิจัยพบว่าทำการประมวลผลได้ที่อัตราเร็วเดียวกัน ซึ่งในเอกสารวิชาการที่ศึกษานี้ได้ระบุว่าในขณะที่พวกเขาทำการวิจัยอยู่นั้นยังไม่สามารถทำการออปติไมเซชันลงบน Blackfin DP ได้ซึ่งเป็นผลที่ตรงกันกับของโครงงานที่ทำอยู่นี้

3.3.1 การแก้ปัญหาการ Overflow ของข้อมูล

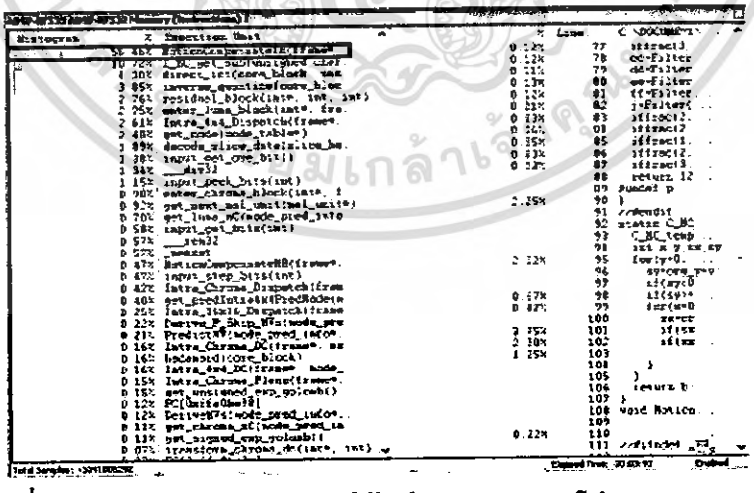
การแก้ไขปัญหานี้โดยเข้าไปยังฟังก์ชันการควบคุมการจัดการหน่วยความจำที่ตัวโปรแกรม Visual DSP ได้ทำการจำลองให้เพื่อความสะดวกในการจัดการหน่วยความจำ ซึ่งเมื่อได้เข้าไปดูแล้วได้พบว่าหน่วยความจำของส่วนใดในโปรแกรมที่ไม่เพียงพอ จึงทำการย้ายไปยัง External Memory ซึ่งเราสามารถทำการย้ายได้โดยสะดวกและสามารถจัดการกับหน่วยความจำของส่วนนี้ได้เนื่องจากมีขนาดที่มากเพียงพอในการแบ่งส่วนหน่วยความจำเพื่อนำมาเป็นส่วนของการใช้รันโปรแกรม จึงสามารถทำให้โปรแกรมรันได้ตามปกติโดยการทำให้หน่วยความจำในการรันเพียงพอด้วยวิธีการนี้

3.3.2 การแก้ปัญหาการจัดการ memory

การศึกษาหาข้อมูลและพบว่าในหลายๆกรณีที่ DSP มักจะมีปัญหาในการรันโปรแกรมที่ค่อนข้างช้านั้นบางกรณีเป็นผลมาจากนาดของ Heap และ Stack นั้นมีขนาดเล็กไป ไม่สามารถรองรับการทำงานของการรับอินพุทไฟล์และเก็บค่าต่างๆได้เพียงพอ จึงได้ทำการเพิ่มขนาดให้มีความเหมาะสม โดยดูจากค่าต่างๆที่เก็บระหว่างรันโปรแกรมว่าใช้พื้นที่เต็มหรือไม่ ถ้าใช้เต็มก็จะเพิ่มขนาดให้อีก ทำเช่นนี้ไปเรื่อยๆจนกระทั่งมีขนาดที่เพียงพอกับความต้องการของโปรแกรม ซึ่งนับว่าประสบความสำเร็จเป็นอันมากเพราะหลังจากการทำการแก้ไขขนาดและการจัดเรียงหน่วยความจำใหม่แล้วทำให้โปรแกรมสามารถทำการรันได้ตามปกติและมีความเร็วเพิ่มขึ้นอย่างเห็นได้ชัด นั่นคือมีความเร็วในการประมวลผลภาพเคลื่อนไหวขนาด 640 x 480 อยู่ที่ประมาณ 6 วินาทีต่อเฟรม และมีการประมวลผลภาพเคลื่อนไหวขนาด 320 x 240 อยู่ที่ประมาณ 1.5 วินาทีต่อเฟรม ซึ่งถึงแม้ผลของเวลาจะยังไม่เพียงพอต่อการประมวลผลจนเป็นเป็นที่น่าพอใจ แต่นับว่าเป็นการพัฒนาที่ดีขึ้นกว่าเดิมมาก และสามารถใช้การพัฒนานี้เป็นพื้นฐานในการพัฒนาส่วนของตัวโปรแกรมต่อไป

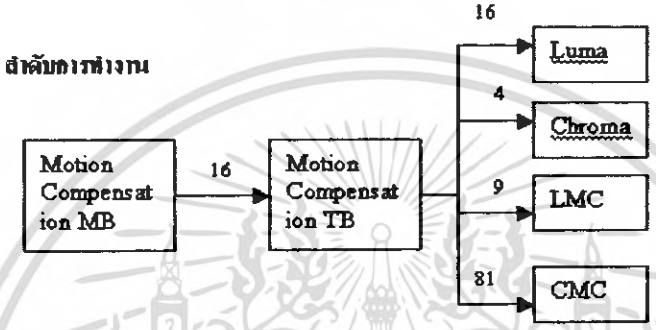
3.3.3 การพัฒนาเพิ่มความเร็วในการถอดรหัสภาพวิดีโอ

ในส่วนของการทำงานของโปรแกรมตามมาตรฐาน H.264 นี้พบว่าโปรแกรมได้ใช้เวลาในการประมวลผลค่อนข้างนาน และส่งผลให้แสดงภาพเคลื่อนไหวไม่ได้แบบเวลาจริง (Real time) ซึ่งถือเป็นปัญหาด้านประสิทธิภาพของโปรแกรมที่ต้องได้รับการแก้ไข ดังนั้นจึงต้องมีการปรับปรุงและแก้ไขโปรแกรมเพื่อให้มีประสิทธิภาพด้านเวลาที่ดีขึ้น ซึ่งได้มีการวัดการทำงานของโปรแกรมโดยใช้โปรแกรม Visual DSP++ ซึ่งเป็นเครื่องมือของบริษัท Analog Device ในส่วนของบอร์ด Blackfin DSP ซึ่งเป็นบอร์ดที่ใช้ในการทำงานของโครงงานนี้ เมื่อวัดการทำงานของโปรแกรมออกมาจะได้ผลดังรูปข้างล่าง



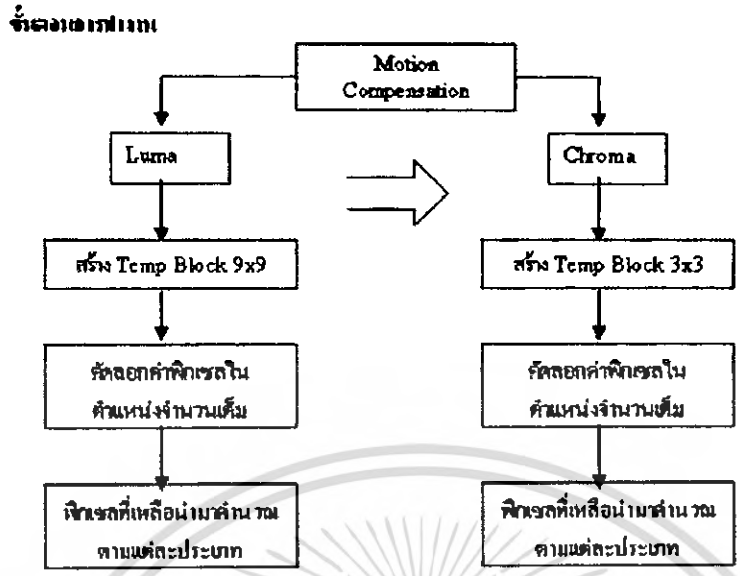
รูปที่3.5 แสดงประสิทธิภาพในการใช้หน่วยความจำของโปรแกรม

พบว่าการทำงานหลักที่โปรแกรมใช้การทำงานของ Microprocessor ประมวลผลจะอยู่ในฟังก์ชันการทำงานของ MotionCompensation ดังนั้นการปรับปรุงแก้ไขโปรแกรมส่วนใหญ่จึงควรแก้ไขในการทำงานที่ฟังก์ชันนี้ซึ่งเมื่อทำการศึกษาเกี่ยวกับประสิทธิภาพด้านเวลาของมาตรฐานนี้พบว่าโปรแกรมจะเกิดปัญหาคอขวด (Bottle Neck) ในส่วนของการทำ Motion Compensation จริงๆ ซึ่งจะมีปัญหาทั้งการทำงานในส่วนของตัวโปรแกรม และ ในส่วนของการจัดการหน่วยความจำ ดังนั้นการทำงานในส่วนนี้จึงเป็นส่วนที่ทำให้เกิดปัญหาความล่าช้ามากที่สุดจึงมีการศึกษาถึงแนวทางการแก้ไขปัญหา โดยเริ่มจากการวิเคราะห์ถึงโครงสร้างการทำงานของโปรแกรมในส่วนของฟังก์ชันนี้ก่อน



รูปที่ 3.6 แสดงลำดับการทำงานและจำนวนลูปโดยรวมของMotion Compensation

จากรูปเป็นการแสดงถึงลำดับในการเรียกใช้งานฟังก์ชัน และจำนวนลูปคร่าวๆในการเรียกใช้ฟังก์ชันหนึ่งๆ ซึ่งจะเห็นว่าในการทำการคำนวณมาโครบล็อกย่อยๆเพียงแค่มาโครบล็อกเดียวยังใช้จำนวนลูปมากถึงขนาดนี้ ซึ่งในเฟรมหนึ่งๆจะมีจำนวนมาโครบล็อกอยู่อีกมากมาย ดังนั้นจึงไม่ใช่เรื่องแปลกที่จะเกิดความล่าช้าในการประมวลผลในส่วนนี้ ต่อมาจะเข้าไปดูถึงการทำงานในโปรแกรมว่ามีขั้นตอนการทำงานอย่างไร



รูปที่3.7 แสดงถึงการทำงานโดยรวมของ Motion Compensation

ดังนั้นพอจะสรุปได้ว่าการทำงานที่ล่าช้าของโปรแกรมนี้เกิดจากกระบวนการทำงานที่ซ้ำซ้อนกัน นั่นคือการคำนวณและการโหลดพิกเซลเพื่อที่จะทำการคำนวณ และการทำงานมีจำนวนลูปที่ค่อนข้างมากส่งผลให้จำนวนรอบในการประมวลผล(Cycles) มากขึ้นตามไปด้วย

ต่อมาจึงได้มีการศึกษาเกี่ยวกับอัลกอริทึมที่จะทำมาใช้ในการ Optimize โปรแกรม และได้พบอัลกอริทึมหนึ่งซึ่งจะสามารถนำมาประยุกต์ใช้กับโปรแกรมได้คือ "Optimization of Motion Compensation for H.264 Decoder by Pre-Calculation" โดย Muhammad Owais Khan, Umar Khan, Sardar Adnan Rahim and Syed Irtiza Ali ซึ่งได้กล่าวถึงสาเหตุและวิธีการในการพัฒนาโปรแกรมเอาไว้ ซึ่งเนื้อหาในบทความที่เขียนไว้มีเนื้อหาโดยรวมดังนี้

3.3.3.1 การออปติไมเซชันที่1 การสร้างขอบ

การสร้างขอบเป็นเทคนิคที่ใช้เพื่อลดการตัด แต่ละครั้งที่ค่า pixel ถูกอ่านจากหน่วยความจำตามที่ได้อธิบายไปแล้วก่อนหน้านี้กระบวนการตัด ทำให้เกิดภาระอย่างมากกับ processor เนื่องจากมันต้องทำงานทุกๆครั้งที่ค่า pixel ถูกเข้าถึงจากหน่วยความจำ

เพื่อที่จะลดจำนวนของการตัดเราสร้างขอบรอบๆรูปภาพในหน่วยความจำ ขอบภาพสร้างอย่างง่ายๆโดยการขยาย pixel ที่ขอบรูปภาพออกไป ค่า pixel ขอบที่ขยายขึ้นคือ 16 pixels (นั่นคือขนาดของ macro block) เหตุผลที่ต้องทำการขยายขอบ 16 pixel เพราะแม้ว่าภายใต้เงื่อนไขที่เกือบสุดขีดนั้น motion vectors จะได้ไม่ชี้ออกนอกรูปภาพโดยจำนวนที่มากกว่าหนึ่ง macro block สิ่งนี้ได้ถูกทดสอบแล้วภายใต้การเข้ารหัสหลายรูปแบบ เหตุผลอื่นซึ่งพิจารณาการเลือกของส่วนขยายของขอบโดย 16 pixels คือว่าในข้อมูลโครงสร้างของเราถูกเข้าถึงโดยการใช้ integer pointer สำหรับเทคนิคการเข้าถึง

หน่วยความจำแบบนี้ ข้อมูลต้องเป็น byte alignment 4 ไบท์ pixels ขอบ 16 pixels ดูแล byte alignment ของข้อมูล สำหรับขนาดภาพ CIF (352 x 288) จะต้องทำการ copy ทั้งหมด 7680 pixels ขณะที่กรณีของการตัดแม้กระทั่งสำหรับรูปภาพที่มีเพียงกรณี full pel การตัดต้องทำถึง 202240 ครั้ง ดังนั้นกระบวนการสร้างขอบนั้นใช้เพียง 3.79 % การตัดแบบเก่า

3.3.3.2 การออปติไมเซชันที่2 การคำนวณ Half-pel ทั้งหมด

ขั้นตอนที่สองของอัลกอริธึมที่เราเสนอเกี่ยวข้องกับการคำนวณของค่า half pixel ทั้งหมด สำหรับรูปทั้งหมดกล่าวคือ 'b' , 'h' และ 'j' การคำนวณของกรณี half pel ทั้งหมด นำการตรวจสอบและ overheads ที่ไม่จำเป็นซึ่งต้องอยู่ในตัวถอดรหัสอ้างอิงออกพร้อมกันทั้งหมด ประโยชน์ที่เราได้รับจากการคำนวณค่า half pixel ทั้งหมด ออกจากส่วนหลักของอัลกอริธึม motion compensation คือ

1. ตามที่ค่า half pel ทั้งหมดต้องถูกคำนวณ เราไม่ต้องทำการตรวจสอบอะไรเลย ค่าทั้งหมดที่ถูกคำนวณในการทำงานครั้งเดียวโดยไม่มีทางเลือก

2. การคำนวณค่าทั้งหมดพร้อมกัน ลดการเข้าถึงหน่วยความจำซ้ำๆที่บริเวณเดียวกัน เช่นเดียวกันกับการเข้าถึงหน่วยความจำสามารถทำให้ดีขึ้นโดยการเข้าโดยใช้ integer pointers เนื่องจากมันไม่สามารถใช้ได้ถ้าเราจะไม่คำนวณค่าทั้งหมด มันจะเกี่ยวข้องกับผลของ byte alignment

3. การใช้ integer pointers เพื่อที่จะเข้าถึงหน่วยความจำ และยังทำให้สามารถใช้วิธีการเฉพาะของ DSP processor ได้ [4] ด้วยเหตุนี้ประสิทธิภาพการคำนวณของอัลกอริธึมจะเพิ่มขึ้นอย่างมาก

การคำนวณค่า half pixel ทั้งหมดอาจจะดูเหมือนสิ้นเปลืองทรัพยากรของ processor เนื่องจากค่าทั้งหมดไม่ได้ถูกใช้ ในทางตรงกันข้ามผลที่เกิดขึ้นจริงกลับได้ว่าเป็นการลดภาระของ processor อย่างมาก การวิเคราะห์สถิติการเกิดขึ้นกรณี motion compessation ต่างๆของ stream ที่ใช้ทดสอบที่แสดงในตารางที่ 1. ผลลัพธ์ที่ไม่รวม "Susie" steams อื่นๆมีกรณี quarter pel เกิดขึ้นประมาณ 80% ดังนั้นมีประมาณ 90% ของค่า half pixel ที่ถูกใช้ ที่ซึ่งอยู่ในการคำนวณค่า quarter pixel หรือ ในกรณีของการเกิดขึ้นของกรณี half pixel

3.3.3.3 การออปติไมเซชันที่3 การลดการทำงานของ Motion Compensation

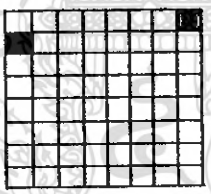
เนื่องจากการทำงานของ Motion Compensation เป็นการทำงานที่หนักมาก ดังนั้นบทความนี้จึงเสนอว่าควรจะทำการคำนวณค่า Half-pel และ Quarter-pel ตั้งแต่เฟรมแรกเผื่อเอาไว้เลย หลังจากนั้นเฟรมต่อๆมาที่อ้างอิงจากเฟรมแรกที่จะชี้ไปยังพิกเซลที่คำนวณไว้แล้วจะเอาค่านั้นไปใช้ได้ไม่ต้องคำนวณ

จากการศึกษาพบว่าบางวิธีการจำเป็นจะต้องไปแก้ไขและกำหนดค่าใหม่ในส่วนของตัวเข้ารหัสแทน บางวิธีการก็อาจจะเป็นไปไม่ได้ที่จะนำมาใช้จริง ดังนั้นจึงมีการศึกษาค้นคว้าเกี่ยวกับวิธีใหม่ที่นำมาใช้พัฒนาโปรแกรม

3.3.3.4 การออปติไมเซชันที่4 การคำนวณ pixel ที่ไม่ซ้ำในการคำนวณ Half-pel

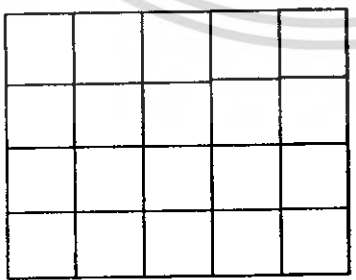
การทำการ Optimize โปรแกรมนี้มีการการคือ ในการอ่านค่าพิกเซลจากเฟรมอ้างอิงมาเก็บไว้ในหน่วยความจำจะทำการอ่าน ณ บริเวณรอบข้างพิกเซลที่ต้องการคำนวณ ดังนั้นเมื่อทำการคำนวณพิกเซลที่อยู่ถัดไปจะทำการคำนวณหาพิกเซลที่อยู่รอบข้างอีก นั้นหมายความว่าพิกเซลที่อยู่รอบข้องของพิกเซลทั้งสองนี้จะมีบางส่วนที่เป็นค่าเดียวกัน แต่ทำการคำนวณหาใหม่ทั้งหมดทุกครั้ง ดังนั้นจึงเสียเวลาและจำนวนรอบในการประมวลผลโดยเปล่าประโยชน์ การพัฒนาโปรแกรมจะเริ่มจาก

1. ทำการตรวจสอบว่า มาโครบล็อคนั้นๆ เป็นส่วนของ Inter Prediction หรือ Intra Prediction เนื่องจากในกรณีที่เป็น Inter Prediction มาก่อน แล้วเมื่อเปลี่ยนเป็น Intra Prediction นั้น ค่าตำแหน่งพิกเซลที่เก็บไว้อาจจะไม่ต่อเนื่องกัน
2. ทำการตรวจสอบบริเวณขอบของเฟรม เนื่องจากในบริเวณขอบขอเฟรมนั้น ค่าพิกเซลก่อนหน้าที่เก็บไว้จะใช้ไม่ได้เมื่อขึ้น index ใหม่
   จากรูปข้างล่าง ในส่วนที่เป็นสีแดงคือส่วนที่เมื่อคำนวณไปถึง แต่เมื่อขึ้น index ใหมคือสีน้ำเงินจะไม่สามารถอ้างอิงจากบล็อคก่อนหน้าได้ เพราะ ค่าพิกเซลไม่ได้อยู่ใกล้เคียงกัน



รูปที่ 3.8 แสดงการอ้างอิงในส่วนที่ไม่สามารถโหลดซ้ำได้

3. ทำการตรวจสอบจำนวนพิกเซลที่สามารถเก็บได้ ณ การคำนวณครั้งถัดไป



รูปที่ 3.9 แสดงส่วนที่ซ้ำและส่วนที่ต้องคำนวณใหม่

จากรูปในส่วนสีน้ำเงินเป็นส่วนที่ส่วนมากค่าพิกเซลที่คำนวณมักจะซ้ำกัน และในส่วนสีแดงเป็นส่วนที่ชิฟออกไปโดยเราจะต้องคำนวณในส่วนนี้ แต่ในส่วนสีน้ำเงินทั้งหมดเราสามารถใช้ค่าเดิมได้ เพราะฉะนั้นโดยทฤษฎีแล้วอัลกอริทึมนี้สามารถลดการคำนวณลงได้ประมาณ 3 ใน 4 เลยทีเดียว แต่ถึงอย่างนั้นก็ตาม ในส่วนของการคำนวณ Half-pel นั้นเป็นเพียงส่วนหนึ่งในหลายๆส่วนที่ใช้คำนวณ Motion Compensation ซึ่งโดยรวมแล้วการทำงานของ Motion Compensation ยังไม่ได้ลดลงจากเดิมมากนัก จึงได้มีการคิดที่จะพัฒนาโปรแกรมในส่วนของการเก็บค่าของบล็อกอ้างอิงไว้ให้มีขนาดที่ใหญ่ขึ้น เพื่อที่จะได้ไม่ต้องโหลดข้อมูลจากหน่วยความจำหลายๆรอบ ซึ่งได้มีการทำการเพิ่มขนาด array ที่เก็บให้มีขนาดใหญ่ขึ้น และเพิ่มฟังก์ชันที่ใช้ในการเลื่อนชี้ค่าที่เก็บไว้อีกที

แต่การทำงานหลังจากการพัฒนานั้นไม่ได้ผลดีเท่าใดนัก เนื่องจากการทำงานของโปรแกรมกลับมากขึ้นกว่าเดิม โดยจากเปอร์เซ็นในการประมวลผลของซีพียูนั้นการทำงานได้เพิ่มขึ้นด้วย ในขณะที่การเข้าถึงหน่วยความจำก็ไม่ได้ลดลง นั่นเพราะการเก็บค่าบล็อกจากเฟรมอ้างอิงนั้นเก็บเฉพาะช่วงแรกเท่านั้น แต่การที่ต้องตรวจสอบการทำงานกลับจะต้องทำทุกรอบของการอ่านค่าพิกเซล ดังนั้นผลที่ได้จากการเก็บค่าบล็อกอ้างอิงซึ่งมีเพียงไม่มาก กับการต้องเพิ่มฟังก์ชันการทำงานในการอ่านบล็อกอ้างอิงกลับต้องทำมากขึ้นกว่าเดิมมาก ดังนั้นผลที่ได้จากการพัฒนาโปรแกรมด้วยวิธีนี้จึงไม่ประสบความสำเร็จ คือนอกจากจะไม่สามารถลดการเข้าถึงหน่วยความจำได้แล้ว ยังเพิ่มภาระให้กลับซีพียูอีก

บทที่ 4

การทดลองและผลการทดลอง

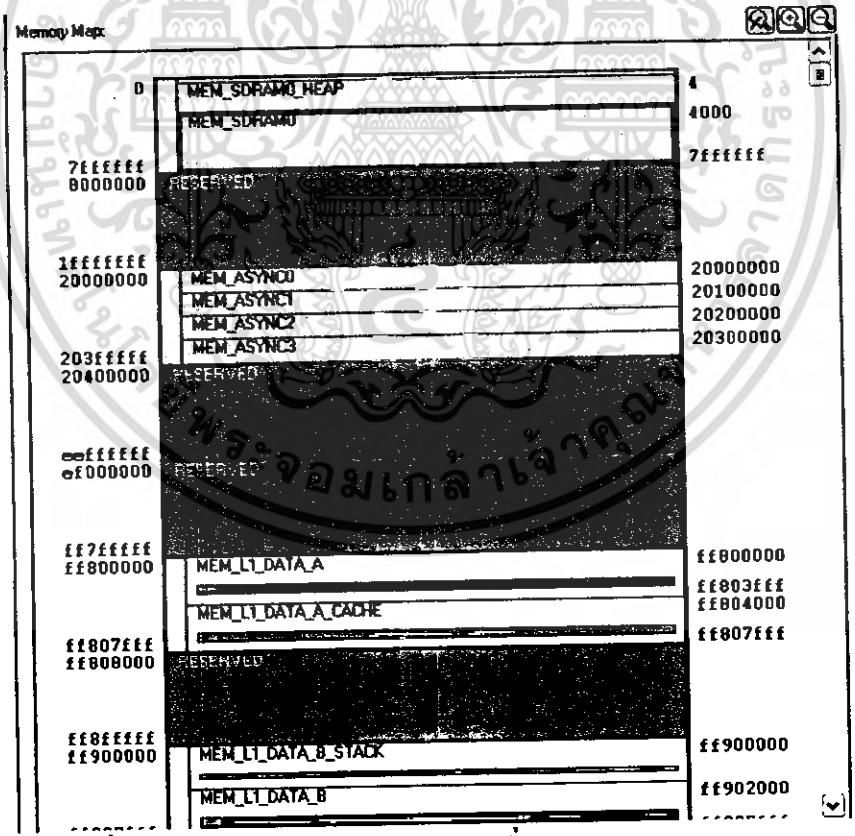
4.1 การทดลอง

ในการโครงงานนี้ได้ทำการทดลองโดยการใช้ Visual DSP ของ Blackfin Board โดยทำการพัฒนาโปรแกรมด้วยวิธีต่างๆกันไป โดยแยกออกเป็นการพัฒนาเกี่ยวกับหน่วยความจำ โดยทำการจัดเรียงหน่วยความจำใหม่ และ ทำการปรับขนาดหน่วยความจำในส่วนต่างๆเพื่อประสิทธิภาพในการประมวลผลที่ดีขึ้น นอกจากนี้ยังมีการพัฒนาตัวโปรแกรมเพื่อให้มีการทำงานที่ดีขึ้น ด้วยอัลกอริทึมต่างๆเพื่อลดการทำงานของโปรแกรมและส่งผลต่อประสิทธิภาพของโปรแกรมโดยรวมในการทดลองนี้จะทดลองกับภาพวิดีโอขนาดต่างๆ แต่จะทดลองกับภาพขนาด 320 x 240 เป็นหลัก

4.2 ผลการทดลอง

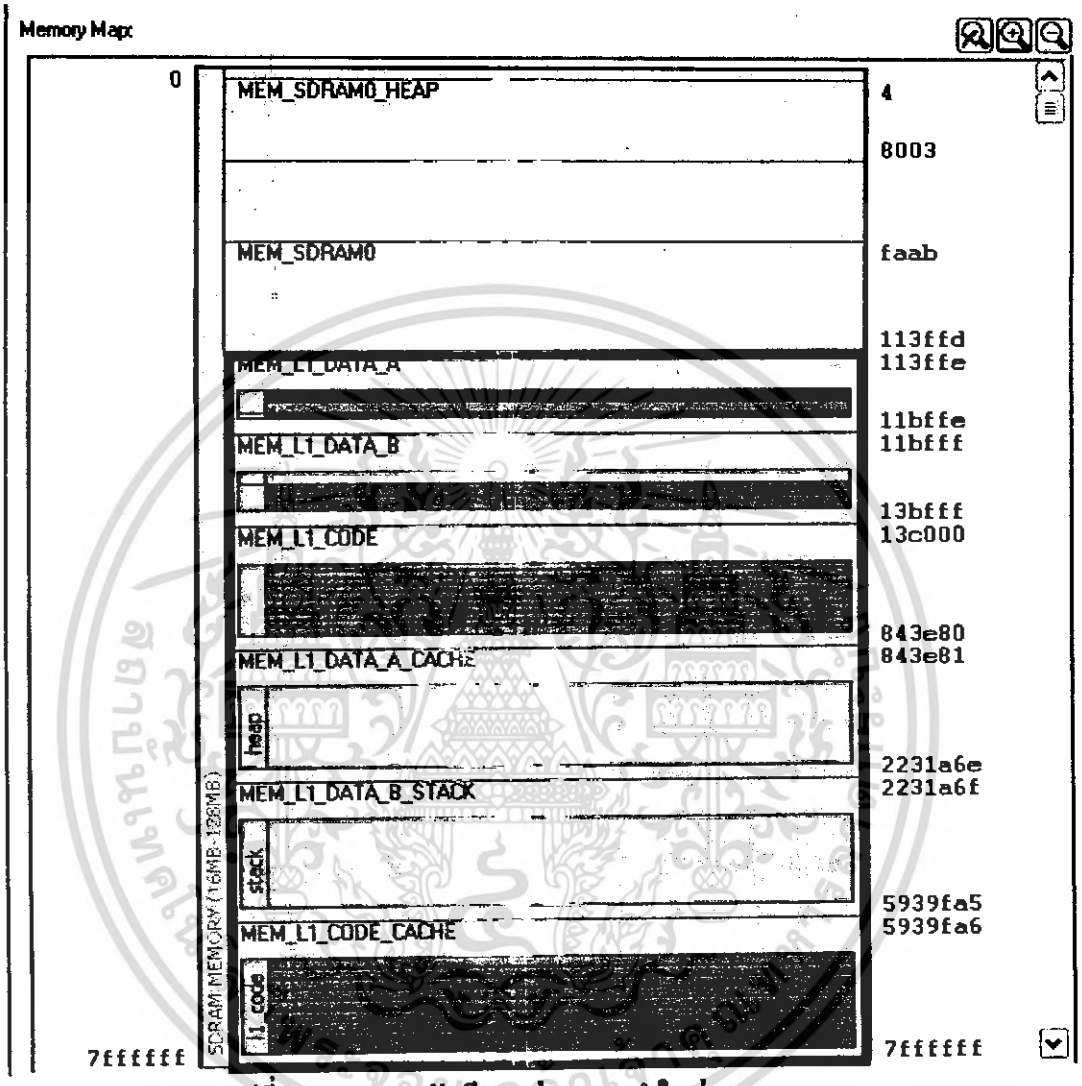
4.2.1 ผลการแก้ไขการ Overflow

ปัญหา Overflow ที่เกิดเป็นเพราะมีหน่วยความจำไม่เพียงพอสำหรับการทำงานในส่วนต่างๆ ซึ่งเมื่อเข้าไปดูการจัดเรียงหน่วยความจำแบบดั้งเดิมแล้ว พบว่าขนาดพื้นที่หน่วยความจำในบางส่วนนั้นมีน้อยไป ดังรูป



รูปที่ 4.1 แสดงการจัดเรียงหน่วยความจำเริ่มแรกแบบ Graphic Mode

จากรูปในส่วนที่ได้ทำการตีกรอบเอาไว้ เป็นส่วนของ MEM_SDRAM0 ซึ่งเป็น External Memory ในการแก้ปัญหาจำทำการลดขนาดของหน่วยความจำในส่วนนี้ลงเพื่อนำข้อมูลและ Source Code มาใส่แทน จะได้ผลการจัดเรียงหน่วยความจำดังรูป

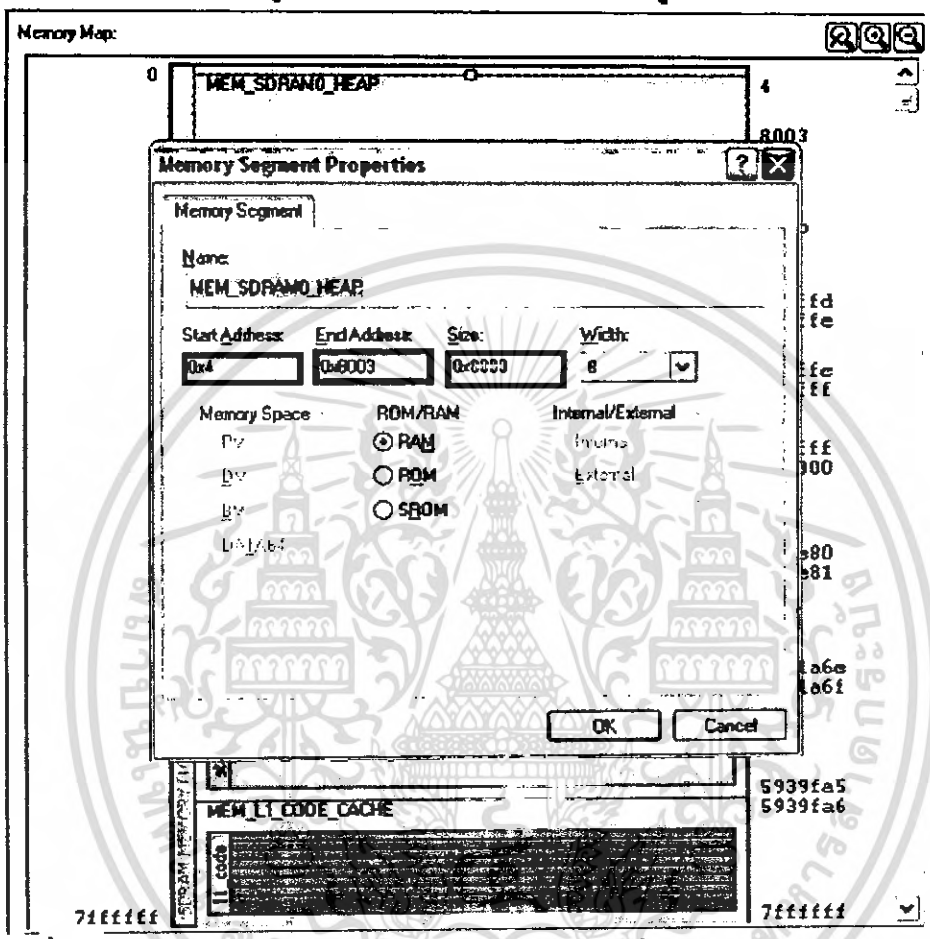


รูปที่ 4.2 แสดงการจัดเรียงหน่วยความจำใหม่

จากรูปภายในกรอบที่สร้างขึ้น คือส่วนของ Memory ที่ Source Code ต้องการเพื่อที่จะใช้รันโปรแกรม ซึ่งได้ทำการย้ายมาจากส่วนของ Internal Memory มาที่ External Memory และทำการขยายขนาด Memory ให้มีขนาดใหญ่ขึ้นเพียงพอกับความต้องการของโปรแกรม ซึ่งสามารถทำได้โดยใช้วิธีเปลี่ยนเปลี่ยนขนาด Memory ไปเรื่อยๆและทำการรันโปรแกรม ถ้ายังเกิดการ Overflow อยู่ก็ทำการเพิ่มขนาดอีก เมื่อเพิ่มขนาดให้เพียงพอกับความต้องการพื้นที่ Memory ที่เพียงพอที่จะใช้แล้ว เมื่อทำการรันผล จะสามารถคอมไพล์และรันผ่านได้ไม่มีปัญหา

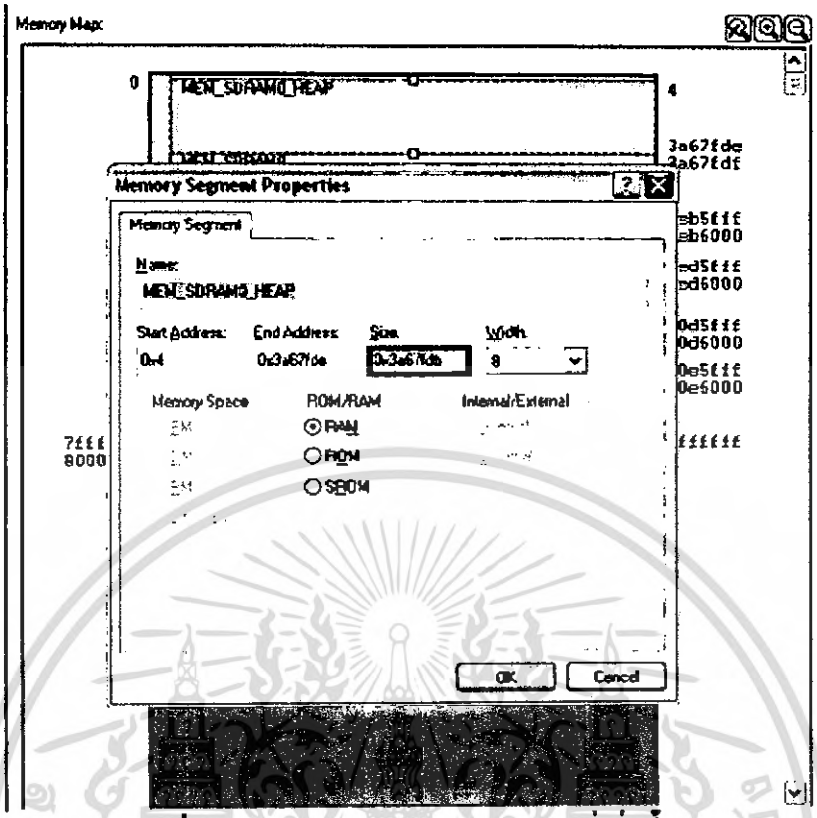
4.2.2 ผลการแก้ไขการจัดการ memory

การรันโปรแกรมเมื่อทำการจัดเรียงหน่วยความจำใหม่แล้วทำให้สามารถรันผ่านได้ แต่พบปัญหาใหม่คือใช้เวลารันนานมาก จึงเข้าไปตรวจสอบหน่วยความจำแบบละเอียดอีกครั้งซึ่งพบว่า ขนาดของ Heap ซึ่งเป็นส่วนที่ใช้พักข้อมูลในหน่วยความจำนั้นมีน้อยมากดังรูป



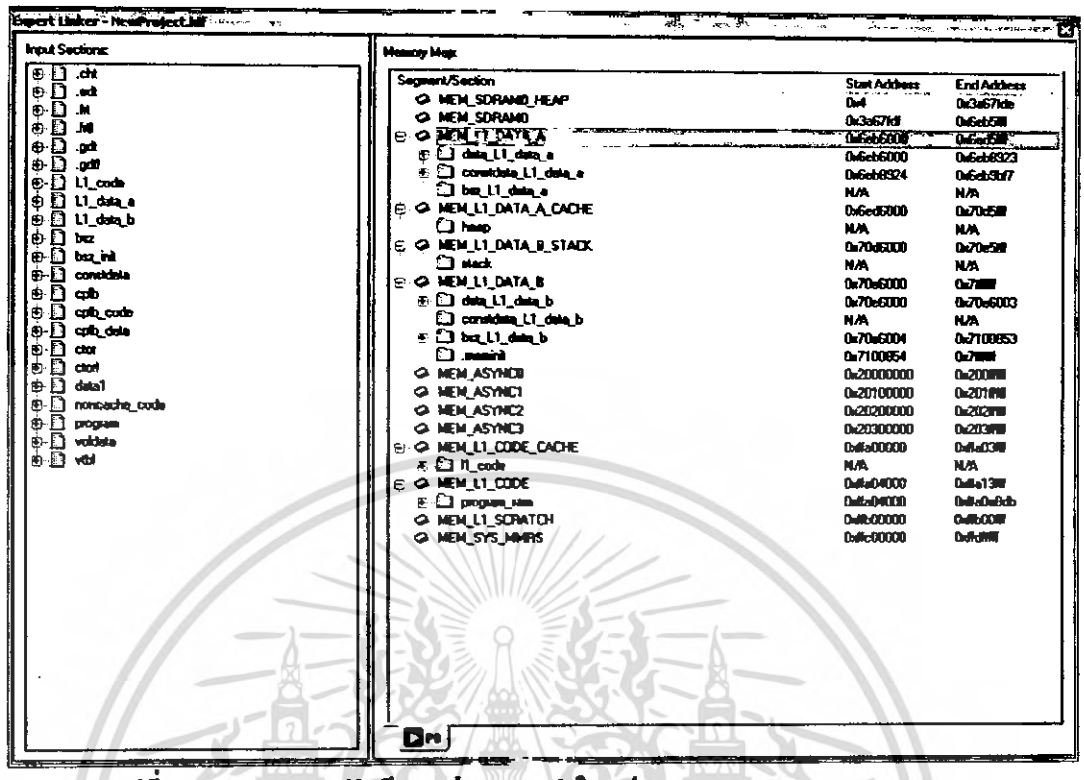
รูปที่ 4.3 แสดงขนาดของ Heap ก่อนหลังจากทำการย้ายฟังก์ชันไว้ใน External Memory

ซึ่งขนาดของ Heap นี้มีไม่เพียงพอต่อการรองรับข้อมูลจำนวนมาก จึงทำการเพิ่มขนาดของ Heap ขึ้น และวิธีการที่ใช้กำหนดขนาดของ Heap คือ ทำการหาขนาดที่ของหน่วยความจำที่เหมาะสมให้กับฟังก์ชันและส่วนที่ใช้งานอื่นๆ จากนั้นพื้นที่ที่เหลือจึงนำมาเพิ่มให้กับ Heap และ Stack นั่นเองโดยเมื่อทำการเพิ่มขนาดแล้วจะได้ขนาดโดยรวมทั้งหมดคือ



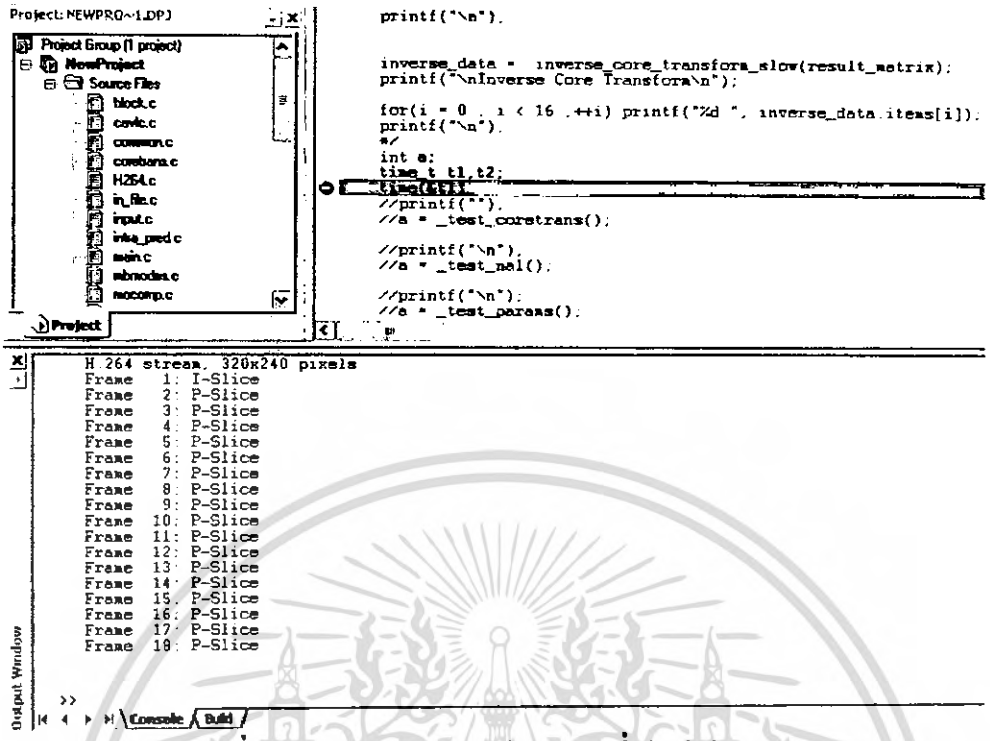
รูปที่ 4.4 แสดงให้เห็นถึงขนาดของ Heap ที่เพิ่มขึ้น

เมื่อทำการจัดเรียงหน่วยความจำใหม่จะทำการเพิ่มขนาดพื้นที่ของ External Memory ในส่วนของ Heap และ Stack ให้มีขนาดมากขึ้นเพียงพอกับความต้องการของโปรแกรมที่ต้องการเรียกใช้ จะได้ผลการจัดเรียงหน่วยความจำได้ ซึ่งขนาดของ Heap นี้ถ้ามากก็จะยิ่งรองรับการทำงานได้มากขึ้นด้วย ดังนั้นในโครงงานนี้จึงใช้วิธีกำหนดขนาดของส่วนอื่นๆที่ต้องการใช้ External Memory ให้เพียงพอกับความต้องการ และหน่วยความจำที่เหลือทั้งหมดจะทำการใส่ให้กับขนาดของ Memory ดังรูป ซึ่งในที่นี้ได้กำหนดให้ Heap มีขนาด 0x3a67fdb (ซึ่งขนาดนี้สามารถเปลี่ยนแปลงได้เนื่องจากหน่วยความจำที่กำหนดให้กับฟังก์ชันอื่นๆนั้นกำหนดไว้เกินความต้องการที่จะใช้ ดังนั้นถ้าต้องการ Heap ขนาดใหญ่ขึ้นก็สามารถทำการลดขนาดของหน่วยความจำในส่วนอื่นลงได้) ซึ่งเป็นขนาดของ External Memory เมื่อทำการใส่ให้กับการทำงานอื่นแล้ว



รูปที่ 4.5 แสดงผลการจัดเรียงหน่วยความจำในรูปแบบ Text Mode

จากรูปเป็นการเป็นภาพเพิ่มขนาดของหน่วยความจำในรูปแบบของ Text Mode ซึ่งเหมือนกับภาพใน Graphic Mode นั่นเอง ในส่วนของ Heap และ Stack ซึ่งจะมีผลต่อความเร็วในการรันทำให้ความเร็วในการรันโปรแกรมมีความเร็วเพิ่มขึ้นมาก ดังรูป



รูปที่ 4.6 แสดงผลการรันของโปรแกรมที่เป็นปกติ

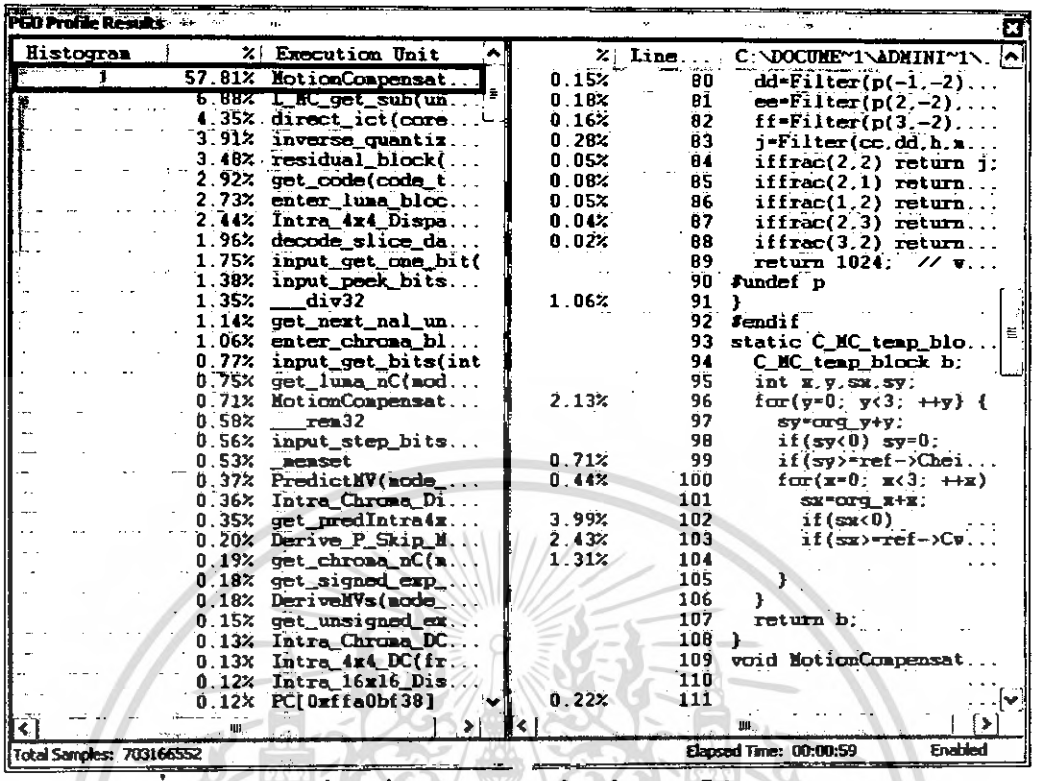
จากรูปเป็นภาพที่แสดงให้เห็นว่าเมื่อทำการเพิ่มขนาดหน่วยความจำในส่วนของ Heap ให้มีปริมาณที่เพียงพอต่อความต้องการของโปรแกรมแล้ว จะสามารถทำการประมวลผลได้ตามปกติ และมีประสิทธิภาพด้านความเร็วในการประมวลผลเพิ่มขึ้นด้วย

4.2.3 ผลการแก้ไขความเร็วในการถอดรหัส

การแก้ไขในส่วนของ Source Code นี้มีการแก้ไขในหลายส่วน และมีอัลกอริทึมที่ใช้หลายรูปแบบด้วยกัน

1. การสร้างซอบ

เป็นวิธีที่ไม่ซับซ้อนอะไรมาก และทำการลดการทำงานโดยรวมของโปรแกรมไม่มากนัก โดยการทำงานโดยรวมของโปรแกรมทั้งหมดไม่ได้มีอะไรเปลี่ยนแปลงจากตอนแรกเลย โดยสามารถแสดงผลการทำการประมวลผลในฟังก์ชันต่างๆ ได้ดังรูป

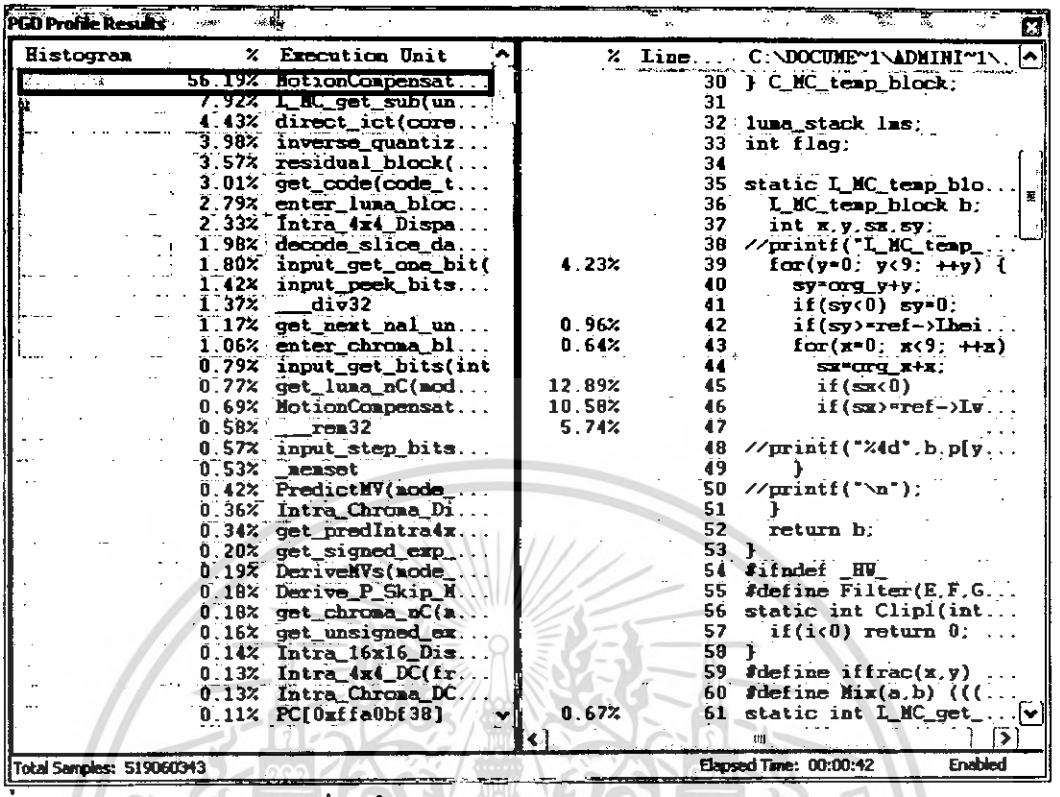


รูปที่ 4.7 แสดงเปอร์เซ็นต์การทำงานของส่วนต่างๆของโปรแกรม

จากรูปเป็นการทำงานโดยรวมของโปรแกรม โดยฟังก์ชันที่ทำการประมวลผลมากที่สุดคือ Motion Compensation โดยคิดเป็นเปอร์เซ็นต์ได้ 57.81 % จากการประมวลผลทั้งหมด

2. การลดการทำงานของ Motion Compensation

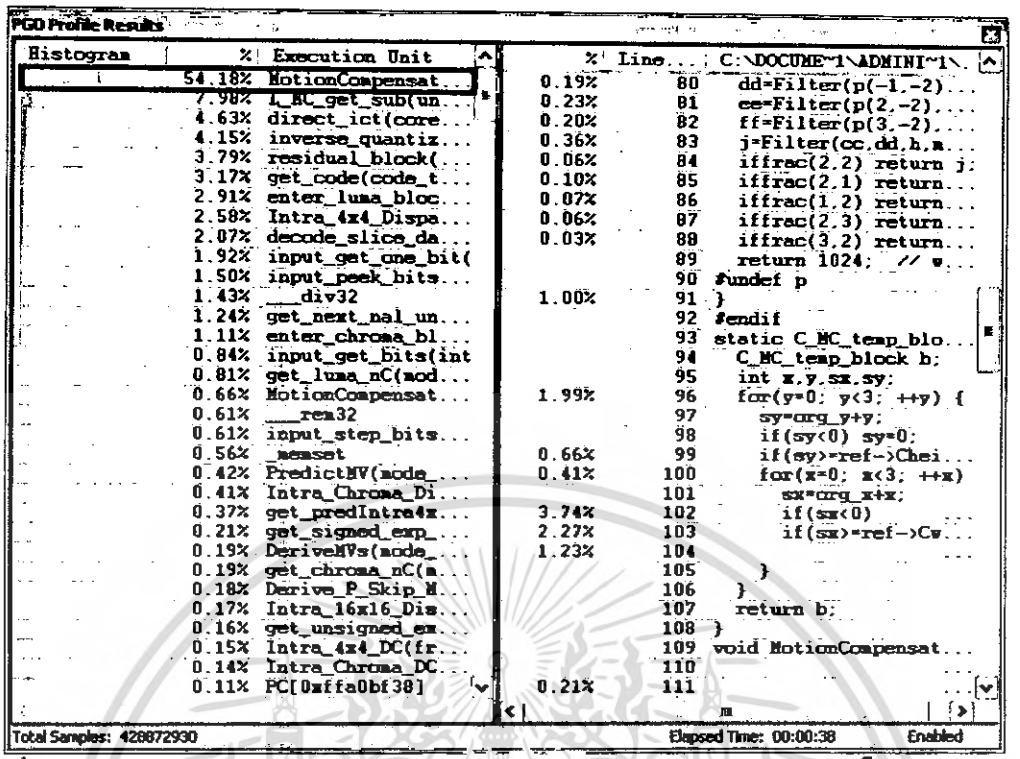
จากการทดลองพบว่าการทำงานโดยรวมของโปรแกรมยังไม่ลดมาก คือมีเปอร์เซ็นต์การทำงานของ Motion Compensation ทั้งหมดที่ 56.19 % ดังรูป



รูปที่ 4.8 แสดงการทำงานของฟังก์ชันต่างๆหลังจากการทำงานของ Motion Compensation ซึ่งจำเป็นต้องหาวิธีการลดการทำงานของฟังก์ชัน Motion Compensation ให้ลดลงให้มากกว่านี้

3. **การคำนวณค่า Half-pel ทั้งหมด**

จากการทดลองพบว่าสามารถทำการลดการทำงานของของตปรแกรมโดยรวมทั้งหมดลงได้มากกว่าวิธีอื่น โดยสามารถลดการทำงานของฟังก์ชัน Motion Compensation ได้ที่ 54.18 %

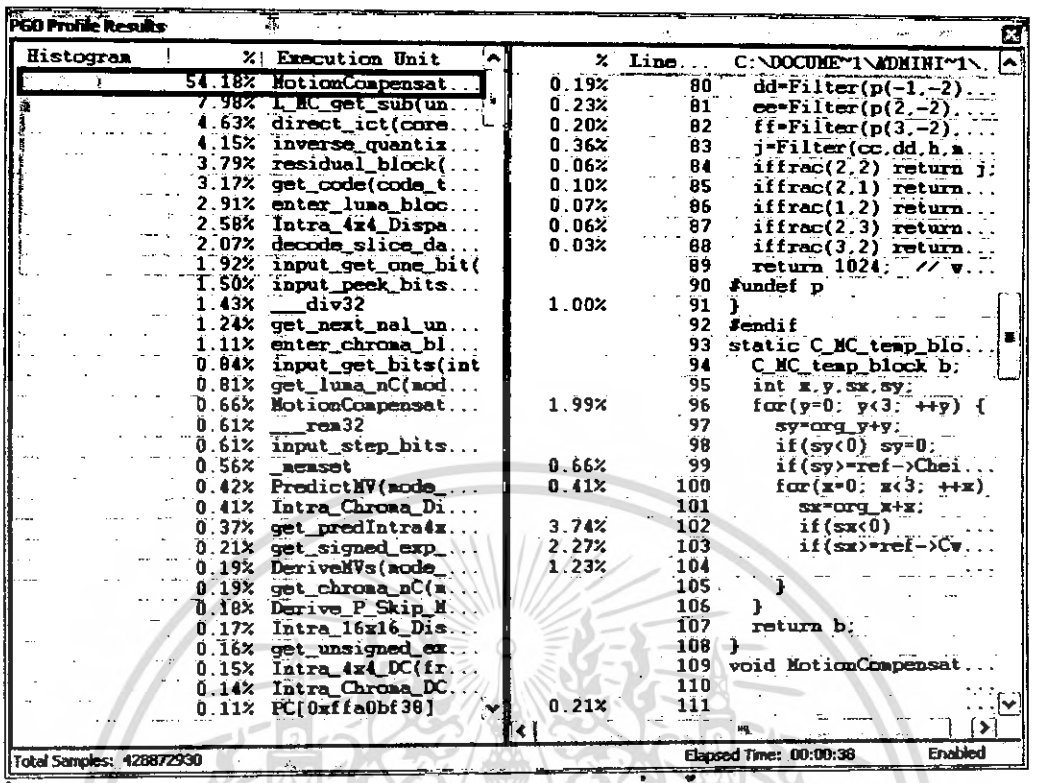


รูปที่ 4.9 แสดงเปอร์เซ็นต์การทำงานของฟังก์ชันหลังใช้วิธีการคำนวณค่า Half-pel ทั้งหมด

ซึ่งถือว่าทำได้ดีกว่าวิธีอื่นที่ผ่านมา แต่อย่างไรก็ตามโดยรวมแล้วโปรแกรมก็ยังมีการทำงานที่หนักอยู่เช่นเดิม จำเป็นต้องมีการพัฒนาโปรแกรมต่อไป

4. การคำนวณ pixel ที่ไม่ซ้ำในการคำนวณ Half-pel

การทดลองนี้ใช้วิธีปรับแต่งค่าพิกเซลที่โหลดจากเฟรมอ้างอิงใส่ไว้ใน Array เพื่อโหลดค่าที่ไม่ซ้ำกันมา แต่ก็ไม่มีผลต่อการทำงานของโปรแกรมมากนัก ทำให้การวัดประสิทธิภาพของโปรแกรมในฟังก์ชันต่างๆจึงไม่มีอะไรแตกต่างไปจากเดิมเลย ดังรูป



รูปที่ 4.10 แสดงการทำงานของฟังก์ชันหลังใช้วิธีคำนวณ pixel ที่ไม่ซ้ำในการคำนวณ Half-pel

จะเห็นว่าการทำงานของโปรแกรมไม่ได้แตกต่างไปจากเดิมเลย วิธีนี้จึงไม่ได้ผลในการทดลองนี้

สรุปผลการแก้ไขโปรแกรมได้ดังตาราง

ตารางที่ 4.1 แสดงผลการเปรียบเทียบของเวลาที่ใช้ในการรันหลังจากพัฒนาโปรแกรม

เทคนิคในการเพิ่มความเร็ว	เปอร์เซ็นต์การทำงานของ Motion Compensation
1. การสร้างขอบ	57.81 %
2. การคำนวณค่า Half-pel ทั้งหมด	56.19 %
3. การลดการทำงานของ Motion Compensation	54.18 %
4. การคำนวณ pixel ที่ไม่ซ้ำในการคำนวณ Half-pel	54.18 %

4.3 วิเคราะห์และสรุปผลการทดลอง

จากผลการทดลองสามารถสรุปได้ว่าการทำการพัฒนาโปรแกรมมาตรฐาน H.264 ลงบน Blackfin นี้มีการทำงานที่ค่อนข้างหนักและซับซ้อนมาก จึงส่งผลต่อเวลาในการประมวลผลโปรแกรม ทำให้ประมวลผลโปรแกรมนานมาก โดยได้มีการพัฒนาในส่วนของการเปลี่ยนแปลงและจัดรูปแบบ

การจัดการ memory ใหม่ และในส่วนของตัวโปรแกรมได้ใช้อัลกอริทึมต่างๆมาช่วยในการพัฒนาโปรแกรมด้วยซึ่งท้ายที่สุดแล้วสามารถพัฒนาความเร็วได้เต็มที่ที่ 0.9 เฟรมต่อวินาทีสำหรับภาพ 320 x 240 เท่านั้น

บทที่5
บทวิจารณ์และสรุป

5.1 บทสรุป

มาตรฐาน H.264 นี้เป็นมาตรฐานที่ดีมากมาตรฐานหนึ่งในการบีบอัดภาพวิดีโอสามารถทำการลดขนาดการบีบอัดภาพวิดีโอลงได้มากหลายเท่า โดยที่ยังไม่มีมาตรฐานใดสามารถบีบอัดได้มากเท่ามาตรฐาน H.264 ในปัจจุบันนี้ เนื่องจากมีการทำงานที่ซับซ้อนและละเอียดจึงทำให้ขนาดภาพวิดีโอมีขนาดเล็กลงมากได้โดยที่ไม่ทำให้คุณภาพของภาพวิดีโอที่ได้เสียหายมากนัก แต่การทำงานที่ซับซ้อนนี้เอง ส่งผลทำให้ในการพัฒนาโปรแกรม H.264 ลงบน Blackfin DSP Board นี้สามารถทำการพัฒนาได้จำกัด รวมถึงข้อจำกัดของ DSP Board ต่างๆในเรื่องของหน่วยความจำและความเร็วในการประมวลผล ซึ่งโครงงานนี้ได้ทำการแก้ไขและพัฒนาในส่วนของโปรแกรมและการจัดเรียงหน่วยความจำให้เหมาะสม เพื่อหวังว่าจะได้โปรแกรมที่รองรับการทำงานของมาตรฐาน H.264 ที่ดีขึ้น โดยมีอัลกอริทึมต่างๆที่ใช้มากมาย ไม่ว่าจะเป็นการจัดเรียงหน่วยความจำในส่วนของ Internal Memory ไปยัง External Memory เพื่อให้เพียงพอต่อการประมวลผลแล้วยังมีการเพิ่มขนาด Heap ให้มากขึ้นเพื่อความเร็วในการประมวลผลที่ดีขึ้น นอกจากนี้ยังทำการปรับปรุงตัวโปรแกรมเพื่อลดการทำงานลงด้วยอัลกอริทึมต่างๆ แต่ทั้งนี้ประสิทธิภาพการทำงานเมื่อทำการพัฒนาทั้งหมดแล้ว สามารถทำการพัฒนาได้เต็มที่ที่ 0.9 เฟรมต่อวินาทีเท่านั้นสำหรับภาพขนาด 320 x 240 เท่านั้น

5.2 แนวทางในการพัฒนาต่อ

สิ่งที่ควรจะพัฒนาต่อไปในการทำโปรแกรม H.264 นี้เป็นในเรื่องของประสิทธิภาพในการทำงานที่ยังมีการทำงานที่ช้าอยู่ ในส่วนหลักที่ทำงานหนักคือการทำ Motion Compensation และ การทำ Inverse Transform ซึ่งถ้าใน 2 ส่วนนี้ในอนาคตมีการพัฒนาให้มีประสิทธิภาพการทำงานที่ดียิ่งขึ้นจะทำให้การทำงานโดยรวมของโปรแกรมทำงานได้เร็วขึ้นด้วย

นอกจากนี้ ส่วนของฟังก์ชันการทำงานที่ขาดไปของโปรแกรม H.264 นี้คือในส่วนของการทำ Deblocking Filter ซึ่งเป็นส่วนที่ช่วยในเรื่องของเส้นขอบรูป ซึ่งเป็นส่วนเพิ่มเติมเพื่อให้รายละเอียดภาพวิดีโอมีความสวยงามมากขึ้นแต่เนื่องจากประสิทธิภาพในการทำงานที่ช้า จึงยังไม่มีการพัฒนาในส่วนนี้ ดังนั้นต่อไปในอนาคต ถ้ามีการพัฒนาการทำงานโดยรวมของโปรแกรมให้เร็วขึ้นได้แล้ว ควรมีการทำฟังก์ชัน Deblocking Filter เพิ่มขึ้นมาด้วย เพื่อประสิทธิภาพในการถอดรหัสภาพวิดีโอที่ดีขึ้น

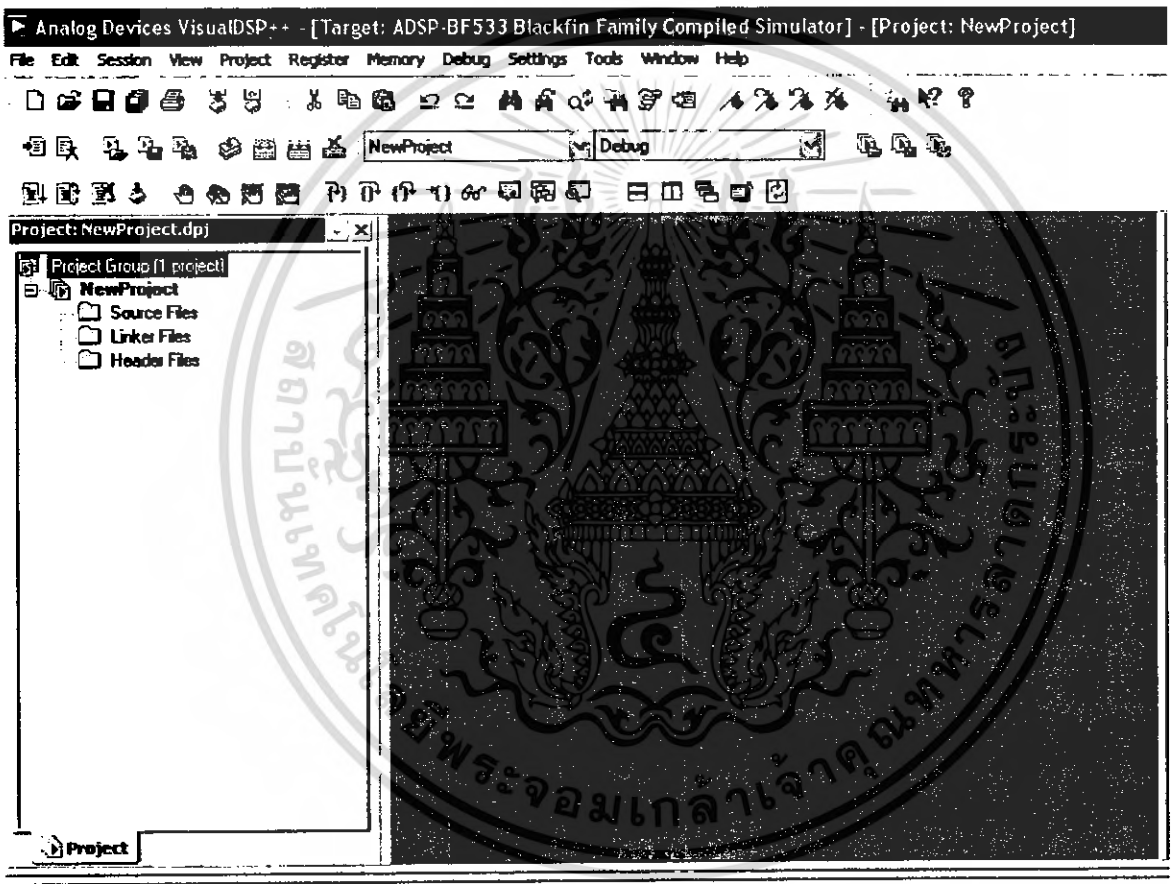
ภาคผนวก

คู่มือผู้ใช้

ในโครงงานนี้ได้ใช้โปรแกรม Visual DSP ของ Analog Device ในการจำลองการทำงานของ Blackfin DSP Board ซึ่งสามารถทำการจำลองได้ทั้งในส่วนของการคอมไพล์ รัน บันทึกโพรไฟล์ และการทำงานต่างๆ เปรียบเสมือนการทำงานบนบอร์ด DSP จริงๆ ซึ่งในส่วนนี้จะเป็นการอธิบายถึงวิธีการใช้งานโปรแกรม Visual DSP ของ Analog Device ตั้งแต่ขั้นเริ่มต้นจนถึงการใช้งานในโครงงานนี้

เริ่มต้นโปรแกรม

เมื่อทำการเปิดโปรแกรมมาจะมีลักษณะดังรูป

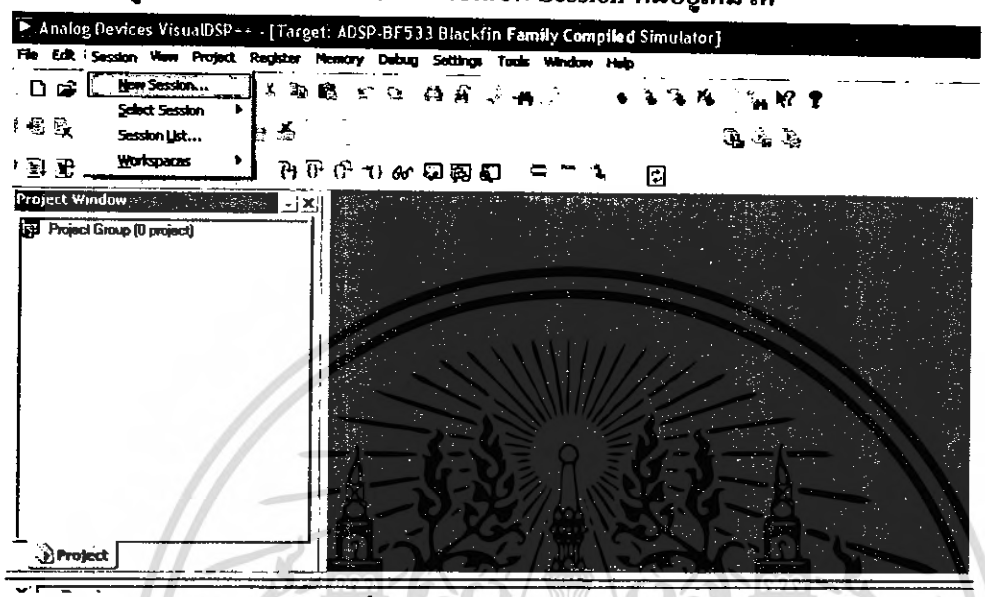


รูปที่1 แสดงหน้าจอเมื่อทำการเปิดโปรแกรม Visual DSP ของ Analog Device

โดยจะเป็น Project ที่ว่างเปล่าและต้องทำการกำหนดค่าการเริ่มต้นการทำงานต่างๆ

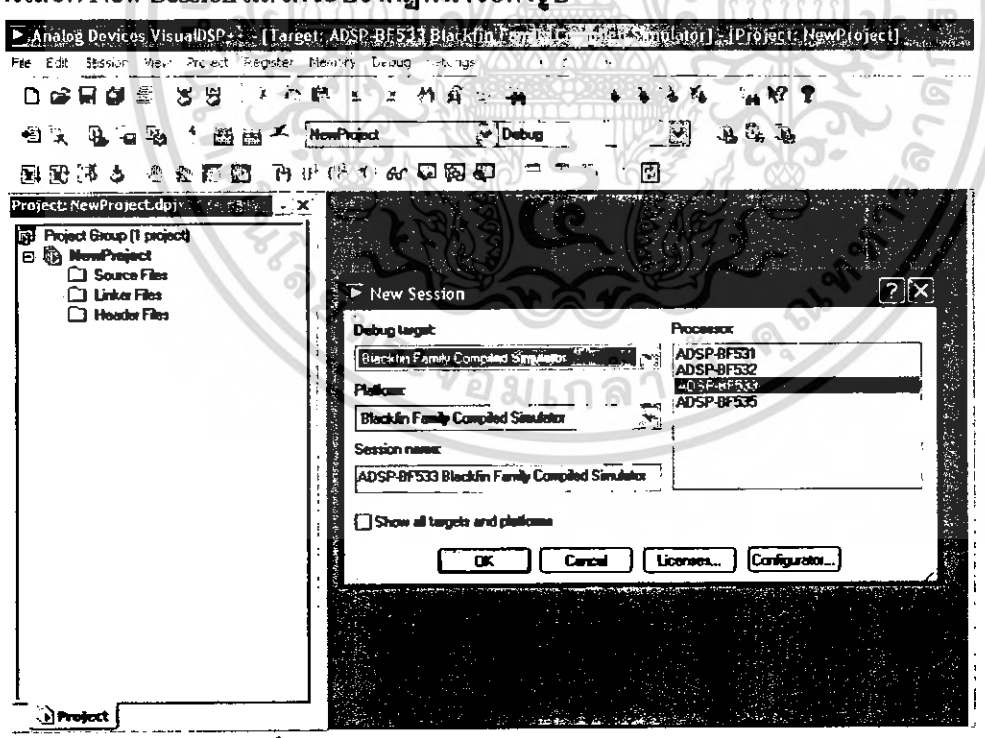
การเลือก Session

เป็นการเลือกชนิดของ DSP Board และเลือก รุ่นที่จะใช้ของ DSP Board นั้นๆ ทำได้โดยการเลือกคำสั่ง Session ที่ เมนูบาร์ และเลือก New ในกรณีที่ไม่เคยเลือก Session มาก่อน แต่ถ้าต้องการเลือก Session ที่มีอยู่ก็ใช้คำสั่ง Select Session เพื่อเลือก Session ที่มีอยู่เดิมได้



รูปที่ 2 การทำ New Session

เมื่อทำการเลือก New Session แล้วก็จะปรากฏหน้าจอดังรูป

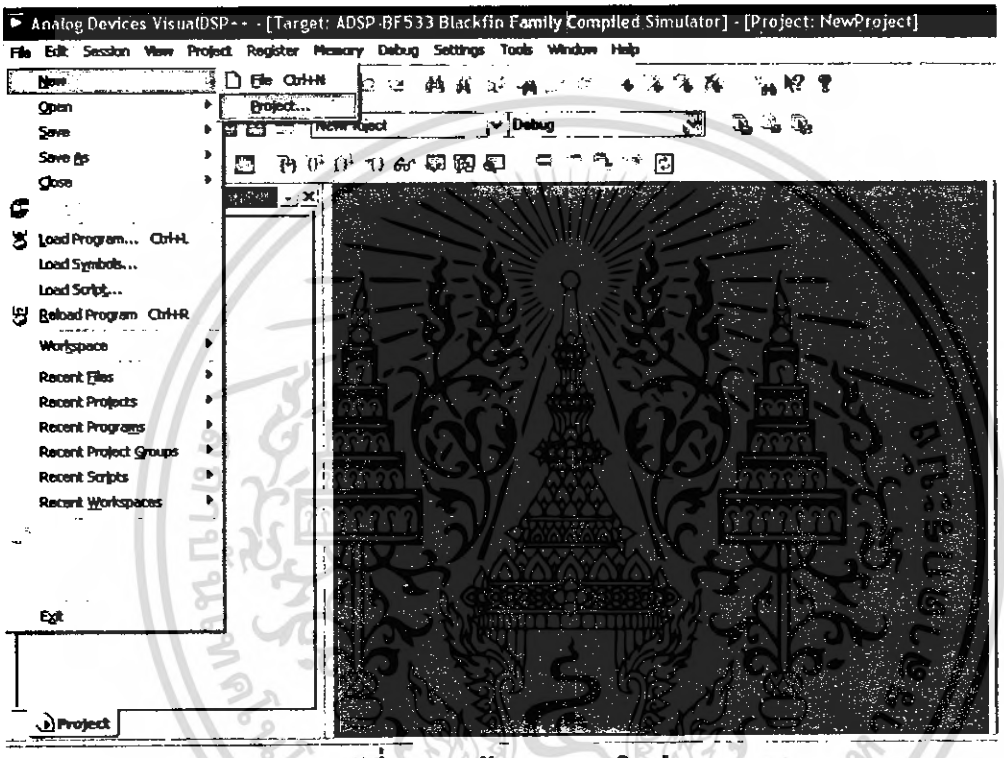


รูปที่ 3 การเลือกรุ่นของ Blackfin DSP Board

ซึ่งในโครงงานนี้ใช้ ADSP-BF533 โดยกำหนด Debug Target เป็น Blackfin Family Compiled Simulator ในการใช้จำลองการทำงานของโปรแกรมบน Blackfin DSP Board

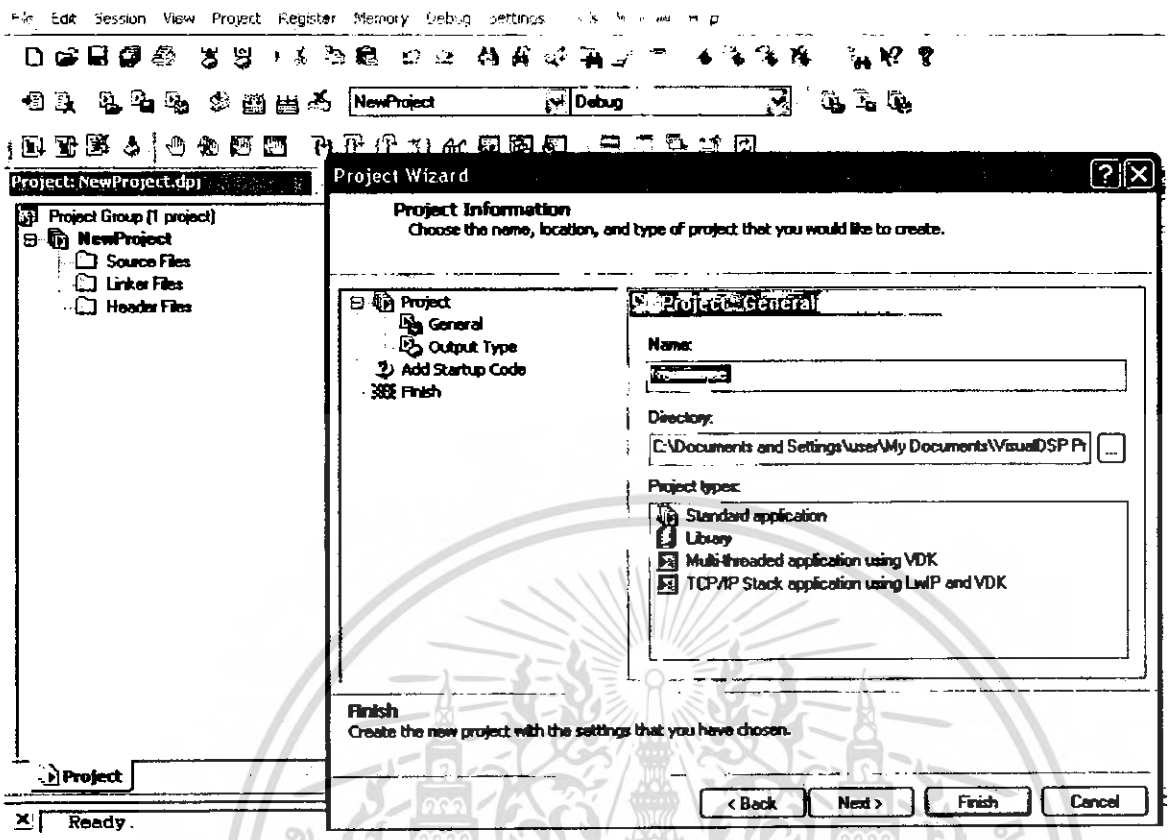
การสร้าง Project

เมื่อทำการเลือก Session แล้ว จึงทำการสร้าง Project โดยทำการเลือก คำสั่ง File -> New -> Project Wizard บนเมนูบาร์



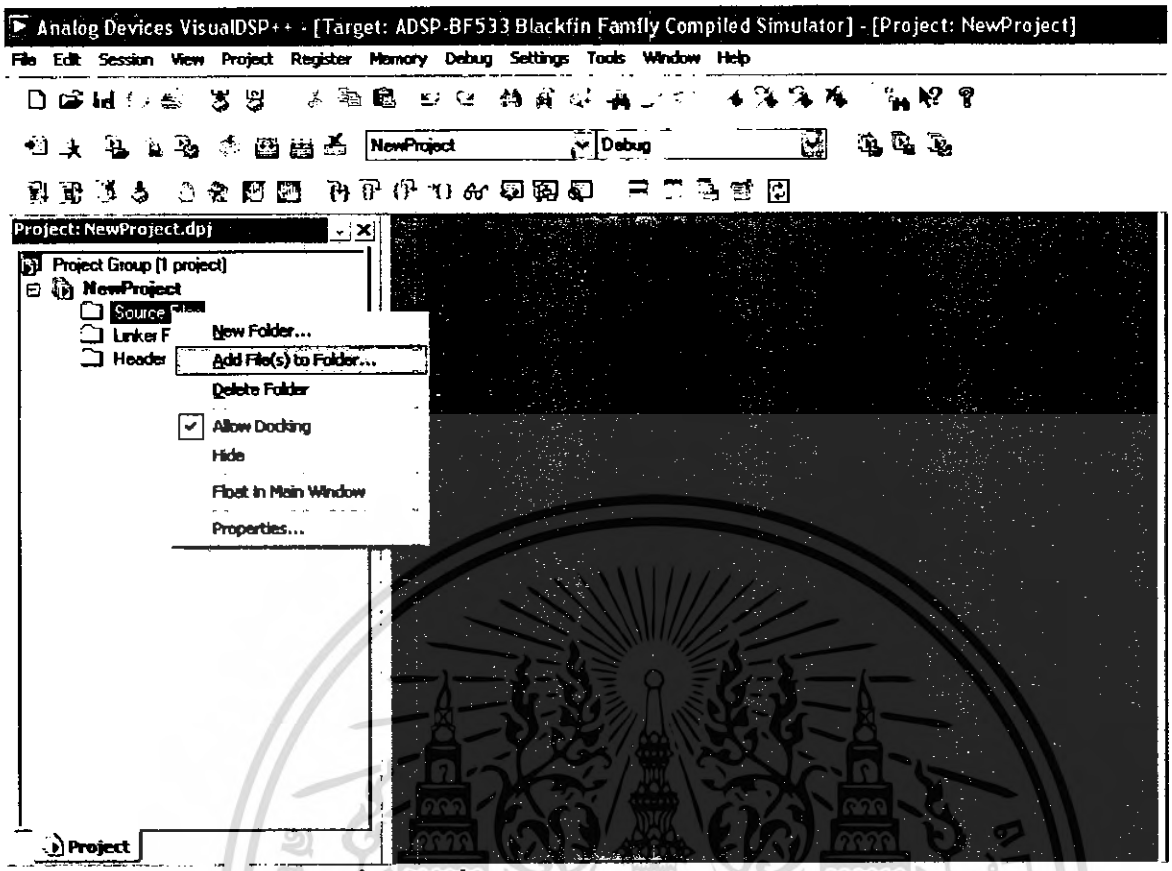
รูปที่ 4 การสร้าง Project ใหม่

เมื่อเลือกแล้วจะปรากฏหน้าจอเพื่อทำการกำหนดค่าเริ่มต้นต่างๆของ Project ใหม่ ซึ่งเมื่อทำการกำหนดทุกอย่างครบตามต้องการแล้วจึงกดปุ่ม Finish



รูปที่ 5 การสร้าง Project ใหม่ขั้นตอนที่2

เมื่อสร้าง Project ได้แล้วก็จะปรากฏProject เปล่าขึ้นมาจากนั้นทำการ Add Source Code ที่ต้องการจะนำมาจำลองการทำงานบน DSP Board นี้

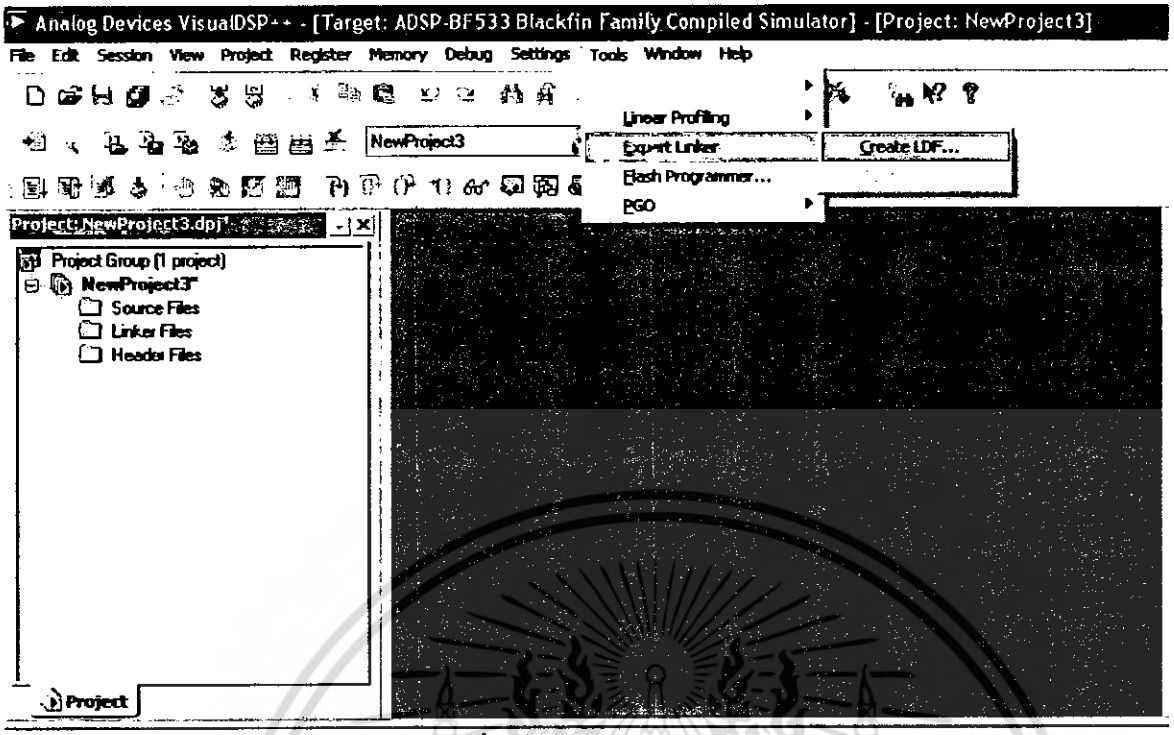


รูปที่ 6 การเพิ่มไฟล์ Source Code ลงใน Project

ในการ Add นี้ให้ทำการแยกชนิดของ File ด้วย เช่นในส่วนของตัว Source Code ก็เก็บไว้ที่ Source File ในส่วนของ Header ก็เก็บไว้ที่ Header File และในส่วนของ Linker File นั้นจะทำการเก็บไฟล์ที่ทำการกำหนดค่าของหน่วยความจำต่างๆเอาไว้ ซึ่งมีได้เพียงไฟล์เดียวในการทำงานเท่านั้น

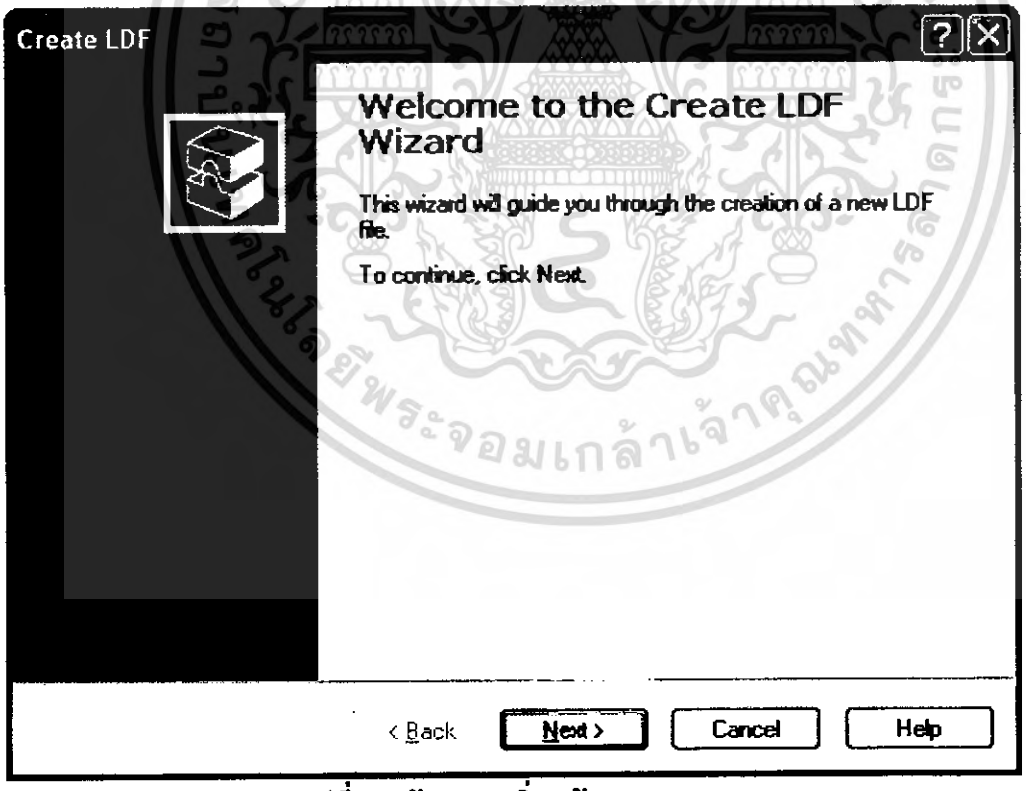
ขั้นตอนการสร้างไฟล์ LDF

ในส่วนของ Linker File นี้สามารถสร้างได้โดยไปที่คำสั่ง Tool บนเมนูบาร์ และเลือก Expert Linker และเลือก Create LDF

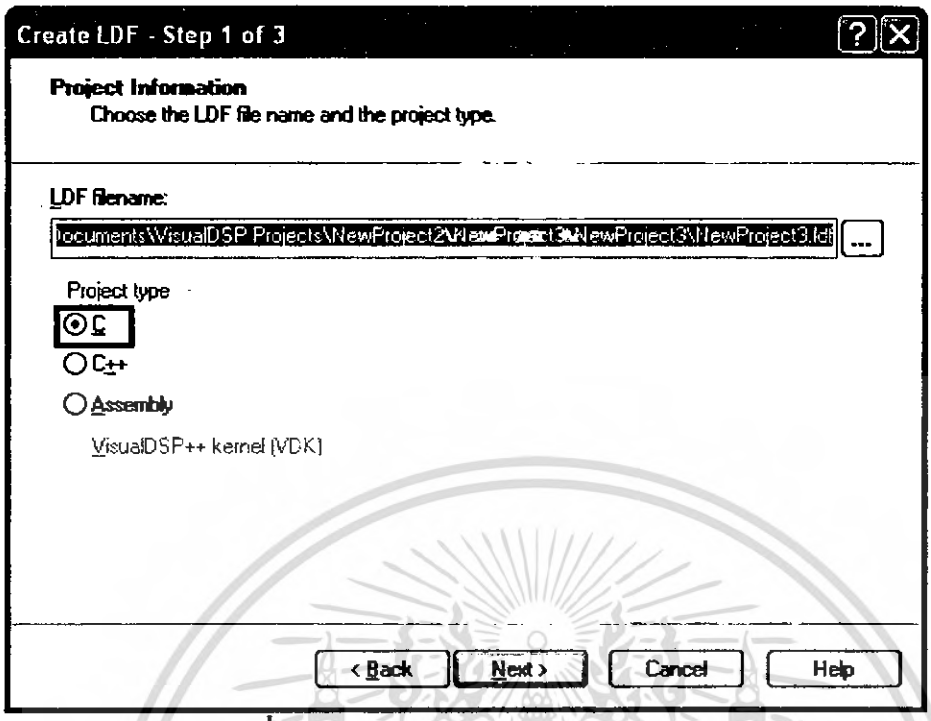


รูปที่ 7 การสร้าง LDF File

จากนั้นจะเข้าสู่หน้าจอการสร้าง Linker File ดังรูปให้ทำการกดปุ่ม Next เพื่อไปหน้าถัดไป

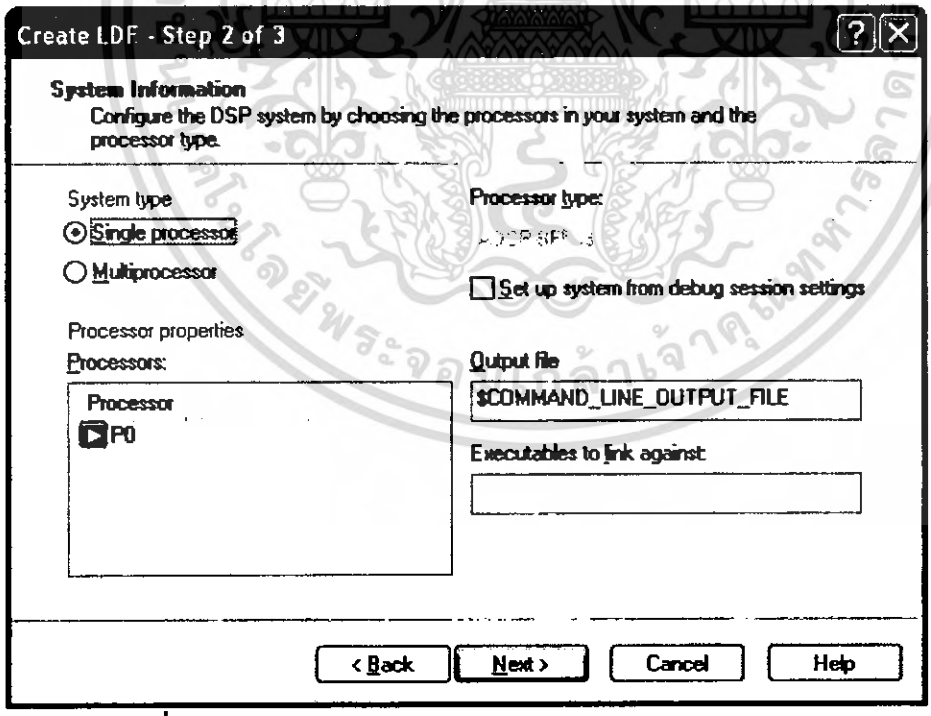


รูปที่ 8 หน้าจอการเริ่มสร้าง LDF File



รูปที่ 9 การเลือกชนิด Project ใน LDF File

ในหน้านี้จะทำการเลือกชนิดของ Project ซึ่งมีให้เลือก 3 ชนิดด้วยกัน คือภาษา C, C++, Assembly ซึ่งในโครงงานนี้ใช้ภาษา C ในการพัฒนาโปรแกรมจึงเลือกที่ C จากนั้นกดปุ่ม Next เพื่อไปหน้าถัดไป

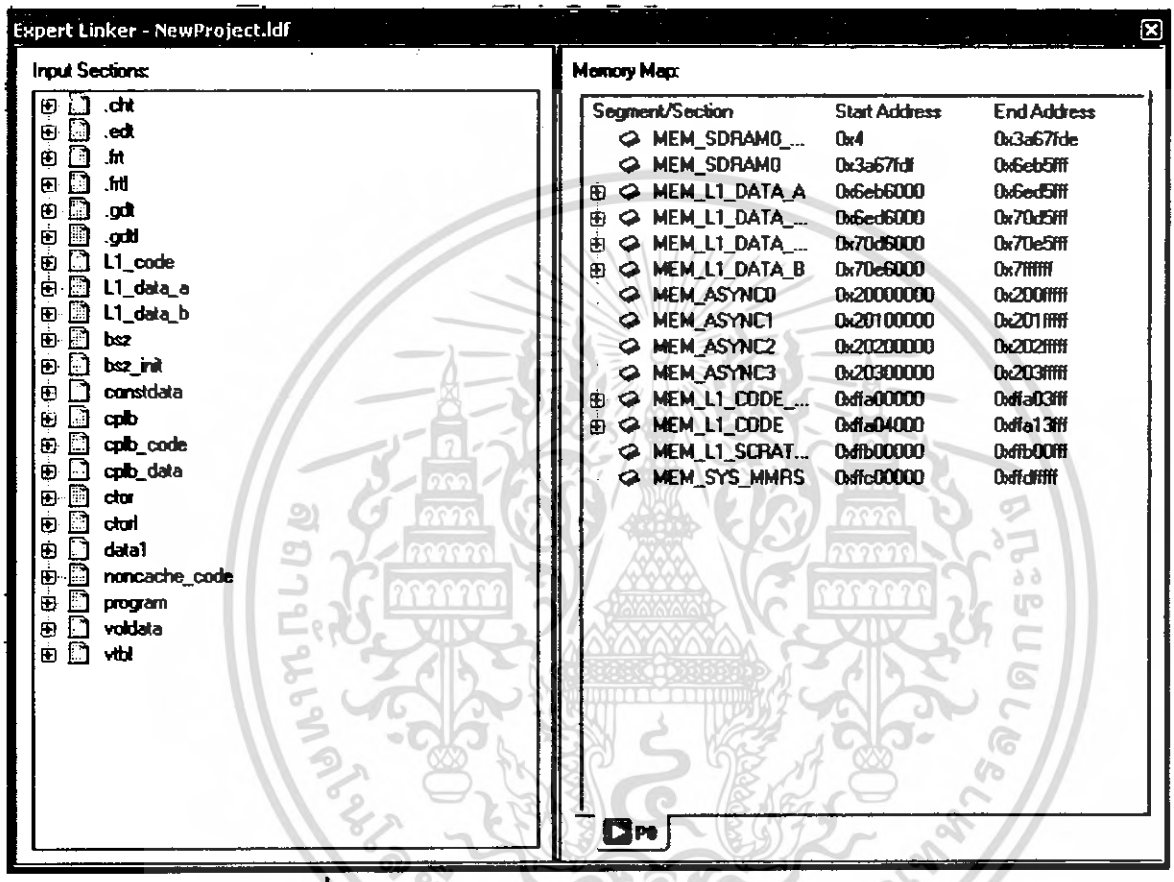


รูปที่ 10 การเลือกชนิดของระบบของ Processor ใน LDF File

หน้านี้จะเป็นการเลือกระบบของ Processor เมื่อเลือกครบทุกอย่างแล้วก็กด Next ไปหน้าถัดไปก็จะจบการสร้าง LDF File

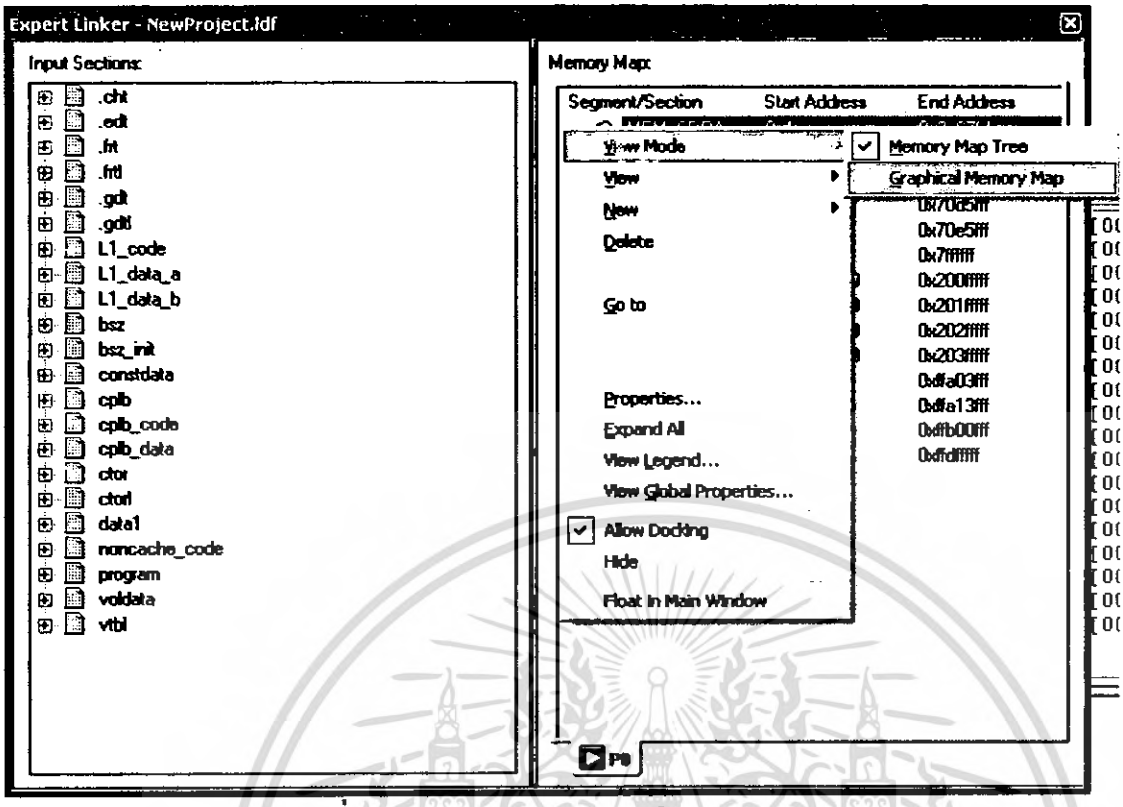
การใช้งาน LDF File

เมื่อต้องการจัดการ File LDF ที่สร้างขึ้น ให้ทำการ ดับเบิ้ลคลิกที่ไฟล์ LDF ที่สร้างขึ้นจะเป็นการแสดงการจัดเรียง Memory ในส่วนต่างๆ ดังรูป



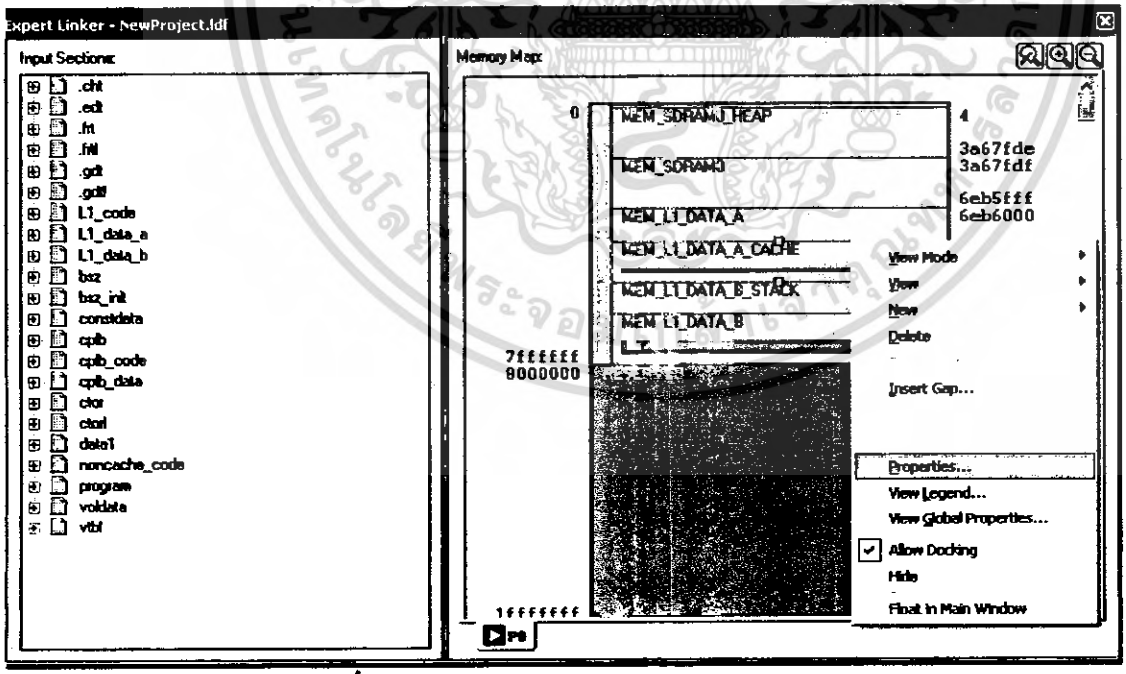
รูปที่ 11 การแสดงผลของ LDF File ใน Text Mode

จากรูปเป็นหน้าจอในการแสดงผลการจัดเรียงหน่วยความจำแบบ Text Mode ซึ่งสามารถทำการเปลี่ยน Mode ให้เป็น Graphic Mode ได้เพื่อให้ง่ายต่อการทำงานดังนี้



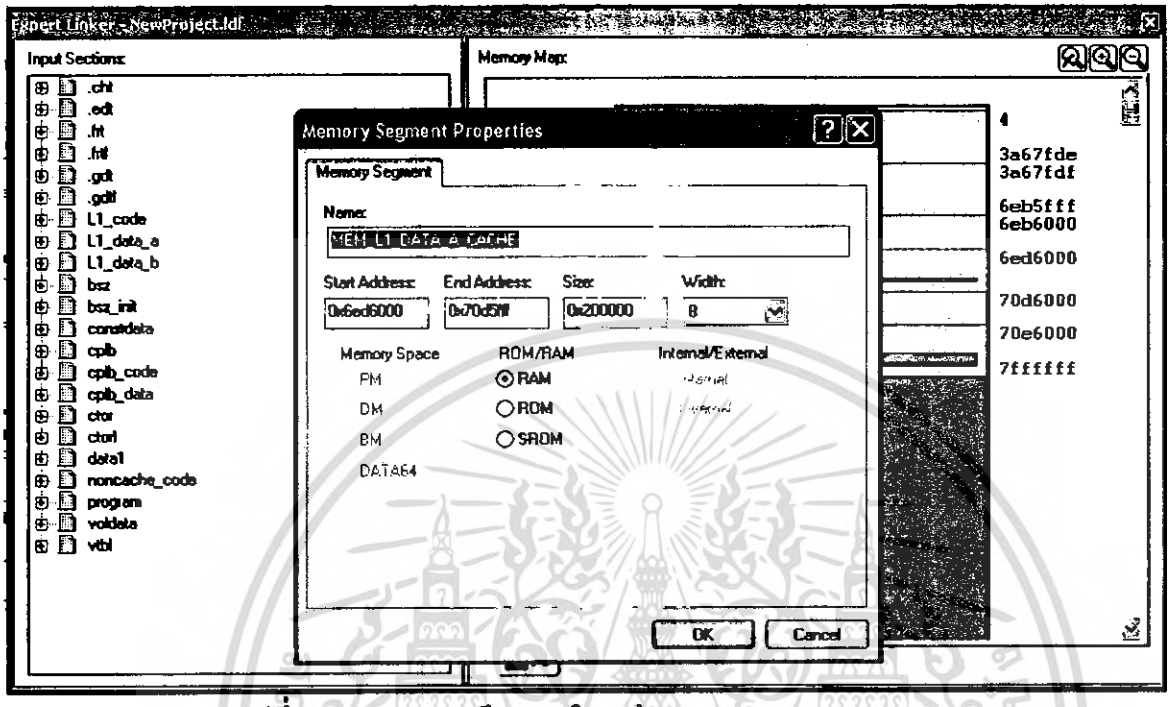
รูปที่ 12 การแสดงผล LDF File ใน Graphic Mode

ซึ่งทำการคลิกขวาที่ Memory Map ในหน้าจอด้านขวา แล้วเลือก View Mode และเลือก Graphical Memory Map จะได้การแสดงผลแบบ Graphic ดังรูป



รูปที่ 13 การจัดการ LDF File ใน Graphic Mode

ซึ่งสามารถทำการคลิกขวาที่ส่วนต่างๆของ Memory เพื่อทำการดูรายละเอียด แก้ไขรายละเอียด หรือทำการย้ายส่วนของหน่วยความจำได้ (โดยการคลิกลากฟังก์ชันที่ต้องการมาไว้ในพื้นที่หน่วยความจำที่เหมาะสมได้)

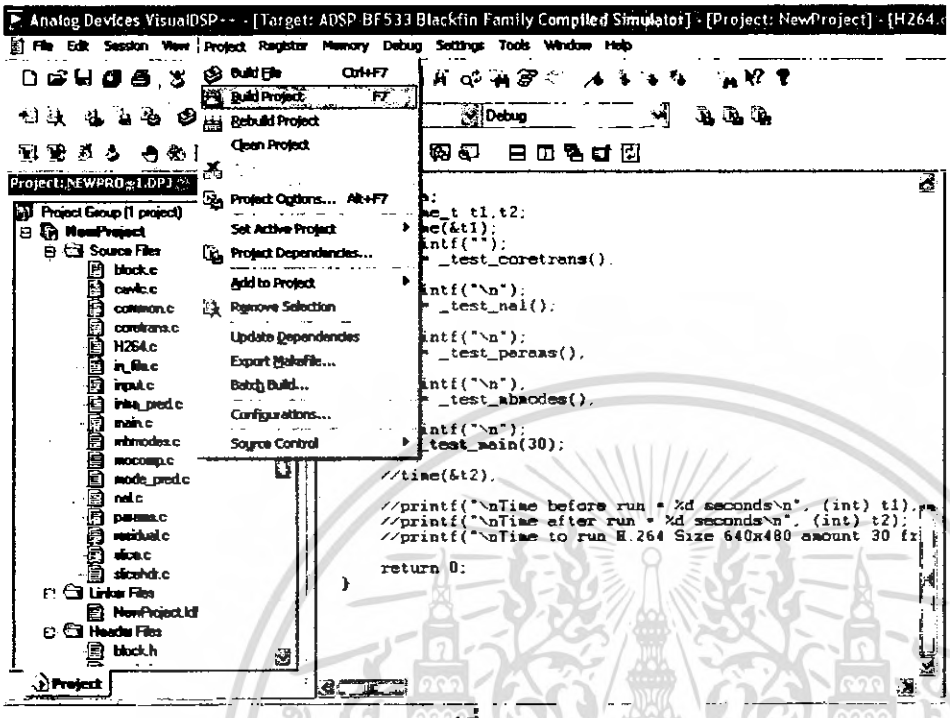


รูปที่ 14 การดูรายละเอียดภายในหน่วยความจำของ LDF File

รูปนี้เป็นตัวอย่างการดูรายละเอียดของหน่วยความจำในส่วนต่างๆ

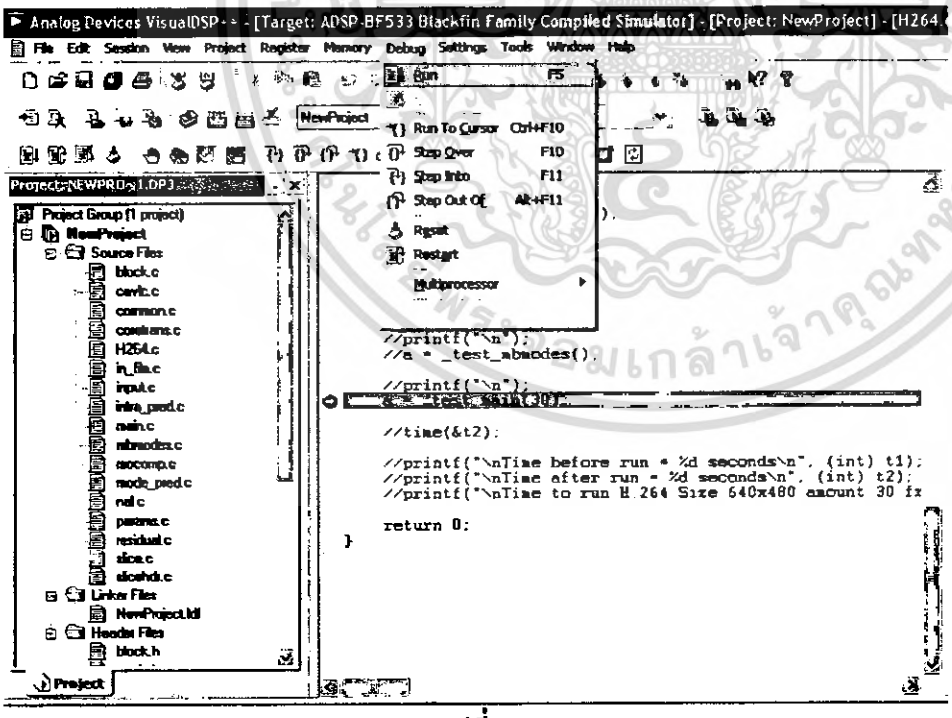
การรันโปรแกรม

เริ่มจากทำการ Build Project ก่อนดังรูป



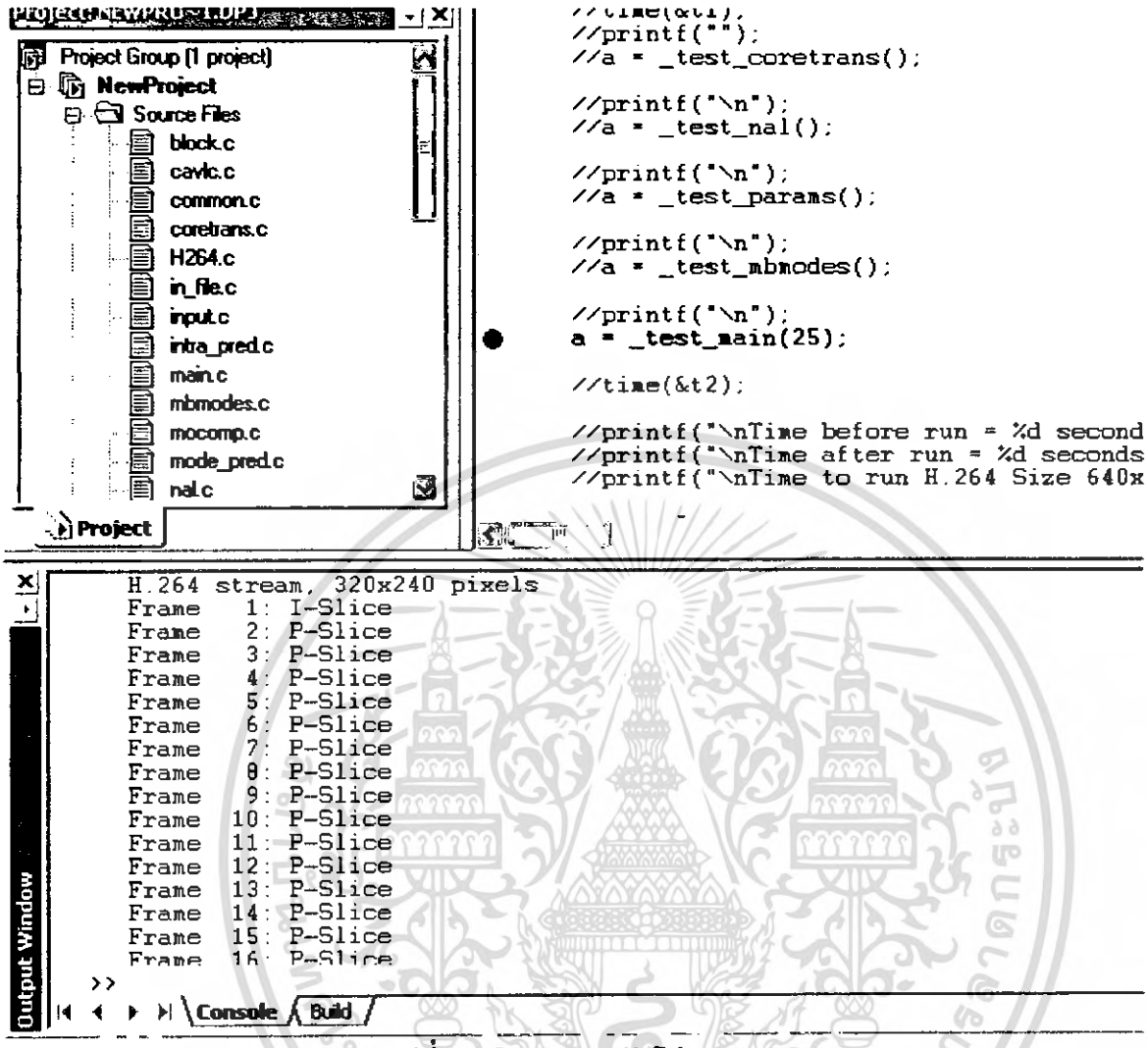
รูปที่ 15 การ Build Project

จากนั้นทำการ Run Project ดังรูป



รูปที่ 16 การ Run Project

จะได้การแสดงผลการรันดังนี้



รูปที่ 17 แสดงผลการรันโปรแกรม

ซึ่งการทำงานของโปรแกรม Visual DSP Board ของ Analog Device ที่ใช้ในโครงงานก็มีหลักการทำงานคร่าวๆดังที่กล่าวมาข้างต้นนี้

เอกสารอ้างอิง

[1] Iain E.G. Richardson. 2004. **H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia** , Aberdeen , Robert Gordon University

[2] Soon-kak Kwon, A. Tamhankar, K.R. Rao. **Overview of H.264 / MPEG-4 Part 10** , University of Texas at Arlington , Dongeui University

[3] MPEG Home Page. "MPEG-4 Discription." [Online]. Available :http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm. 2002.

[4] Martin Feidler. "Implementation of a basic H.264/AVC Decoder" Chemnitz University of Thechnology,June,2004

[5] Muhammad Owais Khan, Umar Khan, Sardar Adnan Rahim and Syed Irtiza Ali. "Optimization of Motion Compensation for H.264 Decoder by Pre-Calculation" National University of Sciences and Technology