เครื่องกำจัดผีในสัญญาณโทรทัศน์โดยใช้วงจรกรองเอฟไออาร์แบบเอฟพีจีเอ

นายพหล ศิริเหลืองทอง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544

ISBN 974-03-1138-5

A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS

Mr.Pahol Siriluangtong

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2001

ISBN 974-03-1138-5

หัวข้อวิทยานิพนธ์	เครื่องกำจัดผีในสัญญาณโทรทัศน์โดยใช้วงจรกรองเอฟไออาร์แบบเอฟพีจีเอ
โดย	นายพหล ศิริเหลืองทอง
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	อาจารย์ ดร.วันเฉลิม โปรา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(รองศาสตราจารย์ ดร.เอกชัย ลีลารัศมี)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.วันเฉลิม โปรา)

.....กรรมการ
(อาจารย์สุวิทย์ นาคพีระยุทธ)

พหล ศิริเหลืองทอง : เครื่องกำจัดผีในสัญญาณโทรทัศน์โดยใช้วงจรกรองเอฟไออาร์แบบเอฟพีจีเอ. (A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS) อ. ที่ปรึกษา : ดร.วันเฉลิม โปรา, 129 หน้า. ISBN 974-03-1138-5.

วิทยานิพนธ์นี้นำเสนอเครื่องกำจัดผีในสัญญาณโทรทัศน์ระบบ PAL ที่ใช้สัญญาณอ้างอิงสำหรับกำจัดผี (GCR Signal) ตามมาตรฐาน ITU-R BT.1124 ส่วนประกอบสำคัญของเครื่องกำจัดผีคือ วงจรกรองเอฟไออาร์แบบปรับตัว (Adaptive FIR Filters) ซึ่งพัฒนาบนชิพเอฟพีจีเอของบริษัท Xilinx เบอร์ XCV300E-6 เทคนิคการใช้ทรัพยากรร่วม (Resource Sharing) ช่วยประหยัดทรัพยากรที่ใช้สร้างวงจรกรองได้มากถึง 6 เท่า การคำนวณหาสัมประสิทธิ์ของวงจรกรองแบบปรับตัวใช้ชิพประมวลผลสัญญาณดิจิตอล (DSP Chip) ของบริษัท Texas Instruments เบอร์ TMS320C6211 เลือกใช้กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด (LMS : Least Mean Square) ในการคำนวณหาค่าสัมประสิทธิ์ของวงจรกรองที่เหมาะสม เครื่องจำลองการเกิดผีอย่างง่ายได้ถูกพัฒนาขึ้นด้วยการใช้แท็ปสายประวิง (Delay Line) เพื่อใช้ในการทดสอบการทำงานของเครื่องกำจัดผีในสัญญาณโทรทัศน์ ผลการทดสอบพบว่าสามารถกำจัดผีได้ภายในเวลา 4 วินาที เมื่อผีมีความแรงของสัญญาณไม่เกิน -6 dB และมีเวลาประวิงไม่เกิน 10 μs เทียบกับสัญญาณหลัก

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2544

4270448321 : MAJOR ELECTRICAL ENGINEERING

KEY WORD : GHOST CANCELLER / GCR SIGNAL / Adaptive FIR FILTERS / FPGA / RESOURCE SHARING

PAHOL SIRILUANGTONG : A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS. THESIS ADVISOR : WANCHALERM PORA, Ph.D., 129 pp. ISBN 974-03-1138-5.

This thesis presents a PAL TV ghost canceller that employs a ghost cancellation reference signal (GCR Signal) recommended by the ITU-R BT.1124 standard. The ghost canceller consists of an adaptive FIR filter, which is developed on a Xilinx FPGA, XCV300E-6. The resource sharing technique is exploited. It reduces required hardware resource by six times. A DSP chip, TMS320C6211, is chosen to optimize the parameters of the filter according to the LMS (Least Mean Square) algorithm. We also developed a ghost generator using delay line for testing the TV ghost canceller. Experiments show that the ghost canceller can visually cancel the ghost in 4 seconds if the ghost power is not more than -6 dB and its delays are not more than 10 μs, compared to that of the main signal.

Department Electrical Engineering Student's signature .....

Field of study Electrical Engineering Advisor's signature .....

Academic year 2001 .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของอาจารย์ ดร.วันเฉลิม โปรา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่างๆ ในการวิจัยด้วยดีตลอดมา

ขอขอบคุณ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์ และคอมพิวเตอร์แห่งชาติ (NECTEC) และโครงการเงินก้นถุงของจุฬาลงกรณ์มหาวิทยาลัย ที่ได้ให้ทุนวิจัยสนับสนุนโครงการนี้เป็นอย่างดี

นอกจากนี้ยังมีเพื่อนๆ พี่ๆ และน้องๆ ทุกคนในห้องปฏิบัติการวิจัยระบบเชิงเลขที่คอยห่วงใยและคอยช่วยเหลือให้คำแนะนำที่ดีต่างๆ มาโดยตลอด

และขอขอบคุณ คุณปัญจภา เรืองสินทรัพย์ สำหรับกำลังใจมากมาย ทั้งยังช่วยให้คำปรึกษาและคำแนะนำต่างๆ เกี่ยวกับการเขียนบทความ และวิทยานิพนธ์

ท้ายนี้ ผู้วิจัยใคร่ขอกราบของพระคุณ บิดา-มารดา รวมทั้งพี่ๆ ทั้ง 4 คน ที่สนับสนุนในด้านการเงิน และให้กำลังใจแก่ผู้วิจัยด้วยดีเสมอมา

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 แนวเหตุผลในการทำวิทยานิพนธ์

ปัจจุบันนี้ มีการใช้งานโทรทัศน์กันอย่างแพร่หลาย อาจเรียกได้ว่ามีใช้กันแทบทุกครัวเรือน จึงก่อให้เกิดการพัฒนาเทคโนโลยีในระบบการรับ-ส่งสัญญาณโทรทัศน์ ทั้งด้านสถานีส่ง และด้านเครื่องรับโทรทัศน์ให้มีคุณภาพของทั้งภาพ และเสียงที่ดีขึ้น แต่ปัญหาหนึ่งที่ยังพบเห็นได้บ่อยครั้ง โดยเฉพาะในเมืองใหญ่ที่มีตึกสูงเป็นจำนวนมาก คือ การเกิดภาพซ้อนบนจอภาพ เนื่องมาจากเครื่องรับโทรทัศน์ได้รับคลื่นสัญญาณมาจากหลายทิศทาง โดยในแต่ละทิศทางจะใช้เวลาในการเดินทางไม่เท่ากัน และสัญญาณที่เดินทางมาจากแต่ละทิศทางจะสูญเสียพลังงานไปไม่เท่ากัน ทำให้เมื่อสัญญาณเหล่านี้มารวมกันที่สายอากาศของเครื่องรับโทรทัศน์ จะปรากฏเป็นเงาหรือภาพซ้อน หรือที่เรียกว่า ผี (Ghost) บนจอภาพของเครื่องรับโทรทัศน์

ปัญหานี้เป็นปัญหาที่ไม่สามารถแก้ไขด้วยกรรมวิธีทางอิเล็กทรอนิกส์แบบดั้งเดิมเพียงอย่างเดียว แต่ต้องใช้กรรมวิธีในการประมวลผลสัญญาณดิจิตอล มาช่วยแก้ปัญหา ส่วนประกอบอย่างหนึ่งที่สำคัญในกระบวนการประมวลผลสัญญาณดิจิตอลคือ วงจรกรองแบบปรับตัว (Adaptive Filter) ซึ่งเป็นวงจรกรองที่สามารถเปลี่ยนค่าสัมประสิทธิ์ของวงจรกรองให้มีคุณสมบัติที่เหมาะสมกับระบบที่เปลี่ยนแปลงได้ โดยในช่วงที่ผ่านมามีการนำเสนองานวิจัยที่เกี่ยวข้องกับระบบการกำจัดผีในสัญญาณโทรทัศน์เป็นจำนวนมาก [1-10] เช่น การพัฒนาสัญญาณอ้างอิงสำหรับกำจัดผี (GCR : Ghost Cancellation Reference Signal), การพัฒนาเครื่องจำลองการเกิดผี (Ghost Generator) และการพัฒนาเครื่องกำจัดผีในสัญญาณโทรทัศน์ (Ghost Canceller) เป็นต้น

อย่างไรก็ตามในปัจจุบัน ระบบการกำจัดผียังไม่ได้รับความนิยม เนื่องจากต้นทุนการผลิตวงจรกำจัดผีค่อนข้างสูง จึงมีการใส่วงจรกำจัดผีรวมอยู่ในเครื่องรับโทรทัศน์ที่มีราคาสูงมากเท่านั้น ผู้วิจัยจึงได้เกิดแนวความคิดที่จะพัฒนาต้นแบบเครื่องกำจัดผีที่มีต้นทุนการผลิตลดลง

ด้วยเทคโนโลยีที่ก้าวหน้ามากขึ้น ทำให้สามารถสร้างวงจรต้นแบบได้อย่างรวดเร็ว โดยการใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language) เช่น ภาษาวีเอชดีแอล (VHDL) [11] เขียนอธิบายการทำงานของฮาร์ดแวร์ที่ต้องการ ซึ่งรหัส (Code) ที่เขียนขึ้นนั้น สามารถนำไปกำหนดรูปแบบการทำงานของวงจรรวมแบบหนึ่งที่เรียกว่าเอฟพีจีเอ (FPGA) ให้มีคุณลักษณะ

ตามที่ได้อธิบายเอาไว้ การออกแบบวงจรด้วยวิธีนี้เป็นที่นิยมกันมาก เพราะช่วยพัฒนาวงจรดิจิตอลได้สะดวก รวดเร็ว และมีค่าใช้จ่ายน้อย เนื่องจากซอฟต์แวร์ที่สนับสนุนการออกแบบวงจรดิจิตอลแบบนี้ มีประสิทธิภาพมากขึ้น และราคาถูกลงมาก สามารถที่จะออกแบบ แก้ไข และตรวจสอบด้วยการจำลองการทำงาน (Simulation-based Verification) โดยงานในแต่ละขั้นตอนจะทำอยู่บนเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer : PC) เมื่อวงจรมีความถูกต้องแล้วจึงโปรแกรมลงเอฟพีจีเอ เพื่อทดสอบการทำงานจริง และหากต้องการปรับปรุงแก้ไขวงจรเพิ่มเติมก็กลับไปทำตามขั้นตอนข้างต้นซ้ำใหม่ได้ไม่จำกัดจำนวนครั้ง จากข้อดีดังกล่าวจึงได้เกิดแนวความคิดที่จะนำเอฟพีจีเอมาใช้ในการสร้างวงจรกรองแบบปรับตัว ซึ่งโครงสร้างภายในประกอบไปด้วย รีจิสเตอร์, วงจรคูณ และวงจรบวก เป็นจำนวนมาก รวมทั้งวงจรดิจิตอลอื่นๆ ซึ่งเป็นส่วนประกอบเครื่องกำจัดผีในสัญญาณโทรทัศน์ที่จะพัฒนาขึ้น

งานวิจัยนี้จึงได้นำเสนอ การพัฒนาต้นแบบเครื่องกำจัดผีในสัญญาณโทรทัศน์ ที่ใช้เอฟพี-จีเอสร้างวงจรกรองแบบปรับตัว เพื่อที่จะเป็นแนวทางลดต้นทุนการผลิตเครื่องกำจัดผีลงได้ และเป็นแนวทางเริ่มต้นในการพัฒนาคุณภาพการรับชมโทรทัศน์ภายในประเทศไทยให้ดียิ่งขึ้น

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อออกแบบและพัฒนาโครงสร้างของเครื่องกำจัดผีในสัญญาณโทรทัศน์
2. เพื่อสร้างต้นแบบของเครื่องกำจัดผีในสัญญาณโทรทัศน์

1.3 ขอบเขตของการวิจัย

1. สร้างต้นแบบของเครื่องกำจัดผีในสัญญาณโทรทัศน์ที่มีคุณสมบัติ ดังนี้
   a. ใช้วงจรกรองแบบเอฟไอาร์ (FIR Filters) ในการกำจัดผี โดยสังเคราะห์ลงบนชิพเอฟพีจีเอ (FPGA Chip)
   b. ใช้กระบวนวิธีแบบค่าเฉลี่ยกำลังสองน้อยสุด (LMS : Least Mean Square Algorithm) ในการคำนวณหาค่าสัมประสิทธิ์ที่เหมาะสมในการกำจัดผี
2. สร้างเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์ที่ความถี่เบสแบนด์โดยใช้แท็ปสายประวิง (Delay Line) เพื่อใช้ในการทดสอบการทำงานของเครื่องกำจัดผี

1.4 วิธีดำเนินการวิจัย

1. ศึกษาการเกิดผีในสัญญาณโทรทัศน์ และกระบวนวิธีในการกำจัดผีในสัญญาณโทรทัศน์

2. ออกแบบ และสร้างเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์
3. ออกแบบ และสังเคราะห์วงจรกรองเอฟไอาร์ และส่วนประกอบอื่นๆ เพื่อโปรแกรมลงชิพเอฟพีจีเอ
4. ออกแบบ และเขียนโปรแกรมคำนวณค่าสัมประสิทธิ์เพื่อกำจัดผี ลงในชิพประมวลผลสัญญาณเชิงเลข
5. ประกอบวงจรในส่วนต่างๆ เข้าด้วยกัน
6. ทดสอบการทำงาน และปรับปรุงอุปกรณ์ต้นแบบ
7. สรุปผลการทดลอง และเขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. อุปกรณ์ต้นแบบของเครื่องกำจัดผีในสัญญาณโทรทัศน์ โดยใช้สัญญาณอ้างอิงสำหรับกำจัดผีตามมาตรฐาน ITU-R BT.1124 สามารถนำไปพัฒนาต่อในเชิงอุตสาหกรรมได้
2. ช่วยยกระดับคุณภาพการรับชมโทรทัศน์ภายในประเทศไทยให้สูงขึ้น

1.6 ลำดับขั้นตอนในการเสนอผลการวิจัย

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 6 บทดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึงที่มาและความสำคัญของปัญหา วัตถุประสงค์ ขอบเขต และวิธีดำเนินงานวิจัย รวมทั้งประโยชน์ที่คาดว่าจะได้รับ บทที่ 2 จะสรุปแนวความคิด ทฤษฎีที่เกี่ยวข้องกับการวิจัย และผลงานที่เคยมีมาก่อน บทที่ 3 เสนอโครงสร้างหลักภายในเครื่องกำจัดผี ในบทที่ 4 ได้อธิบายรายละเอียดของวงจรภายในชิพเอฟพีจีเอ ซึ่งประกอบด้วย 2 ส่วนหลักคือ วงจรกรองเอฟไอาร์ และวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี บทที่ 5 จะกล่าวถึงกระบวนวิธีที่ใช้ในการคำนวณ และรายละเอียดของโปรแกรมกำจัดผีที่ทำงานบนตัวประมวลผลสัญญาณดิจิตอล และบทที่ 6 เสนอผลการทดสอบประสิทธิภาพของเครื่องกำจัดผี รวมทั้งบทสรุปงานวิจัย และข้อเสนอแนะ

1.7 ผลงานที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของงานวิจัยที่ได้รับการตีพิมพ์เป็นบทความทางวิชาการ ในหัวข้อ “เครื่องกำจัดผีในสัญญาณโทรทัศน์โดยใช้วงจรกรองเอฟไอาร์แบบเอฟพีจีเอ” หน้า 1182-1187 โดยพหล ศิริเหลืองทอง และวันเฉลิม โปรา ในงานประชุมวิชาการ “การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 24 (24th Electrical Engineering Conference : EECON24)” ซึ่งจัดโดยคณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยี พระจอมเกล้าเจ้าคุณทหารลาดกระบัง ณ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ระหว่างวันที่ 22-23 พฤศจิกายน 2544

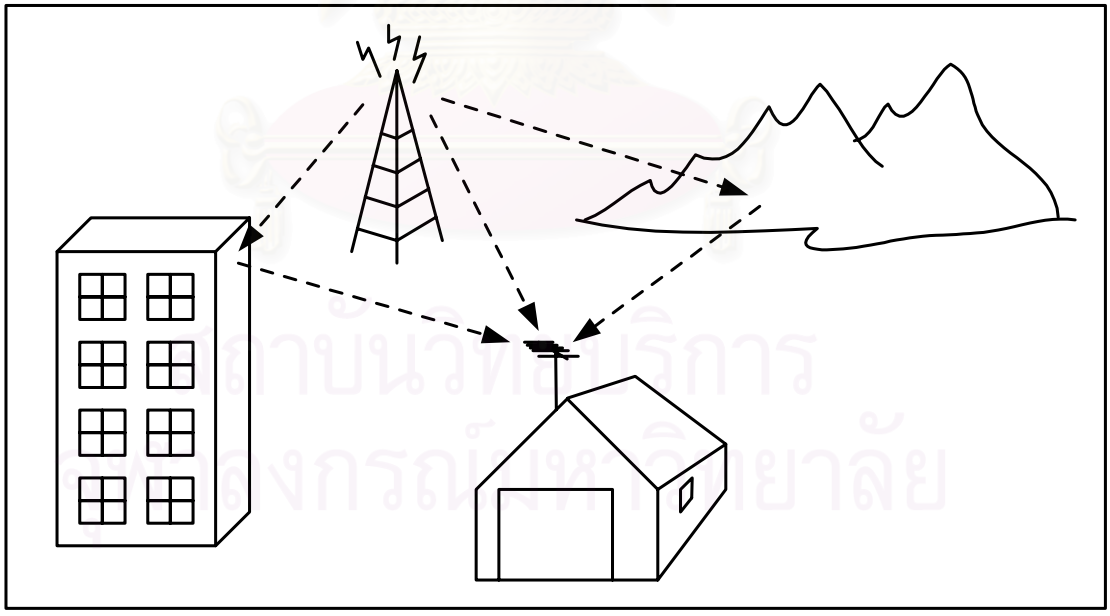
ส่วนหนึ่งของงานวิจัยที่อยู่ระหว่างการพิจารณาเพื่อตีพิมพ์เป็นบทความทางวิชาการ ในหัวข้อ “A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS” โดยวันเฉลิม โปราและ พหล ศิริเหลืองทอง ในงานประชุมวิชาการ “2002 IEEE ASIA PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS” ซึ่งจะจัดขึ้นที่ “Kartika Plaza Beach Hotel, Bali, Indonesia” ระหว่างวันที่ 28-31 ตุลาคม 2545

บทที่ 2

ความรู้พื้นฐาน และปริทัศน์วรรณกรรม

2.1 การเกิดผีในสัญญาณโทรทัศน์

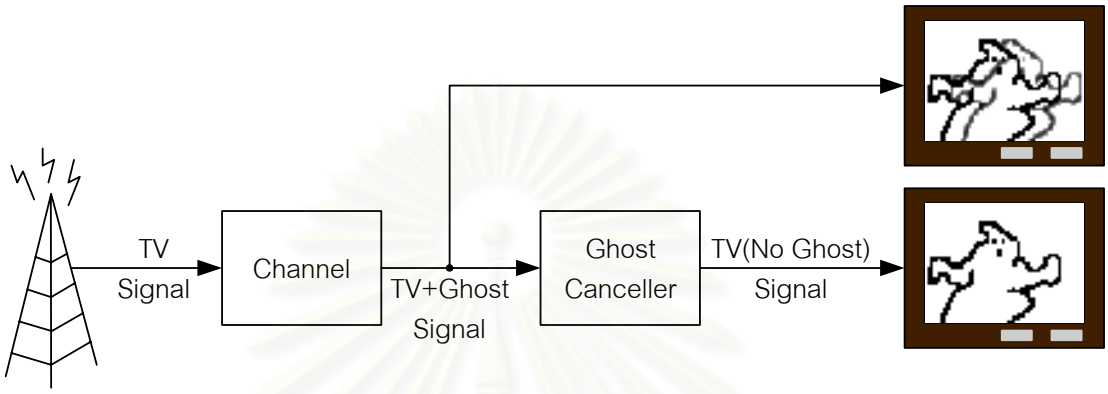
เนื่องจากสถานีโทรทัศน์ทำการส่งสัญญาณโทรทัศน์ออกมาจากเสาอากาศไปทุกทิศทาง [13] ดังรูปที่ 2.1 ในเมืองใหญ่ที่มีตึกสูง หรือสถานที่ที่มีภูเขาสูง สัญญาณโทรทัศน์ที่เครื่องรับโทรทัศน์รับได้นั้น จะเกิดจากการรวมกันของคลื่นสัญญาณที่เดินทางมาโดยตรงกับสัญญาณที่สะท้อนมาจากสิ่งกีดขวาง เช่นตึกสูง หรือภูเขา บางครั้งอาจไม่มีสัญญาณที่เดินทางมาจากสถานีโดยตรงเลย สัญญาณที่เดินทางมาในแต่ละเส้นทางนี้ จะใช้เวลาในการเดินทางไม่เท่ากัน อีกทั้งยังมีความเข้มไม่เท่ากัน เนื่องจากระยะทางที่ใช้เดินทางไม่เท่ากัน และชนิดสิ่งกีดขวางไม่เหมือนกัน ปรากฏการณ์นี้จะทำให้เกิดภาพหลักปรากฏเป็นภาพที่เข้มที่สุดบนจอภาพของเครื่องรับโทรทัศน์ อันเกิดจากสัญญาณที่มีความเข้มมากที่สุด ส่วนสัญญาณภาพที่มีความเข้มของสัญญาณน้อยกว่า จะทำให้เกิดเงาของภาพหลัก หรือที่เรียกสั้นๆ ว่า ผี (Ghost) ซ้อนอยู่บนภาพหลักดังกล่าว



รูปที่ 2.1 การแพร่กระจายแบบหลายทาง (Multipath Propagration) ทำให้เกิดผี

2.2 การกำจัดผีในสัญญาณโทรทัศน์

จากลักษณะการแพร่ภาพสัญญาณโทรทัศน์ออกมาทุกทิศทางนั้น เสมือนว่าสัญญาณโทรทัศน์ส่งผ่านช่องสัญญาณ (Channel) อันหนึ่ง [13] ซึ่งช่องสัญญาณนี้มีคุณสมบัติทำให้เกิดสัญญาณผีรวมเข้าไปกับสัญญาณภาพเดิมดังรูปที่ 2.2



รูปที่ 2.2 ช่องสัญญาณทำให้เกิดผี ซึ่งสามารถกำจัดได้โดยเครื่องกำจัดผี

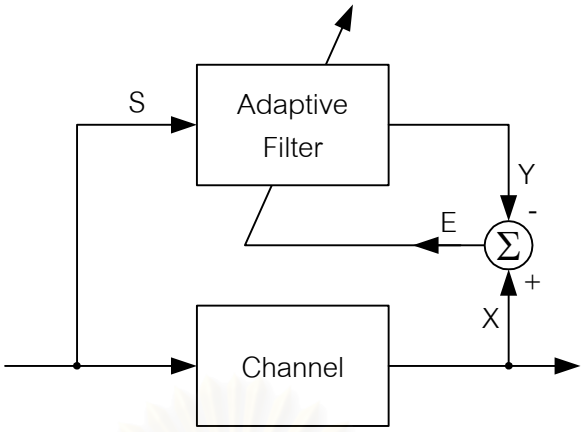
เพื่อจะกำจัดสัญญาณผีที่ถูกรวมเข้ามาในสัญญาณโทรทัศน์ ซึ่งเกิดจากช่องสัญญาณจำเป็นต้องมีอุปกรณ์ที่มีคุณสมบัติตรงกันข้ามกับคุณสมบัติของช่องสัญญาณดังกล่าว นั่นก็คือเครื่องกำจัดผีในสัญญาณโทรทัศน์ที่ได้พัฒนาขึ้น

2.3 ลักษณะการใช้งานการประมวลผลสัญญาณดิจิตอล

การประมวลผลสัญญาณดิจิตอล สามารถนำมาประยุกต์ใช้งานได้ในหลายลักษณะ [12] แต่จะขอยกตัวอย่างการนำมาประยุกต์ใช้งานใน 2 ลักษณะซึ่งเกี่ยวข้องกับการกำจัดผี ดังนี้

2.3.1 การระบุเอกลักษณ์ (Identification)

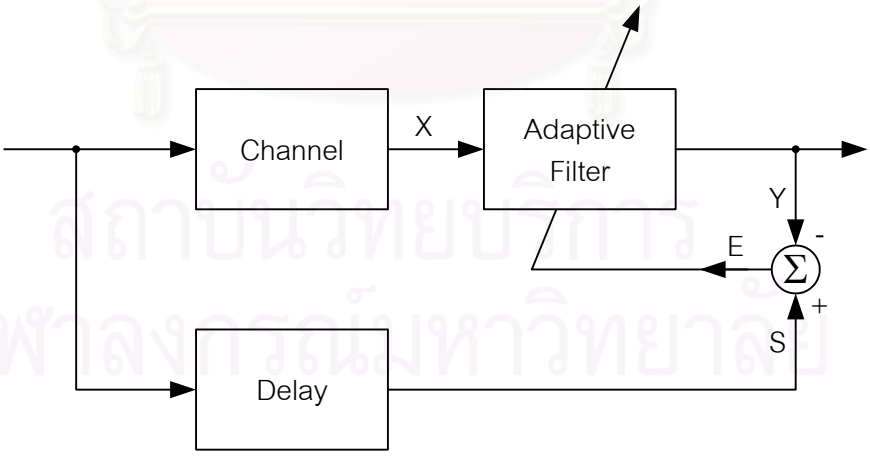
การระบุเอกลักษณ์เป็นการนำวงจรกรองแบบปรับตัว (Adaptive Filter) มาปรับให้มีคุณสมบัติเหมือนช่องสัญญาณ (Channel) ที่เราสนใจดังรูปที่ 2.3 โดยสัญญาณขาเข้าของวงจรกรอง S เป็นสัญญาณตั้งต้นที่ส่งออกไปผ่านช่องสัญญาณ จากนั้นหาค่าผิดพลาด E ระหว่างผลตอบของช่องสัญญาณ X กับผลตอบของวงจรกรองแบบปรับตัว Y แล้วนำไปปรับค่าสัมประสิทธิ์ของวงจรกรองให้มีค่าผิดพลาดลดลง เมื่อค่าผิดพลาดที่ได้ใกล้ศูนย์มากๆ จะได้ว่าวงจรกรองมีคุณสมบัติเช่นเดียวกับช่องสัญญาณที่เราสนใจ หรือสามารถพูดได้ว่า วงจรกรองจะมีฟังก์ชันถ่ายโอน (Transfer function) เหมือนกับช่องสัญญาณที่เราสนใจ



รูปที่ 2.3 โครงสร้างการระบุเอกลักษณ์ วงจรกรองจะถูกปรับให้มีคุณสมบัติเหมือนช่องสัญญาณ

2.3.2 การสร้างแบบจำลองผกผัน (Inverse modeling channel equalization)

การสร้างแบบจำลองผกผันเป็นการนำวงจรกรองแบบปรับตัว มาปรับให้มีคุณสมบัติผกผัน (Inverse) กับช่องสัญญาณที่เราสนใจดังรูปที่ 2.4 โดยสัญญาณขาเข้าของวงจรกรอง X เป็นสัญญาณขาออกของช่องสัญญาณ แล้วนำผลตอบที่ได้จากวงจรกรอง Y มาเปรียบเทียบกับสัญญาณตั้งต้น S นำค่าผิดพลาด E มาปรับสัมประสิทธิ์ของวงจรกรองให้มีค่าผิดพลาดลดลง และเมื่อค่าผิดพลาดใกล้ศูนย์มากๆ จะได้วงจรกรองที่มีคุณสมบัติผกผันกับช่องสัญญาณที่เราสนใจ หรืออีกนัยหนึ่ง ฟังก์ชันถ่ายโอนของวงจรกรองจะเป็นฟังก์ชันผกผันของฟังก์ชันถ่ายโอนของช่องสัญญาณ



รูปที่ 2.4 โครงสร้างการสร้างแบบจำลองผกผัน วงจรกรองจะถูกปรับให้มีคุณสมบัติผกผันกับช่องสัญญาณ

ในวิธีแรกที่ใช้การระบุเอกลักษณ์ เมื่อหาคุณสมบัติของช่องสัญญาณที่ทำให้เกิดผี หลังจากที่รู้ฟังก์ชันถ่ายโอนของช่องสัญญาณได้แล้ว ก็จะสามารถนำมาคำนวณหาค่าสัมประสิทธิ์ของวงจร

กรองที่เป็นฟังก์ชันผกผันกับช่องสัญญาณได้ [14] เมื่อผ่านสัญญาณที่มีผีไปในวงจรกรองนั้นแล้ว สัญญาณผีจะถูกกำจัดไป เนื่องจากฟังก์ชันถ่ายโอนโดยรวมของช่องสัญญาณกับวงจรกรองที่ออกแบบมีค่าเท่ากับหนึ่ง

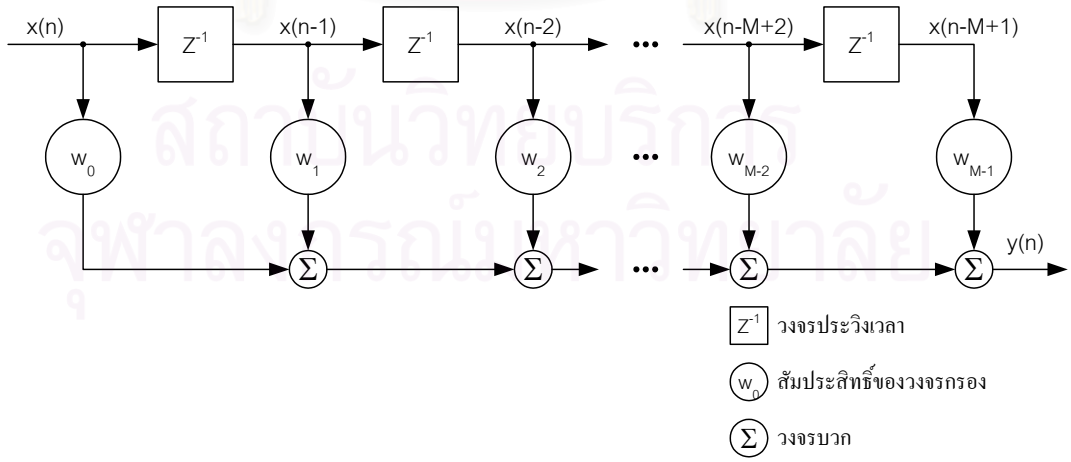
ส่วนในวิธีที่สองที่ใช้การสร้างแบบจำลองผกผันนั้น เมื่อปรับค่าสัมประสิทธิ์ของวงจรกรองแบบปรับตัว จนได้ผลต่างของสัญญาณขาออกกับสัญญาณตั้งต้นมีค่าใกล้ศูนย์มากๆ แล้ว ผลลัพธ์ของวงจรกรองจะเป็นสัญญาณที่ถูกกำจัดผีแล้วด้วย ซึ่งจะเห็นว่าวิธีนี้ มีขั้นตอนการคำนวณที่ง่าย และซับซ้อนน้อยกว่าวิธีแรก แต่อย่างไรก็ดีวิธีที่สองจะทนสัญญาณรบกวนได้น้อยกว่าวิธีแรก [12]

2.4 วงจรกรองแบบปรับตัว

องค์ประกอบของวงจรกรองแบบปรับตัวมีอยู่สองอย่าง อย่างแรกคือวงจรกรอง ซึ่งมีสัมประสิทธิ์ที่สามารถปรับค่าได้ และอย่างที่สองคือกระบวนวิธี (Algorithm) ทำหน้าที่รับค่าผลต่างของสัญญาณที่วงจรกรองสร้างขึ้น กับสัญญาณอ้างอิง มาประมวลผลเพื่อปรับค่าสัมประสิทธิ์ของวงจรกรอง ให้มีความผิดพลาดลดลง สามารถแบ่งได้เป็นสองประเภทใหญ่ คือ

2.4.1 วงจรกรองเอฟไอาร์ (FIR Filter)

วงจรกรองเอฟไอาร์เป็นวงจรกรองที่ผลลัพธ์ของวงจรขึ้นกับข้อมูลขาเข้าในอดีตช่วงเวลาหนึ่งเท่านั้น ข้อมูลในอดีตที่ยาวนานกว่านั้น จะไม่มีผลใดๆ ต่อผลลัพธ์เลย วงจรกรองประเภทนี้จะมีโครงสร้างแบบป้อนไปข้างหน้าอย่างเดียวเท่านั้น

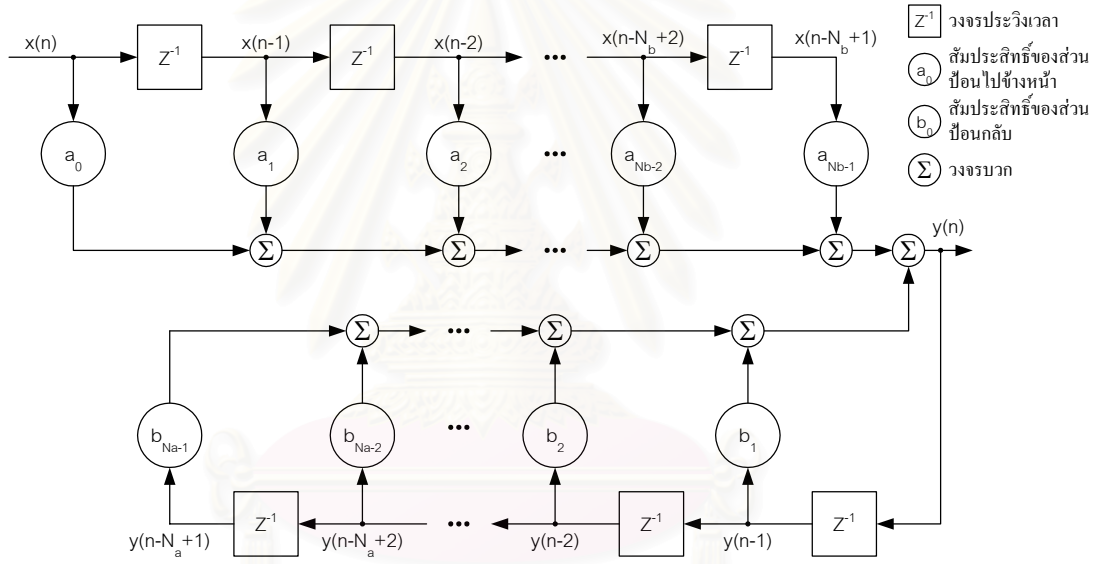


รูปที่ 2.5 โครงสร้างของวงจรกรองเอฟไอาร์เป็นแบบป้อนไปข้างหน้าเท่านั้น

จากโครงสร้างของวงจรกรองเอฟไอออาร์ดังรูปที่ 2.5 จะเห็นว่าผลลัพธ์ของวงจรกรอง y(n) จะเกิดจากการรวมกันของผลคูณระหว่างข้อมูลขาเข้าในอดีต x(n), x(n-1), x(n-2), …, x(n-M+1) กับค่าสัมประสิทธิ์ a_0, a_1, a_2, …, a_{M-1} นั่นคือข้อมูลในอดีต M ตัวเท่านั้น ที่มีผลกับผลลัพธ์ของวงจรกรอง

2.4.2 วงจรกรองไอไออาร์ (IIR Filter)

วงจรกรองไอไออาร์เป็นวงจรกรองที่ผลลัพธ์ของวงจรขึ้นกับข้อมูลในอดีตเป็นช่วงเวลานานมากๆ ซึ่งเกิดจากการป้อนกลับข้อมูลขาออกในอดีตมารวมให้เกิดผลลัพธ์เป็นข้อมูลขาออกในเวลาปัจจุบัน ดังนั้นวงจรกรองประเภทนี้จะมีโครงสร้างทั้งแบบป้อนไปข้างหน้า และแบบป้อนกลับ ดังรูปที่ 2.6



รูปที่ 2.6 โครงสร้างของวงจรกรองไอไออาร์มีทั้งส่วนป้อนไปข้างหน้าและป้อนกลับ

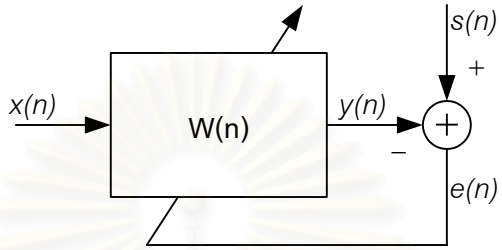
ในการสร้างเครื่องกำจัดผีในสัญญาณโทรทัศน์ จะต้องใช้วงจรกรองที่มีความยาวสูงมาก และถึงแม้ว่าวงจรกรองเอฟไอออาร์ จะต้องใช้ความยาวของวงจรกรองที่มากกว่าวงจรกรองไอไออาร์เพื่อให้ได้ผลตอบสนองทางอิมพัลส์ที่ใกล้เคียงกัน [15] แต่วงจรกรองไอไออาร์มีเสถียรภาพ และความทนทานต่อสัญญาณรบกวนน้อยกว่าวงจรกรองเอฟไอออาร์มาก

2.5 **กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด** (LMS : Least Mean Square Algorithm)

กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด เป็นกระบวนวิธีที่ง่าย มีการคำนวณไม่ซับซ้อน และมีเวลาการลู่เข้า (Convergence Time) ที่เร็ว [2] แบ่งออกได้เป็น 2 แบบตามลักษณะของวงจรกรองที่ใช้ [16] คือ

2.5.1 <u>กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองเอฟไอาร์</u>

กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองเอฟไอาร์มีลักษณะแสดงได้ดังรูปที่ 2.7 สัญญาณขาเข้า $x(n)$ ถูกป้อนผ่านวงจรกรองแบบปรับตัวที่มีค่าสัมประสิทธิ์ของวงจรกรองเป็น $\mathbf{W(n)} = [w_0(n), w_1(n), ..., w_{N-1}(n)]^{\mathbf{T}}$



รูปที่ 2.7 กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองเอฟไอาร์

ผลลัพธ์ของวงจรกรอง $y(n)$ หาได้จากสมการที่ 2.1 และความผิดพลาด $e(n)$ คำนวณจากผลต่างระหว่างผลลัพธ์ กับสัญญาณอ้างอิง $s(n)$ ดังสมการที่ 2.2

$$y(n) = \mathbf{W^T(n) * X(n)} \text{ หรือ } y(n) = \sum_{i=0}^{N-1} w_i(n) \cdot x(n-i) \quad \text{.....สมการที่ 2.1}$$

$$e(n) = s(n) - y(n) \text{ หรือ } e(n) = s(n) - \sum_{i=0}^{N-1} w_i(n) \cdot x(n-i) \quad \text{.....สมการที่ 2.2}$$

โดยที่ $\mathbf{X(n)} = [x(n), x(n-1), ..., x(n-N+1)]^{\mathbf{T}}$ และ N เป็นจำนวนแท็ปของวงจรกรอง

หลักการของกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดคือ จะพยายามทำให้ค่า $J(n) = \mathrm{E}[e^2(n)]$ ซึ่งเป็นค่าคาดหวัง (Expectation Value) กำลังสองของค่าความผิดพลาด มีค่าน้อยที่สุด ซึ่งจะส่งผลให้ผลลัพธ์ของวงจรกรองมีค่าใกล้เคียงกับสัญญาณอ้างอิงมากที่สุดด้วย จากหลักแคลคูลัสการหาค่า $J(n)$ ที่น้อยที่สุดจากการปรับ $\mathbf{W(n)}$ ทำได้โดยหา $\nabla_{\mathbf{W}} J(n)$ ดังสมการที่ 2.3

$$\nabla_{\mathbf{W}} J(n) = \begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \\ \mathrm{M} \\ \frac{\partial J}{\partial w_{N-1}} \end{bmatrix} = 2\,\mathrm{E}\begin{bmatrix} e(n)\frac{\partial e(n)}{\partial w_0} \\ e(n)\frac{\partial e(n)}{\partial w_1} \\ \mathrm{M} \\ e(n)\frac{\partial e(n)}{\partial w_{N-1}} \end{bmatrix} \quad \text{.....สมการที่ 2.3}$$

จากสมการที่ 2.2 จะสามารถหาค่าของ $\frac{\partial e(n)}{\partial w_i}$ ได้ดังสมการ ที่ 2.4 ดังนั้นค่า $\nabla_{\mathbf{W}} J(n)$ เป็นดังสมการที่ 2.5

$$\frac{\partial e(n)}{\partial w_i} = -x(n-i) \quad \text{.....สมการที่ 2.4}$$

$$\nabla_{\mathbf{W}} J(n) = -2\,\mathrm{E}\left[e(n)\cdot \mathbf{X(n)}\right] \quad \text{.....สมการที่ 2.5}$$

หลักการค้นหาแบบสตีเพสเดสเซนต์ (Steepest Descent Search) ดังสมการที่ 2.6 ได้ถูกนำมาใช้ในการปรับค่าสัมประสิทธิ์ของวงจรกรอง

$$\mathbf{W(n+1)} = \mathbf{W(n)} - \frac{1}{2}\mu \cdot \nabla_{\mathbf{W}} J(n) \quad \text{.....สมการที่ 2.6}$$

เมื่อ μ คือแฟกเตอร์การลู่เข้า (Convergence factor) และหลังจากแทนค่าของ $\nabla_{\mathbf{W}} J(n)$ จากสมการที่ 2.5 ลงในสมการที่ 2.6 จะเป็นดังสมการที่ 2.7

$$\mathbf{W(n+1)} = \mathbf{W(n)} + \mu \cdot \mathrm{E}\left[e(n)\cdot \mathbf{X(n)}\right] \quad \text{.....สมการที่ 2.7}$$

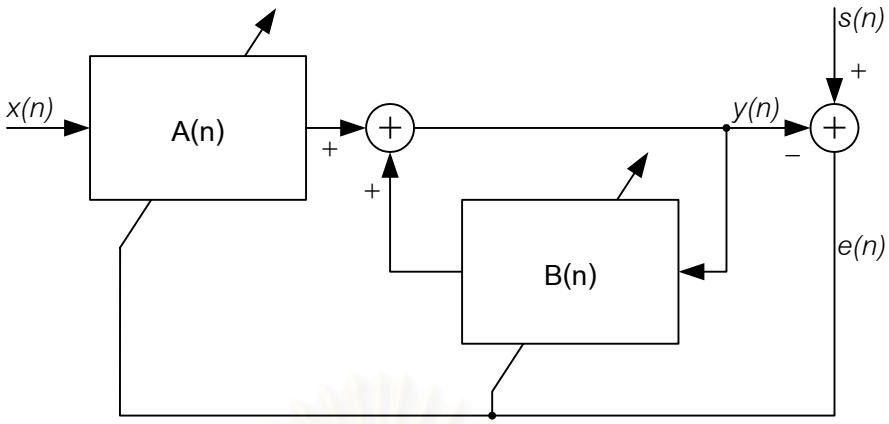
แต่การหาค่าคาดหวังจะใช้การคำนวณมาก ซึ่งในระบบกำจัดผีต้องทำงานแบบเวลาจริง (Real Time) จึงใช้หลักการค้นหาแบบสโตแคสติค (Stochastic Search) จะสามารถประมาณค่าของ $\mathrm{E}[e(n)x(n-i)]$ ให้เป็น $e(n)x(n-i)$ ดังนั้นการประมาณค่าผิดพลาด และการประมาณค่าสัมประสิทธิ์เป็นดังสมการที่ 2.8 และสมการที่ 2.9 ตามลำดับ

$$e(n) = s(n) - \mathbf{W^T(n)} * \mathbf{X(n)} \quad \text{.....สมการที่ 2.8}$$

$$\mathbf{W(n+1)} = \mathbf{W(n)} + \mu \cdot e(n) \cdot \mathbf{X(n)} \quad \text{.....สมการที่ 2.9}$$

2.5.2 <u>กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองไอไออาร์</u>

เนื่องจากวงจรกรองแบบไอไออาร์มีโครงสร้างทั้งแบบป้อนไปข้างหน้า และแบบป้อนกลับ ทำให้ในกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยสุดต้องมีการคำนวณแยกเป็นสองส่วน [16] ดังรูปที่ 2.8 โดยส่วนป้อนไปข้างหน้ามีค่าสัมประสิทธิ์เป็น $\mathbf{A(n)} = [a_0(n), a_1(n), ..., a_{N_a-1}(n)]^{\mathbf{T}}$ และส่วนป้อนกลับมีค่าสัมประสิทธิ์เป็น $\mathbf{B(n)} = [b_1(n), b_2(n), ..., b_{N_b-1}(n)]^{\mathbf{T}}$



รูปที่ 2.8 กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองไอไออาร์

ค่าผลลัพธ์ของวงจรกรอง $y(n)$ สามารถหาได้ดังสมการที่ 2.10 และค่าความผิดพลาด $e(n)$ สามารถหาได้จากผลต่างระหว่างค่าผลลัพธ์ กับค่าสัญญาณอ้างอิง $s(n)$ ดังสมการที่ 2.11

$$y(n) = \mathbf{A^T(n) * X(n) + B^T(n) * Y(n-1)}$$

หรือ $y(n) = \sum_{i=0}^{N_a-1} a_i \cdot x(n-i) + \sum_{j=1}^{N_b-1} b_j \cdot y(n-j)$ .....สมการที่ 2.10

$$e(n) = s(n) - y(n)$$ .....สมการที่ 2.11

โดยที่ $\mathbf{Y(n-1)} = \left[y(n-1), y(n-2), \mathrm{K}\ , y(n-N_b+1)\right]^{\mathbf{T}}$

หลักการของกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดคือ จะพยายามทำให้ค่า $J(n) = \mathrm{E}[e^2(n)]$ ซึ่งเป็นค่าเฉลี่ยกำลังสองของค่าความผิดพลาดมีค่าน้อยที่สุด ทำได้โดยการหาค่าของ $\nabla_{\mathbf{A}} J(n)$ และ $\nabla_{\mathbf{B}} J(n)$ ดังนี้

$$\nabla_{\mathbf{A}} J(n) = \begin{bmatrix} \frac{\partial J}{\partial a_0} \\ \frac{\partial J}{\partial a_1} \\ \mathrm{M} \\ \frac{\partial J}{\partial a_{N_a}} \end{bmatrix} = 2\,\mathrm{E}\begin{bmatrix} e(n)\frac{\partial e(n)}{\partial a_0} \\ e(n)\frac{\partial e(n)}{\partial a_1} \\ \mathrm{M} \\ e(n)\frac{\partial e(n)}{\partial a_{N_b}} \end{bmatrix}$$ .....สมการที่ 2.12

ดังนั้นในการหาค่า $\nabla_{\mathbf{A}} J(n)$ จำเป็นต้องหาค่า $\frac{\partial e(n)}{\partial a_i}$ หรือ $\frac{\partial y(n)}{\partial a_i}$ ก่อน ซึ่งจากสมการที่ 2.10 และสมการที่ 2.11 จะสามารถหาค่าของ $\frac{\partial y(n)}{\partial a_i}$ ได้ดังสมการ ที่ 2.13

$$\frac{\partial y(n)}{\partial a_i} = x(n-i) + \sum_{j=1}^{N_b-1} b_j \cdot \frac{\partial y(n-j)}{\partial a_i} \quad \text{.....สมการที่ 2.13}$$

ในทำนองเดียวกัน ในการหาค่า $\nabla_{\mathbf{B}} J(n)$ จำเป็นต้องหาค่า $\frac{\partial y(n)}{\partial b_j}$ ก่อน ซึ่งจากสมการที่ 2.10 และสมการที่ 2.11 จะสามารถหาค่าของ $\frac{\partial y(n)}{\partial b_j}$ ได้ดังสมการ ที่ 2.14

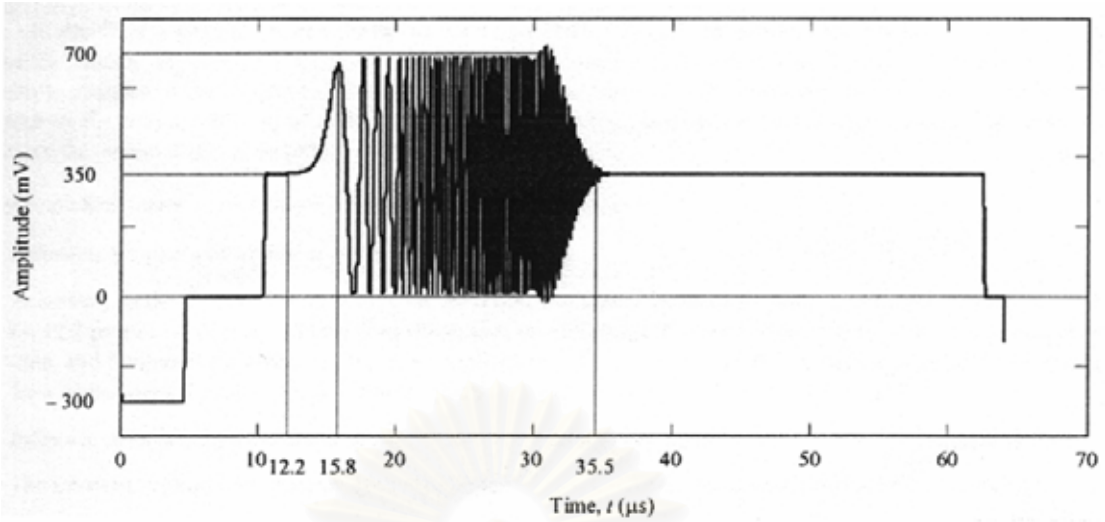
$$\frac{\partial y(n)}{\partial b_j} = y(n-j) + \sum_{k-1}^{N_b-1} b_k \cdot \frac{\partial y(n-k)}{\partial b_j} \quad \text{.....สมการที่ 2.14}$$

จะเห็นว่าในการหาค่าของสมาชิกภายใน $\nabla_{\mathbf{A}} J(n)$ หนึ่งตัว จะมีการคำนวณโดยใช้คำสั่งประเภทคูณบวก (Multiply-Accumulate) N_b ครั้งดังสมการที่ 2.13 และสมาชิกของ $\nabla_{\mathbf{A}} J(n)$ มี N_a ตัว ทำให้ต้องคำนวณคำสั่งประเภทคูณบวกทั้งสิ้น $N_a N_b$ ครั้ง ในทำนองเดียวกัน $\nabla_{\mathbf{B}} J(n)$ ต้องคำนวณคำสั่งประเภทคูณบวก $N_a N_b$ ครั้งเช่นกัน ดังนั้นจะเห็นได้ว่า กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดสำหรับวงจรกรองแบบไอไออาร์ ต้องใช้คำสั่งในการคำนวณมากกว่าแบบเอฟ-ไออาร์มาก

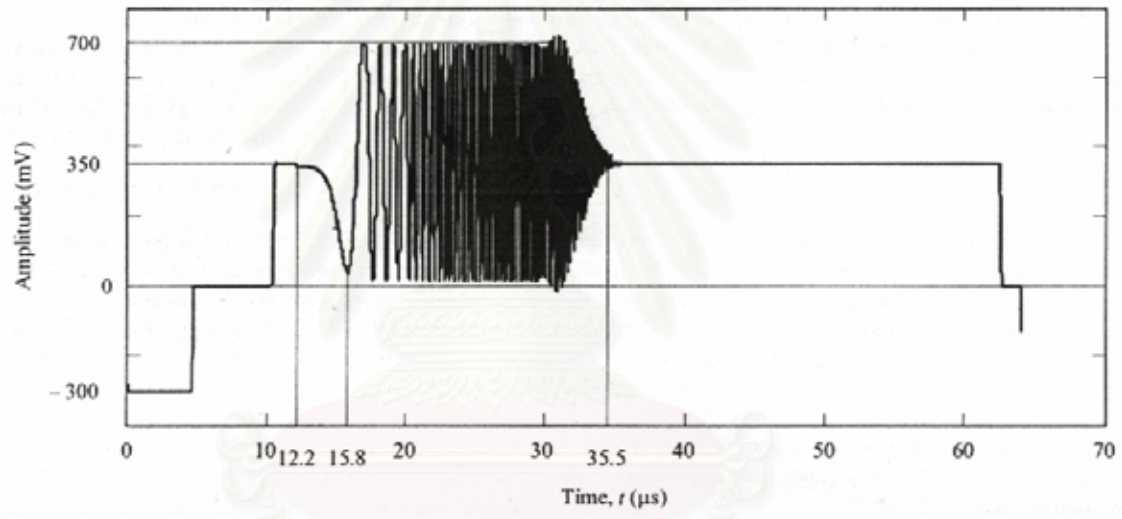
2.6 **สัญญาณอ้างอิงสำหรับกำจัดผี** (GCR : Ghost Cancelling Reference Signal)

สำหรับระบบกำจัดผีในสัญญาณโทรทัศน์ สัญญาณอ้างอิงสำหรับกำจัดผีจะถูกแทรกเข้าไปในช่วงไร้ภาพทางแนวตั้ง (Vertical Blanking Interval) ของสัญญาณโทรทัศน์ [17] ที่สถานีส่งและถูกส่งแพร่ภาพออกมาพร้อมสัญญาณโทรทัศน์ เพื่อให้เครื่องกำจัดผีที่ภาครับ ใช้สัญญาณอ้างอิงในการกำจัดผีก่อนปล่อยให้สัญญาณที่ปราศจากผีผ่านไปสู่เครื่องรับโทรทัศน์

ในโครงงานนี้ เราใช้สัญญาณอ้างอิงตามมาตรฐาน ITU-R BT.1124 แบบ C [18] แสดงดังรูปที่ 2.9 ซึ่งแบ่งออกเป็น (ก) สัญญาณอ้างอิงแบบบวก และ (ข) สัญญาณอ้างอิงแบบลบ โดยใช้เครื่องสร้างและแทรกรูปแบบสัญญาณโทรทัศน์รุ่น PM 5655 (PM 5655 VITS Generator & Inserter) ในการแทรกสัญญาณอ้างอิงแบบบวกสลับกันกับแบบลบเข้าไปในสัญญาณวีดิทัศน์เส้นที่ 318 ซึ่งสัญญาณอ้างอิงมีลักษณะเป็นสัญญาณรูปไซน์ (Sine Signal) ที่เปลี่ยนแปลงความถี่ตามเวลาจากความถี่ต่ำไปความถี่สูง มีขนาดของผลตอบความถี่เท่ากันตลอดความถี่ของสัญญาณโทรทัศน์ และมีผลตอบเฟสที่ต่อเนื่อง (Smooth Phase) ข้อดีที่สำคัญของสัญญาณในลักษณะนี้คือ การเปลี่ยนแปลงอัตราสุ่มข้อมูล หรือจำนวนบิตของข้อมูลที่ยังอยู่ในช่วงที่เหมาะสมจะไม่มีผลกระทบต่อคุณสมบัติของสัญญาณอ้างอิง [19] อีกทั้งยังช่วยให้เวลาลู่เข้ามีค่าน้อย [1]



(ก) แบบบวก สัญญาณเริ่มแกว่งในทางเพิ่มขึ้นก่อน



(ข) แบบลบ สัญญาณเริ่มแกว่งในทางลดลงก่อน

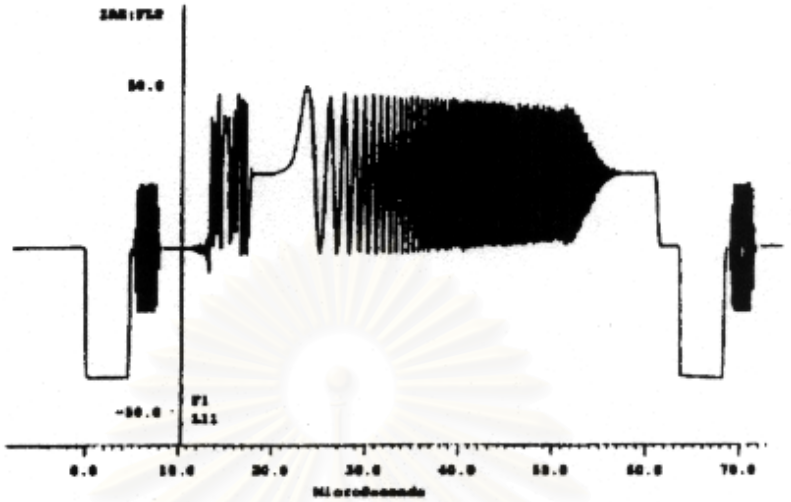
รูปที่ 2.9 ลักษณะของสัญญาณอ้างอิงที่ใช้กำจัดผี (ก) แบบบวก (ข) แบบลบ
คล้ายกับสัญญาณไซน์ ที่มีความถี่เพิ่มขึ้นตามเวลา

2.7 งานวิจัยที่เคยมีมาก่อน

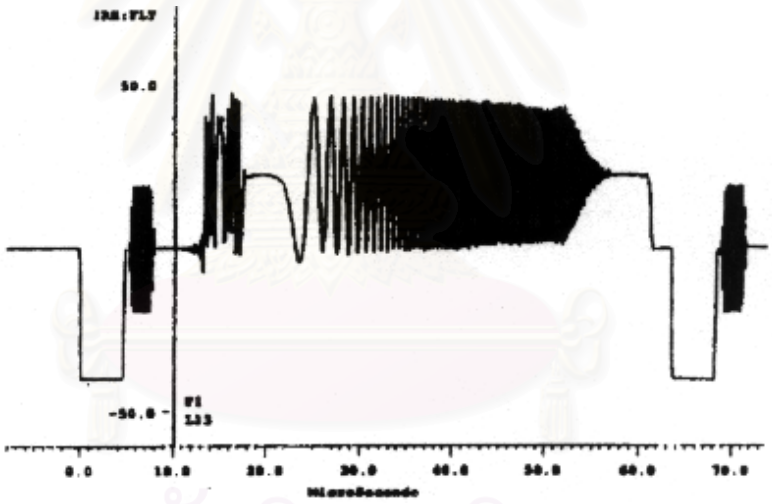
ในช่วง 20 ปีที่ผ่านมา มีงานวิจัยที่เกี่ยวกับระบบกำจัดผีในสัญญาณโทรทัศน์ถูกนำเสนอออกมามากมาย ซึ่งจะขอกล่าวถึงผลงานวิจัยที่น่าสนใจ ดังนี้

ในปี ค.ศ.1993 งานวิจัยเรื่อง “A Novel TV Ghost Cancellation System” [1] นำเสนอระบบกำจัดผีในสัญญาณโทรทัศน์ โดยสัญญาณอ้างอิงสำหรับกำจัดผีที่ใช้ เกิดจากการรวมกันของสัญญาณอ้างอิงสองลักษณะคือ สัญญาณอ้างอิงแบบลำดับพีเอ็น (PN Sequence) กับสัญญาณ

อ้างอิงแบบรูปคลื่นเชิพ (Chirp waveform) ดังรูปที่ 2.10 ซึ่งแบ่งออกเป็น (ก) แบบบวก และ (ข) แบบลบ



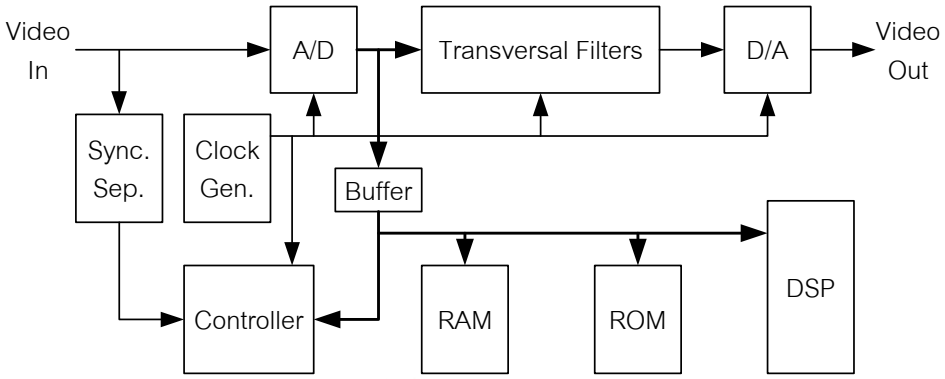
(ก) แบบบวก สัญญาณเริ่มแกว่งในทางเพิ่มขึ้นก่อน



(ข) แบบลบ สัญญาณเริ่มแกว่งในทางลดลงก่อน

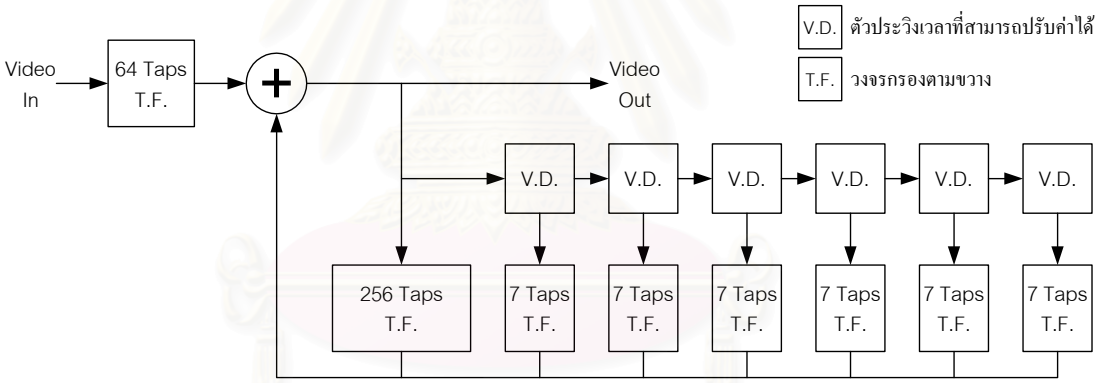
รูปที่ 2.10 สัญญาณอ้างอิงแบบพีเอ็น และรูปคลื่นเชิพ (ก) แบบบวก (ข) แบบลบ

งานวิจัยนี้นำเสนอเครื่องกำจัดผีที่มีโครงสร้างเป็นดังรูปที่ 2.11 ซึ่งมีตัวควบคุม (Controller) ออกแบบเป็น ASIC ทำหน้าที่ควบคุมการติดต่อระหว่างอุปกรณ์ต่างๆ และจังหวะการโอนถ่ายข้อมูลแบบเข้าถึงหน่วงความจำโดยตรง (DMA : Direct Memory Access) ของการถ่ายโอนข้อมูลจากวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลไปยังหน่วยความจำ และการถ่ายโอนข้อมูลจากหน่วยความจำไปยังวงจรกรองตามขวาง (Transversal filter)



รูปที่ 2.11 โครงสร้างของเครื่องกำจัดผีที่มีมาก่อน [1]

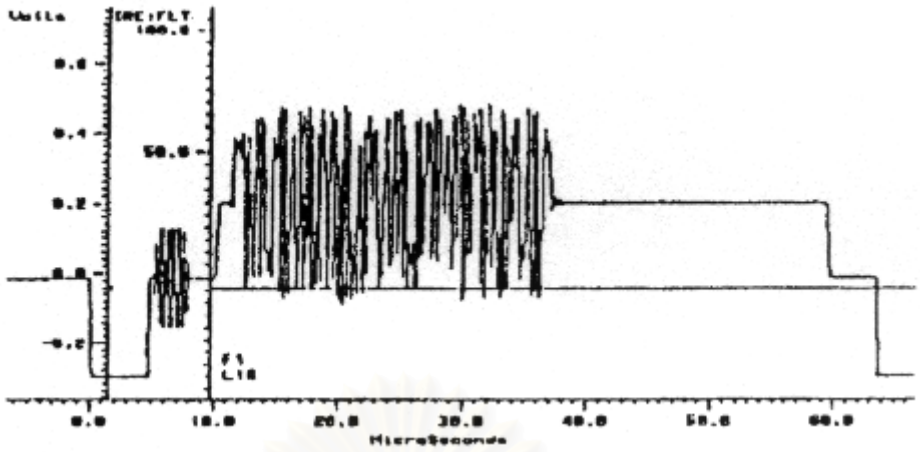
โครงสร้างของวงจรกรองตามขวางประกอบไปด้วย วงจรกรองเอฟไอออาร์ขนาด 64 แท็ปใช้สำหรับกำจัดผีที่มีความหน่วงอยู่ในช่วง -2.4 μs ถึง 2.0 μs และวงจรกรองไอไอออาร์สองชุด ชุดแรกเป็นวงจรกรองไอไอออาร์ขนาด 256 แท็ปใช้ในการกำจัดผีที่มีความหน่วงไม่เกิน 15.0 μs ชุดที่สองถูกสร้างโดยวงจรกรองเอฟไอออาร์ 6 กลุ่ม ซึ่งแต่ละกลุ่มประกอบด้วยวงจรกรองเอฟไอออาร์ขนาด 7 แท็ป และตัวหน่วงเวลาแบบปรับค่าได้ (Variable Delay) ดังรูปที่ 2.12



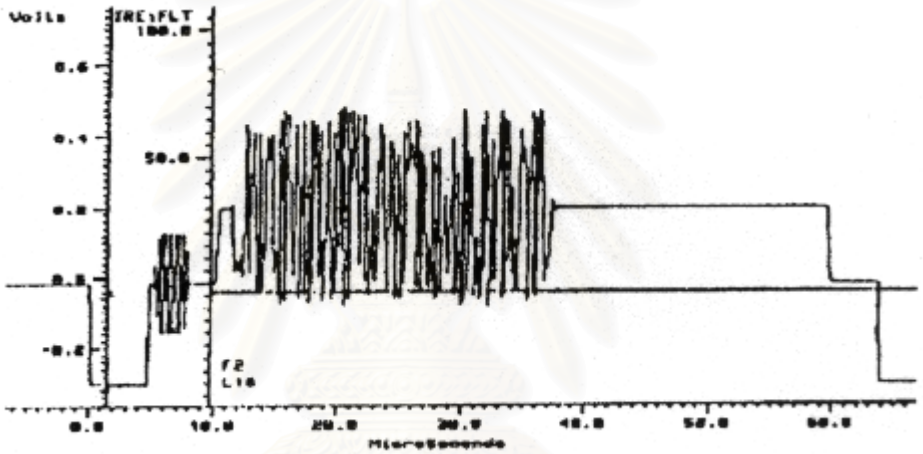
รูปที่ 2.12 สถาปัตยกรรมของวงจรกรองตามขวาง

ตัวประมวลผลสัญญาณเชิงเลข (DSP) ทำหน้าที่คำนวณหาค่าสัมประสิทธิ์ของวงจรกรองที่เหมาะสม โดยจะคำนวณหาความสัมพันธ์ในอณาจักรความถี่ (Frequency domain) ซึ่งประสิทธิภาพในการคำนวณหาค่าสัมประสิทธิ์ที่เหมาะสมทำได้ภายในเวลา 0.6 วินาที

ในปี ค.ศ.1994 งานวิจัยเรื่อง “A New Ghost Cancellation System” [2] นำเสนอระบบกำจัดผีในสัญญาณโทรทัศน์ โดยใช้สัญญาณอ้างอิงสำหรับกำจัดผีของประเทศเกาหลี (KGCR : Korean GCR) ซึ่งมีลักษณะเป็นสัญญาณอ้างอิงแบบลำดับสามองค์ประกอบ (Ternary Sequence) ประกอบด้วย 0, +1 และ -1 ดังรูปที่ 2.13 เป็นลักษณะของสัญญาณอ้างอิงของประเทศเกาหลี แบ่งออกเป็น (ก) แบบบวก และ (ข) แบบลบ



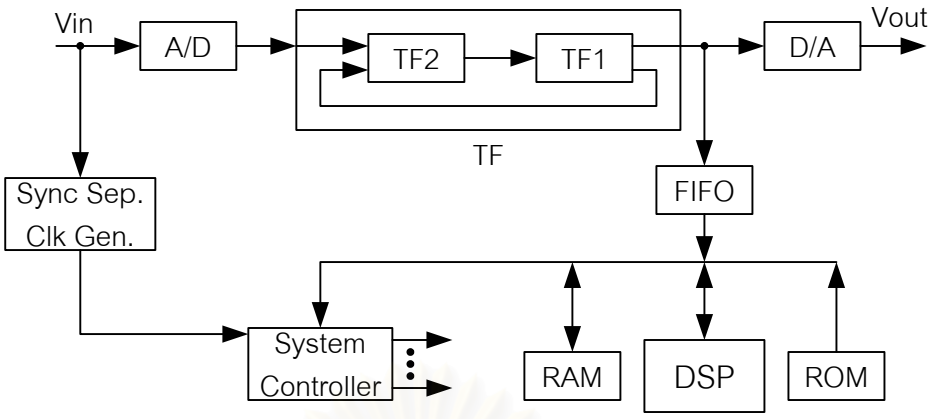
(ก) แบบบวก สัญญาณเริ่มแกว่งในทางเพิ่มขึ้นก่อน



(ข) แบบลบ สัญญาณเริ่มแกว่งในทางลดลงก่อน

รูปที่ 2.13 สัญญาณอ้างอิงของประเทศเกาหลี (ก) แบบบวก (ข) แบบลบ มีลักษณะแบบลำดับสามองค์ประกอบ

โครงสร้างของเครื่องกำจัดผีที่ถูกนำเสนอเป็นดังรูปที่ 2.14 โดยวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลจะทำการแปลงสัญญาณวีดิทัศน์ขาเข้า (Vin) ให้เป็นข้อมูลดิจิตอลขนาด 8 บิตที่อัตราสุ่มข้อมูล 14.31 MHz ข้อมูลดิจิตอลนี้จะถูกส่งผ่านวงจรกรองซึ่งแบ่งออกเป็นสองส่วนคือ TF1 เป็นวงจรกรองแบบเอฟไอาร์ขนาด 72 แท็ป ทำหน้าที่กำจัดผีที่มีความหน่วงในช่วง -2.5 μs ถึง 2.5 μs และ TF2 เป็นวงจรกรองแบบไอไออาร์ขนาด 576 แท็ป ทำหน้าที่กำจัดผีที่มีความหน่วงในช่วง 0 ถึง 40 μs ซึ่งค่าสัมประสิทธิ์ของวงจรกรองทั้งสองจะถูกคำนวณโดยตัวประมวลผลสัญญาณเชิงเลข ด้วยกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด และข้อมูลดิจิตอลที่ผ่านกระบวนการกำจัดผีแล้ว จะถูกส่งให้กับตัวแปลงสัญญาณดิจิตอลเป็นแอนาลอกเพื่อแปลงให้กลับเป็นสัญญาณวีดิทัศน์ที่ปราศจากผี



รูปที่ 2.14 โครงสร้างของเครื่องกำจัดผีที่มีมาก่อน [2]

เครื่องกำจัดผีในงานวิจัยนี้ สามารถกำจัดผีที่มีความหน่วงในช่วง -2.5 ถึง 42.5 μs ได้ เมื่อสัญญาณผีมีความเข้มไม่เกิน -3 dB เทียบกับสัญญาณหลัก โดยจะใช้เวลาในการกำจัดผีประมาณ 3 วินาที และหลังจากการกำจัดผีเสร็จสิ้น ความเข้มของสัญญาณผีที่เหลืออยู่มีค่าไม่เกิน -40 dB เทียบกับสัญญาณหลัก

นอกจากผลงานวิจัยที่น่าสนใจดังกล่าว ยังมีงานวิจัยที่นำเสนอเกี่ยวกับส่วนประกอบต่างๆ ภายในระบบกำจัดผีไว้มากมายเช่น งานวิจัยเกี่ยวกับสัญญาณอ้างอิงสำหรับกำจัดผี [3, 4, 5] จะเสนอสัญญาณอ้างอิงที่สามารถหาคุณสมบัติของช่องสัญญาณได้ดีขึ้น, งานวิจัยวงจรกรองที่ใช้ในการกำจัดผี [6, 7, 8] โดยส่วนใหญ่นำเสนอวงจรกรองเอฟไออาร์ใช้กำจัดผีที่มีความหน่วงน้อยร่วมกับวงจรกรองไอไออาร์ใช้กำจัดผีที่มีความหน่วงมากขึ้น และงานวิจัยเกี่ยวกับโครงสร้างของเครื่องกำจัดผี [9, 10] พบว่างานวิจัยส่วนใหญ่นำเสนอโครงสร้างของเครื่องกำจัดผีที่คล้ายคลึงกัน ประกอบไปด้วย วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล, วงจรแยกสัญญาณซิงก์, วงจรกรอง, ตัวประมวลผลสัญญาณดิจิตอล, หน่วยความจำแบบเข้าก่อนออกก่อน, และวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล เป็นต้น

2.8 สรุปท้ายบท

ในบทนี้ได้กล่าวถึงความรู้พื้นฐานต่างๆ ที่จำเป็นต่อการพัฒนาเครื่องกำจัดผีในสัญญาณโทรทัศน์ เริ่มจากการเกิดผีในธรรมชาติ และได้กล่าวถึงระบบกำจัดผีในสัญญาณโทรทัศน์ ซึ่งมีส่วนประกอบที่สำคัญคือเครื่องกำจัดผีในสัญญาณโทรทัศน์ ที่ต้องอาศัยการประมวลผลสัญญาณดิจิตอลเข้ามาช่วย โดยมีส่วนสำคัญคือวงจรกรองแบบปรับตัว แบ่งออกได้เป็นสองแบบคือ วงจรกรองแบบเอฟไออาร์ และวงจรกรองแบบไอไออาร์ จากนั้นได้กล่าวถึงกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด ซึ่งเป็นกระบวนวิธีที่เหมาะสม สำหรับนำมาใช้ในการกำจัดผี อีกทั้งยังได้กล่าวถึง

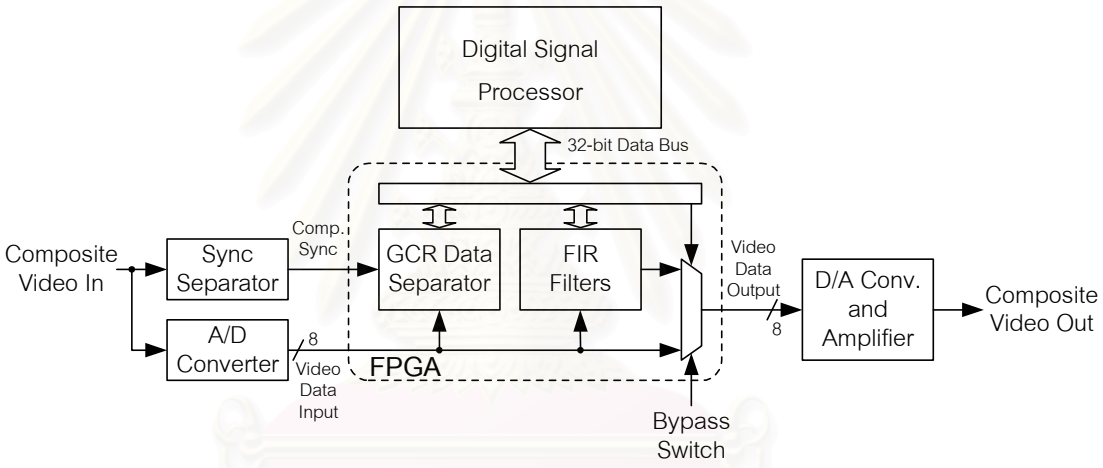
ลักษณะของสัญญาณอ้างอิงสำหรับกำจัดผีที่จะนำมาใช้ในงานวิจัยนี้ และสุดท้ายได้กล่าวถึงงานวิจัยที่เกี่ยวข้องกับการกำจัดผีในสัญญาณโทรทัศน์

บทที่ 3

เครื่องกำจัดผีในสัญญาณโทรทัศน์

3.1 โครงสร้างภายในเครื่องกำจัดผีในสัญญาณโทรทัศน์

เครื่องกำจัดผีในสัญญาณโทรทัศน์ประกอบด้วย วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล , วงจรแยกซิงก์, วงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก, วงจรขยาย, ตัวประมวลผลสัญญาณดิจิตอล และวงจรที่อยู่ภายในเอฟพีจีเอ ซึ่งมีสองวงจรหลักคือวงจรกรองเอฟไอาร์ และวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี ดังรูปที่ 3.1 โดยมีหลักการทำงานดังนี้

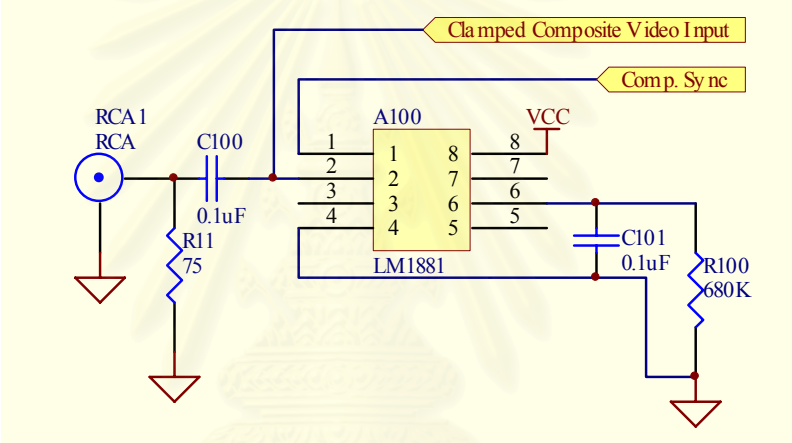


รูปที่ 3.1 ส่วนประกอบหลักภายในเครื่องกำจัดผีในสัญญาณโทรทัศน์

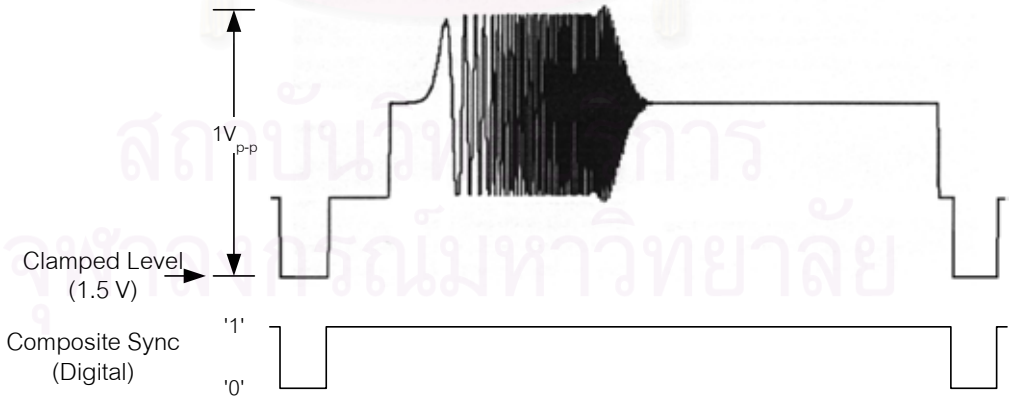
เมื่อสัญญาณภาพรวม (Composite Video) ที่มีสัญญาณผีรวมอยู่ด้วยถูกป้อนเข้าสู่เครื่องกำจัดผี สัญญาณจะผ่านวงจรแปลงสัญญาณแอนาลอกให้เป็นสัญญาณดิจิตอล ในขณะเดียวกันก็จะถูกแยกซิงก์ออกมาด้วยวงจรแยกซิงก์ (Sync Separator) แล้วป้อนให้กับวงจรที่อยู่ภายในชิพเอฟพีจีเอ โดยวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี จะทำการแยกข้อมูลสัญญาณอ้างอิงออกมาเตรียมไว้ให้กับตัวประมวลผลสัญญาณดิจิตอล ซึ่งจะนำไปคำนวณหาค่าสัมประสิทธิ์ที่เหมาะสมให้กับวงจรกรองเอฟไอาร์เพื่อนำไปใช้ในการกำจัดผี จากนั้นสัญญาณภาพแบบดิจิตอลที่ถูกกำจัดผีแล้ว จะถูกป้อนให้กับวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอกให้กลับมาเป็นสัญญาณภาพรวมที่ใช้ป้อนให้กับเครื่องรับโทรทัศน์ ผู้ใช้สามารถเลือกรับชมภาพสัญญาณโทรทัศน์ที่ไม่ผ่านกระบวนการกำจัดผีได้ โดยการควบคุมผ่านทางสวิตซ์ปล่อยผ่าน (Bypass Switch) รายละเอียดของวงจรแต่ละส่วนจะอธิบายในหัวข้อต่อๆ ไป

3.2 วงจรแยกสัญญาณซิงก์ (Sync Separator)

สัญญาณภาพรวมที่ป้อนเข้าสู่เครื่องกำจัดผีทางตัวเชื่อมต่อแบบอาร์ซีเอ (RCA Connector) ถูกป้อนผ่านตัวเก็บประจุไปยังวงจรรวม LM1881 [20] ซึ่งภายในวงจรรวม LM1881 มีวงจรที่ทำหน้าที่แยกสัญญาณซิงก์ให้ออกมาเป็นสัญญาณดิจิตอล เพื่อป้อนให้กับวงจรแยกสัญญาณอ้างอิงสำหรับกำจัดผีภายในเอฟพีจีเอ วงจรรวม LM1881 ยังสามารถทำการคงค่าระดับแรงดัน (Clamped) ทำให้ระดับแรงดันตรง (DC Level) ที่ปลายซิงก์ (Sync Tip) มีค่าคงที่ประมาณ 1.5 V อยู่เสมอ สัญญาณภาพรวมที่ถูกคงค่าระดับแรงดันนี้ จะถูกป้อนให้กับวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลต่อไป รายละเอียดของการต่อวงจรรวม LM1881 แสดงในรูปที่ 3.2 และลักษณะของสัญญาณที่สำคัญแสดงไว้ดังรูปที่ 3.3



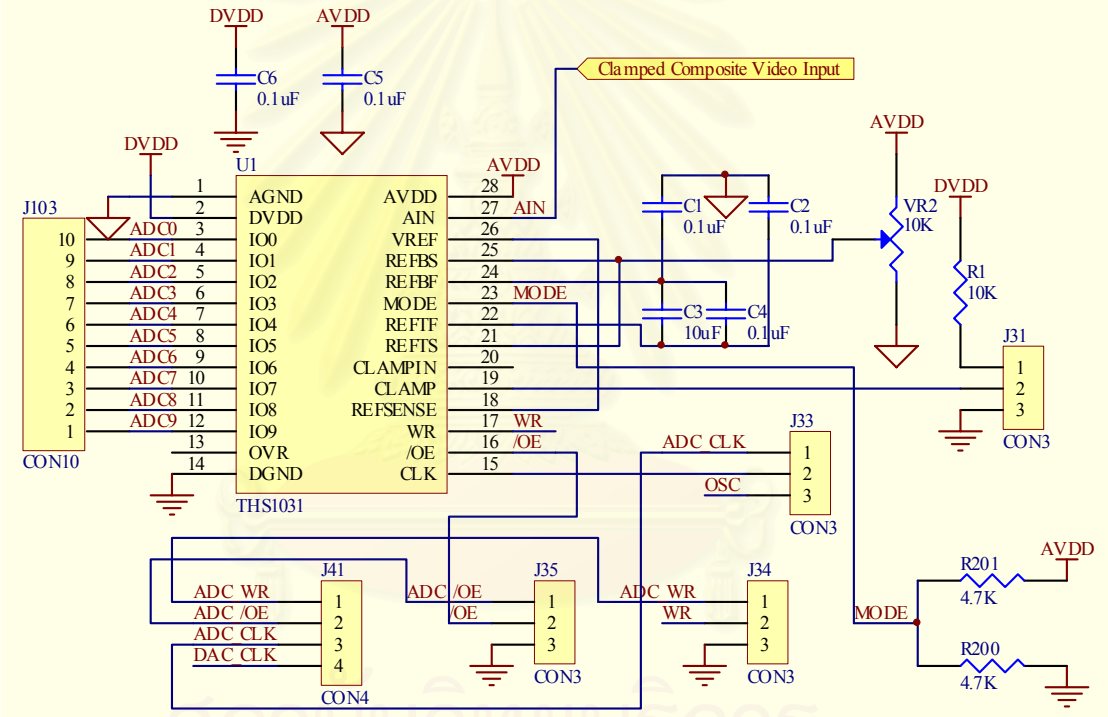
รูปที่ 3.2 แผนผังวงจรแยกสัญญาณซิงก์ด้วยวงจรรวม LM1881



รูปที่ 3.3 ลักษณะสัญญาณที่สำคัญของวงจรแยกสัญญาณซิงก์ด้วยวงจรรวม LM1881

3.3 วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล (A/D Converter)

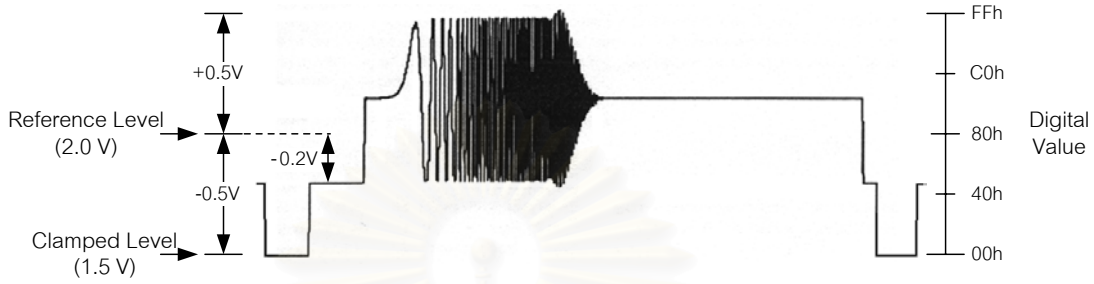
สัญญาณภาพที่ถูกคงค่าระดับแรงดันด้วยวงจรรวม LM1881 จะถูกป้อนให้กับวงจรรวม THS1031 [21] ของบริษัท Texas Instrument ซึ่งเป็นวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล ขนาด 10 บิต สามารถทำงานที่อัตราสุ่มข้อมูลสูงสุดเป็น 30 ล้านตัวอย่างต่อวินาที (Msamples/sec) และด้วยการต่อวงจรให้ขา Mode (ขาที่ 23) มีระดับแรงดันเป็น 1/2 ของระดับแรงดันไฟเลี้ยงด้านแอนาลอก (AVDD) และต่อขา VREF (ขาที่ 26) เข้ากับขา REFSENSE (ขาที่ 18) ดังรูปที่ 3.4 ทำให้โมดการทำงานของวงจรรวม THS1031 เป็นการแปลงสัญญาณที่เปลี่ยนแปลงในช่วง ±0.5 V รอบระดับแรงดันอ้างอิงค่าหนึ่งที่ป้อนให้กับทั้งขา REFBS (ขาที่ 25) และขา REFTS (ขาที่ 21) โดยขา REFBS และขา REFTS จะต้องถูกต่อถึงกัน (Short Circuit)



รูปที่ 3.4 แผนผังวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลด้วยวงจรรวม THS1031

เนื่องจากวงจรรวม LM1881 ทำการคงค่าระดับแรงดันสัญญาณภาพรวมไว้ที่ 1.5 V และสัญญาณภาพรวมมีขนาด 1 $V_{p\text{-}p}$ ดังนั้นจึงได้ใช้ความต้านทานปรับค่าได้ในการปรับระดับแรงดันอ้างอิงให้อยู่ที่ 2.0 V เพื่อให้วงจรรวม THS1031 แปลงสัญญาณภาพรวมที่เปลี่ยนแปลงในช่วง 1.5 V ถึง 2.5 V ให้เป็นค่าดิจิตอลขนาด 10 บิต แต่เพื่อประหยัดทรัพยากรภายในเอฟพีจีเอที่ใช้ในการสร้างวงจรกรอง ในงานวิจัยนี้จึงใช้เพียงข้อมูล 8 บิตบน (ADC[9..2]) เท่านั้น ซึ่งลักษณะของสัญญาณที่สำคัญแสดงในรูปที่ 3.5 สำหรับอัตราสุ่มข้อมูลตามมาตรฐาน ITU-R BT.1124 แนะนำให้ใช้อัตราสุ่มข้อมูลที่ 13.5 ล้านตัวอย่างต่อวินาที แต่เนื่องจากต้องการให้วงจรกรองมีผลตอบทาง

เวลาต่อเนื่องนานที่สุด จึงได้ลดอัตราสุ่มลงเหลือ 12 ล้านตัวอย่างต่อวินาที ซึ่งยังเป็นอัตราสุ่มที่สูงกว่าความถี่แบนด์ (Bandwidth) ของสัญญาณโทรทัศน์ระบบ PAL อยู่ 2 เท่าด้วย และได้ทดลองสร้างภาพโดยป้อนสัญญาณวีดิทัศน์ผ่านวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล และวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอกที่ทำงานที่อัตราสุ่มนี้ พบว่ามีความเพี้ยนไปน้อยมาก



รูปที่ 3.5 ลักษณะสัญญาณที่สำคัญของวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล

3.4 วงจรกรองเอฟไออาร์ และวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี (FIR Filters and GCR Data Separator)

วงจรกรองเอฟไออาร์ และวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี เป็นวงจรที่ทำงานกับข้อมูลความถี่สูงแบบเวลาจริง (Real Time) ซึ่งอยู่ภายในชิพเอฟพีจีเอ (FPGA : Field Programmable Gate Array) โดยใช้แบบจำลองวีเอชดีแอล (VHDL : VHSIC Hardware Description Language) และแบบจำลองผังงาน (Schematic) ช่วยในการออกแบบวงจร และตรวจสอบความถูกต้องด้วยการจำลองการทำงาน (Simulation) โดยใช้ซอฟท์แวร์ Xilinx Foundation 3.3i ของบริษัท Xilinx [22] เมื่อวงจรทำงานได้ถูกต้องแล้ว จึงสังเคราะห์วงจร (Synthesis) เป็นการเชื่อมต่อของอุปกรณ์ย่อยของชิพเอฟพีจีเอออกมา เพื่อบรรจุข้อมูลดังกล่าวลงบนชิพเอฟพีจีเอเบอร์ XCV300E-6 [23] เนื่องจากวงจรในส่วนนี้มีความซับซ้อนมาก จึงจะขอกล่าวถึงรายละเอียดของการออกแบบไว้ในบทที่ 4

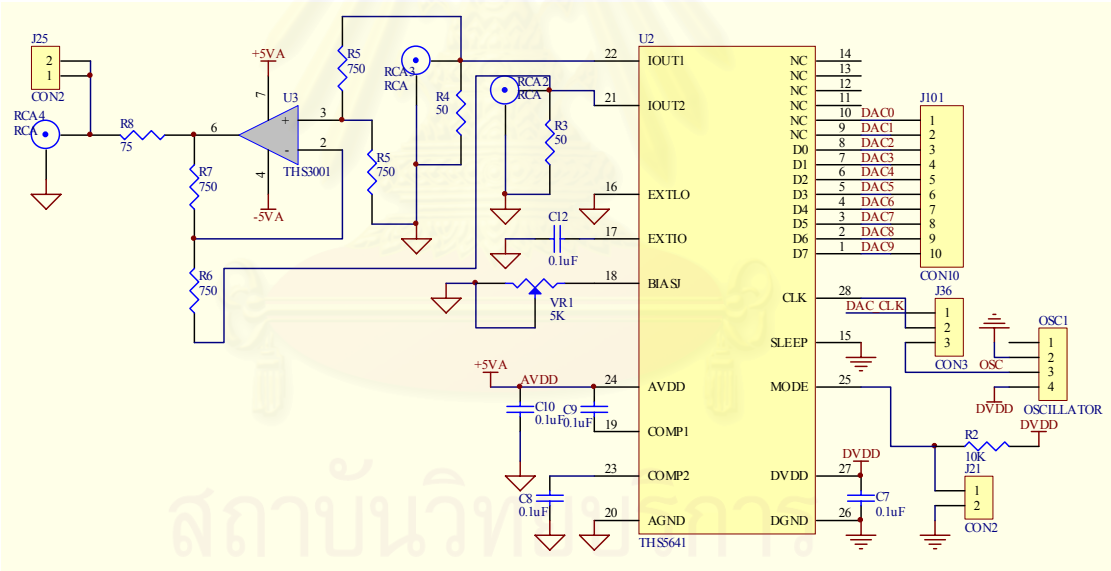
3.5 ตัวประมวลผลสัญญาณดิจิตอล (Digital Signal Processor)

ตัวประมวลผลสัญญาณดิจิตอล เป็นส่วนที่ทำหน้าที่คำนวณหาค่าสัมประสิทธิ์ของวงจรกรองที่ใช้ในการกำจัดผี และเนื่องจากในการคำนวณหาค่าสัมประสิทธิ์ต้องมีการคำนวณที่ซับซ้อนเป็นจำนวนมาก จึงต้องการตัวประมวลผลที่มีความเร็วในการคำนวณสูง ซึ่งได้เลือกใช้ตัวประมวลผลสัญญาณดิจิตอลเบอร์ TMS320C6211 ซึ่งอยู่ในชุดทดลอง TMS320C6211 DSP STARTER KIT [24] ของบริษัท Texas Instruments สำหรับกระบวนวิธีค่าเฉลี่ยกำลังสองน้อย

ที่สุด ที่ใช้ในการกำจัดผี และโปรแกรมกำจัดผีที่ทำงานบนตัวประมวลผลสัญญาณดิจิตอล จะขอกล่าวถึงรายละเอียดในบทที่ 5

3.6 วงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก และวงจรขยาย

สัญญาณภาพแบบดิจิตอลที่ผ่านกระบวนการกำจัดผีแล้ว จะถูกป้อนให้กับวงจรส่วนนี้ให้กลับมาเป็นสัญญาณภาพรวมแบบแอนาลอก โดยเลือกใช้วงจรรวม THS5641 [25] ของบริษัท Texas Instrument ซึ่งทำหน้าที่แปลงสัญญาณดิจิตอลขนาด 8 บิตที่อัตราสุ่มข้อมูลสูงสุดที่สามารถทำงานได้คือ 100 ล้านตัวอย่างต่อวินาที ให้เป็นสัญญาณแอนาลอกขนาด 1 V_{P-P} จากนั้นสัญญาณภาพรวมจะถูกป้อนให้กับวงจรขยายสัญญาณที่ต่อความต้านทานขาออก 75 Ω เพื่อเป็นการแมตช์อิมพีแดนซ์กับความต้านทานขาเข้าของเครื่องรับโทรทัศน์ที่มีค่า 75 Ω ซึ่งจะทำให้ขนาดของสัญญาณลดลง 1/2 เท่า ดังนั้นวงจรขยายสัญญาณดังกล่าวจึงได้ออกแบบให้มีอัตราขยายสัญญาณเป็น 2 เท่า รายละเอียดของวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก และวงจรขยายแสดงในรูปที่ 3.6



รูปที่ 3.6 แผนผังวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก และวงจรขยาย

3.7 สรุปท้ายบท

ในบทนี้ได้กล่าวถึงส่วนประกอบหลักภายในเครื่องกำจัดผีในสัญญาณโทรทัศน์ อันได้แก่ วงจรแยกซิงก์ ใช้วงจรรวม LM1881, วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล ใช้วงจรรวม THS1031 และวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก ใช้วงจรรวม THS5641 สำหรับรายละเอียดของวงจรวงจรที่อยู่ภายในเอฟพีจีเอ ซึ่งทำหน้าที่หลักเป็นวงจรกรองเอฟไออาร์ และ

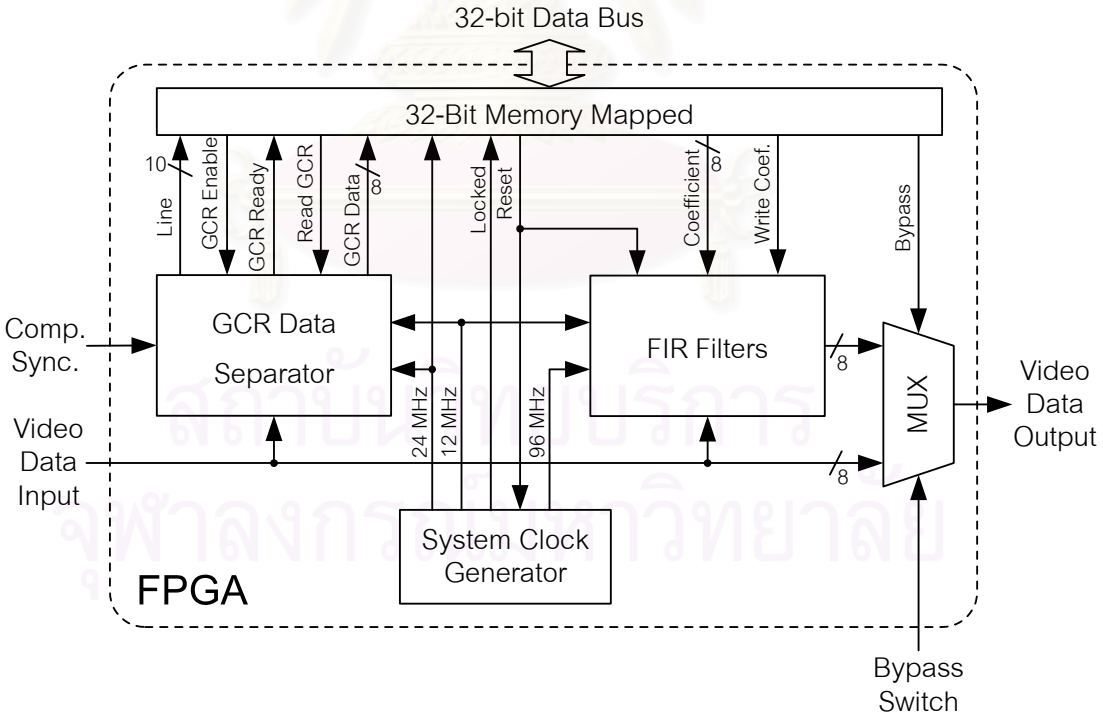
วงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผีจะได้กล่าวถึงในบทที่ 4 และโปรแกรมกำจัดผีที่ทำงานบนตัวประมวลผลสัญญาณดิจิตอลจะได้กล่าวถึงในบทที่ 5

บทที่ 4

วงจรกรองเอฟไออาร์ และวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี

4.1 โครงสร้างของวงจรภายในเอฟพีจีเอ

โครงสร้างของวงจรภายในเอฟพีจีเอ นอกจากจะประกอบด้วย 2 ส่วนหลักคือ วงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี และวงจรกรองเอฟไออาร์ดังที่ได้กล่าวมาแล้ว ยังมีส่วนประกอบอื่นอันได้แก่ วงจรสร้างสัญญาณนาฬิกาของระบบ, วงจรแปลงการอ้างอิงหน่วยความจำเป็นสัญญาณควบคุม และวงจรมัลติเพลกซ์ ดังรูปที่ 4.1 วงจรบางส่วนได้ใช้แบบจำลองภาษาวีเอชดีแอล (VHDL) ช่วยในการออกแบบ (Design Entry) และบางส่วนได้ใช้แผนภาพผังงาน (Schematic) ช่วยในการออกแบบ จากนั้นตรวจสอบความถูกต้องด้วยการจำลองการทำงาน (Simulation) แล้วจึงนำไปสังเคราะห์วงจร เพื่อโปรแกรมลงชิพเอฟพีจีเอเบอร์ XCV300E-6 [23] หลักการทำงานโดยรวมของวงจรทั้งหมดเป็นดังนี้

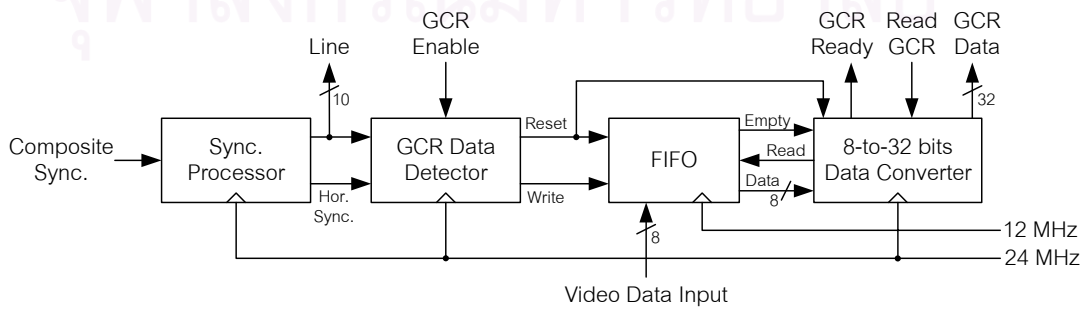


รูปที่ 4.1 โครงสร้างของวงจรภายในเอฟพีจีเอ

สัญญาณซิงก์รวม (Composite Sync) ที่ป้อนเข้ามาจะถูกวิเคราะห์หาข้อมูลทางเวลาของสัญญาณภาพรวม เพื่อใช้ในการเก็บข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี ซึ่งถูกส่งมาพร้อมสัญญาณภาพเส้นที่ 318 (เส้นที่ 5 ของฟิลด์ที่ 2) แล้วรอให้ตัวประมวลผลสัญญาณดิจิตอลนำไปใช้ในการคำนวณ และเมื่อได้ค่าสัมประสิทธิ์ที่เหมาะสมในการกำจัดผีแล้ว ตัวประมวลผลสัญญาณดิจิตอลจะโปรแกรมค่าสัมประสิทธิ์ดังกล่าว ให้กับวงจรกรองเอฟไอาร์เพื่อใช้ในการกำจัดผี โดยสัญญาณควบคุม และข้อมูลต่างๆ จะถูกแปลงให้อยู่ในลักษณะของการอ้างอิงหน่วยความจำขนาด 32 บิต เพื่อใช้ในการติดต่อกับชิพประมวลผลสัญญาณดิจิตอล ซึ่งนอกจากผู้ใช้จะสามารถควบคุมให้เครื่องกำจัดผีปล่อยสัญญาณขาเข้าผ่านออกไปได้ทางสวิตช์ปล่อยผ่าน ในกรณีที่สัญญาณภาพที่ถูกกำจัดผีมีความคมชัดน้อยกว่าสัญญาณภาพขาเข้าแล้ว ตัวประมวลผลสัญญาณดิจิตอลยังสามารถเลือกให้ข้อมูลสัญญาณภาพขาเข้าผ่านออกไปโดยไม่ผ่านวงจรกรองเอฟไอาร์ได้ด้วย โดยควบคุมผ่านทางสัญญาณปล่อยผ่าน (Bypass) เพื่อป้องกันสัญญาณที่ผิดปกติในช่วงการโปรแกรมค่าสัมประสิทธิ์ของวงจรกรอง

4.2 วงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี

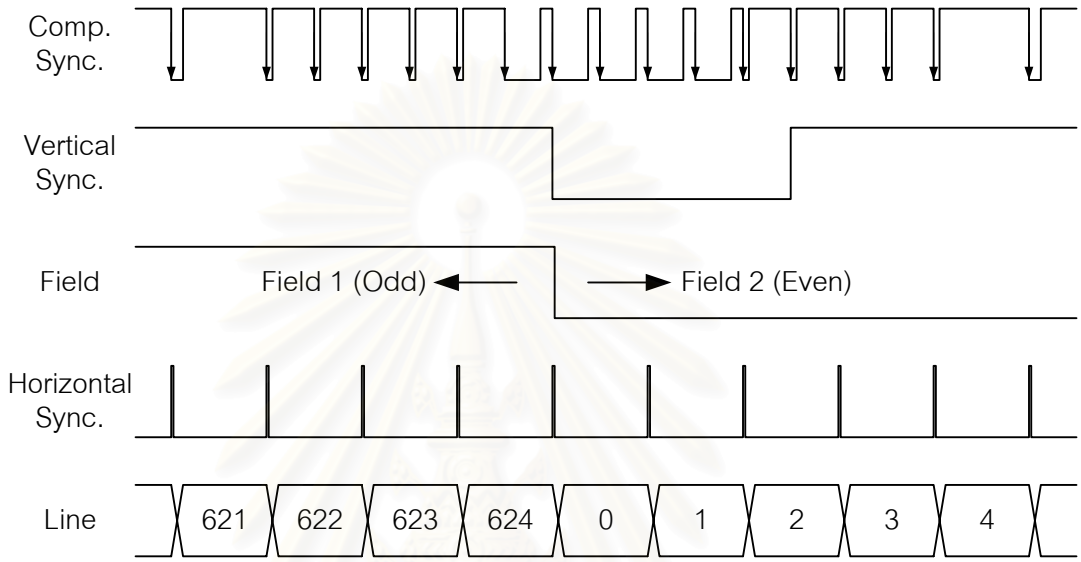
วงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี (GCR Data Seperator) ซึ่งมีโครงสร้างของวงจรดังรูปที่ 4.2 ทำหน้าที่เตรียมข้อมูลสัญญาณอ้างอิงให้กับตัวประมวลผลสัญญาณดิจิตอลเพื่อนำไปใช้ในการคำนวณ โดยตัวประมวลผลซิงก์ (Sync Processor) จะวิเคราะห์สัญญาณซิงก์ เพื่อบอกจังหวะต่างๆ ของสัญญาณภาพรวม รวมทั้งตำแหน่งของเส้นภาพ และตัวตรวจจับข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี จะคอยจนถึงเส้นที่ 4 ของฟิลด์ที่ 2 (เส้นภาพที่ 317) จากนั้นจะให้กำเนิดสัญญาณเขียนข้อมูล (Write) เพื่อสั่งให้หน่วยความจำแบบเข้าก่อนออกก่อน (FIFO Memory) ทำการเก็บข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผีลงในหน่วยความจำ และเมื่อมีข้อมูลในหน่วยความจำ ตัวแปลงขนาดข้อมูล 8 เป็น 32 บิต (8-to-32 bits Data Converter) จะทำการอ่านข้อมูลในหน่วยความจำแบบเข้าก่อนออกก่อนทีละ 8 บิตมาเรียงใหม่ให้เป็น 32 บิต เพื่อเตรียมให้ตัวประมวลผลสัญญาณดิจิตอลมาอ่านออกไปเพื่อใช้ในการคำนวณต่อไป



รูปที่ 4.2 โครงสร้างภายในของวงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี

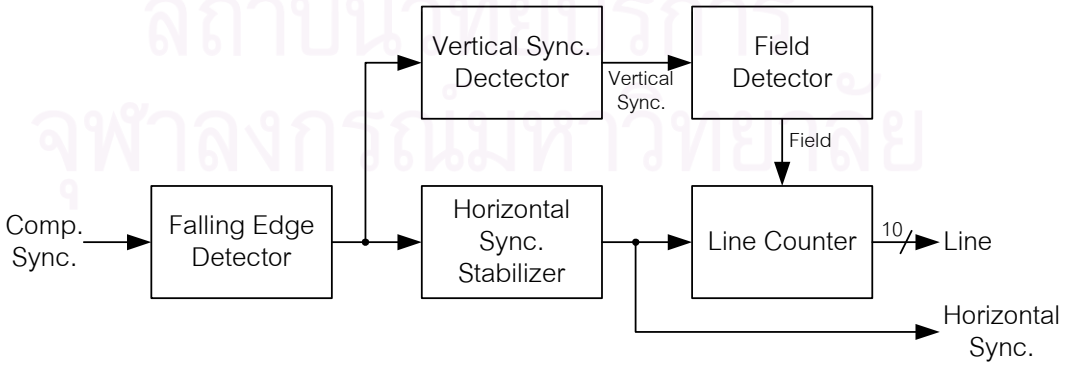
4.2.1 ตัวประมวลผลสัญญาณซิงก์ (Sync Processor)

ตัวประมวลผลสัญญาณซิงก์ ทำหน้าที่วิเคราะห์สัญญาณซิงก์รวมที่ได้รับมา เพื่อบอกจังหวะต่างๆ ของสัญญาณภาพรวม อันได้แก่ ซิงก์แนวตั้ง (Vertical Sync), ซิงก์แนวนอน (Horizontal Sync), เส้น (Line) และฟิลด์ (Field) โดยมีลักษณะของสัญญาณบอกจังหวะต่างๆ เป็นดังรูปที่ 4.3



รูปที่ 4.3 สัญญาณบอกจังหวะที่เกิดจากตัวประมวลผลซิงก์

เนื่องจากวงจรภายในประกอบด้วยวงจรนับเป็นจำนวนมาก หากใช้ความถี่ที่สูงเกินไปจะทำให้สิ้นเปลืองทรัพยากรภายในเอฟพีจีเอเป็นจำนวนมาก ดั้งนั้นวงจรภายในของตัวประมวลผลสัญญาณซิงก์จึงทำงานที่ความถี่ 24 MHz มีโครงสร้างของวงจรเป็นดังรูปที่ 4.4 โดยสัญญาณแต่ละตัว มีความสำคัญ และกรรมวิธีในการสร้างดังต่อไปนี้



รูปที่ 4.4 วงจรภายในของวงจรประมวลผลสัญญาณซิงก์

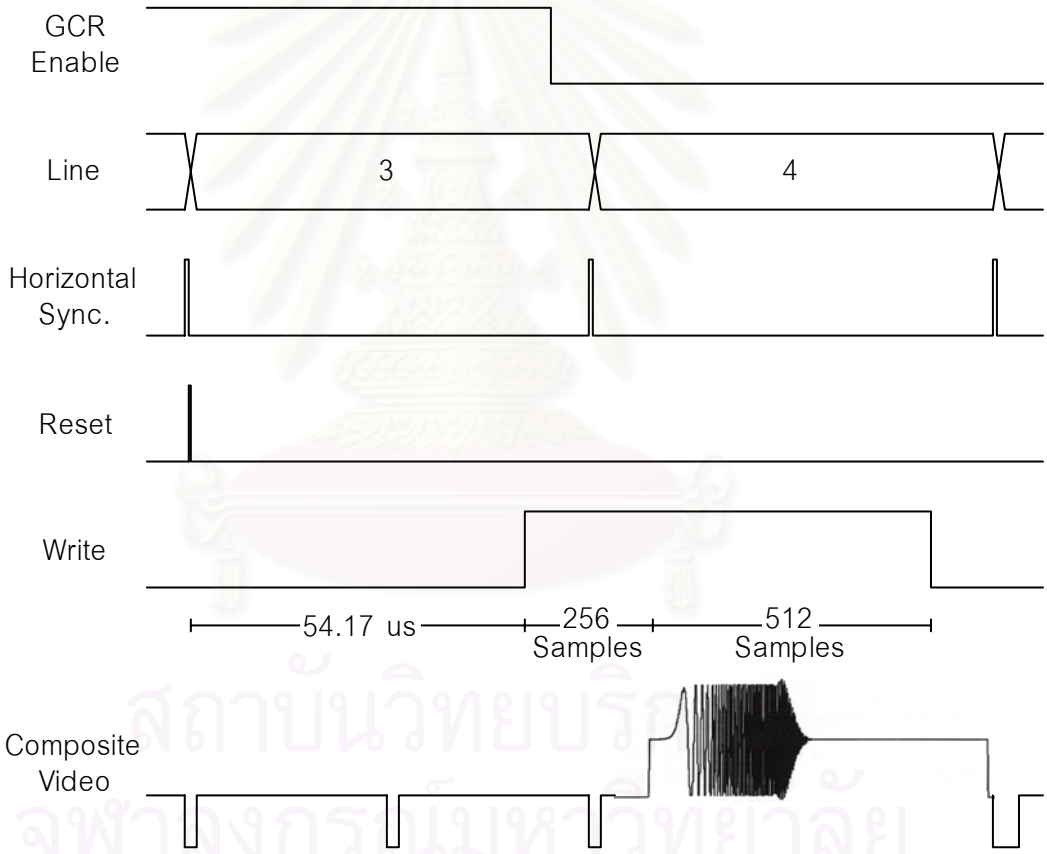
1. *ตัวตรวจจับขอบขาลง (Falling Edge Detector)* เป็นวงจรตรวจจับขอบขาลงของซิงก์รวม แล้วกำเนิดพัลส์ ‘1’ ออกมากว้าง 1 คาบสัญญาณนาฬิกาที่ความถี่ 24 MHz (41.67 ns)
2. *ตัวตรวจหาซิงก์แนวตั้ง (Vertical Sync Detector)* เป็นวงจรตรวจหาช่วงที่ซิงก์รวมมีค่าเป็น ‘0’ นานกว่า 24 μs และเป็น ‘1’ น้อยกว่า 8 μs แล้วจึงกำเนิดสัญญาณซิงก์แนวตั้งให้มีค่าเป็น ‘0’ ที่ขอบขาลงของซิงก์รวม
3. *ตัวตรวจสอบฟิลด์ (Field)* เป็นวงจรที่ทำหน้าที่ตรวจสอบว่าสัญญาณภาพที่ป้อนเข้ามาขณะนั้นว่าเป็นฟิลด์ใด โดยการนับจำนวนพัลส์ปรับเทียบ (Equalizing Pulse) ก่อนที่ซิงก์แนวตั้งจะเป็น ‘0’ หากมีจำนวนเป็นเลขคู่ แสดงว่าฟิลด์ถัดไปเป็นฟิลด์คู่ (Even Field) ซึ่งการตรวจสอบหาช่วงพัลส์ปรับเทียบทำได้โดย จับเวลาระหว่างขอบขาลงของซิงก์รวม หากน้อยกว่า 48 μs จำนวน 4 ครั้งขึ้นไป แสดงว่าเป็นช่วงพัลส์ปรับเทียบ และหากมากกว่า 48 μs แสดงว่าไม่ใช่ช่วงพัลส์ปรับเทียบ จะทำการตั้งค่าของวงจรนับให้เป็น ‘1’
4. *ตัวรักษาเสถียรภาพซิงก์แนวนอน (Horizontal Sync Stabilizer)* หากวงจรตรวจจับขอบขาลงของสัญญาณซิงก์รวม ถึงสัญญาณพัลส์ที่มีระยะห่างจากสัญญาณพัลส์ลูกที่แล้วในช่วง 63 ถึง 65 μs วงจรนี้จะสร้างพัลส์ ‘1’ ออกมากว้าง 1 คาบสัญญาณนาฬิกาตามระยะห่างที่ได้รับ แต่ในกรณีที่ไม่พบขอบขาลงของสัญญาณซิงก์รวมในช่วงระยะเวลาดังกล่าว ถือว่าเกิดการขาดหายไปของสัญญาณซิงก์ วงจรจะทำการสร้างสัญญาณพัลส์ทดแทนขึ้นมาที่ระยะห่าง 65 μs และชดเชยค่าการนับในเส้นถัดไป 1 μs แต่ไม่เกิน 3 ครั้งติดต่อกัน และจะสร้างพัลส์ออกมาใหม่เมื่อพบขอบขาลงของสัญญาณซิงก์ครั้งต่อไป เพื่อลดความผิดพลาดในการนับเส้นภาพ
5. *ตัวนับเส้นภาพ (Line Counter)* เป็นวงจรนับสัญญาณพัลส์ของซิงก์แนวนอน โดยจะถูกตั้งค่าเป็น 0 ใหม่ทุกครั้งที่พบขอบขาลงของสัญญาณฟิลด์ กล่าวคือสัญญาณเส้น (Line) จะเริ่มนับค่า 0 ที่ต้นฟิลด์คู่ (Even Field) เนื่องจากสัญญาณอ้างอิงสำหรับกำจัดผีถูกแทรกมาในเส้นภาพที่ 318 หรือเส้นภาพที่ 5 ของฟิลด์ที่คู่ ดังนั้นได้ตั้งค่าเริ่มต้นใหม่ที่ต้นฟิลด์คู่ เพื่อให้การตรวจจับสัญญาณอ้างอิงมีเสถียรภาพมากขึ้น

4.2.2 ตัวตรวจหาสัญญาณอ้างอิงสำหรับกำจัดผี (GCR Data Detector)

ตัวตรวจหาสัญญาณอ้างอิงสำหรับกำจัดผี มีหน้าที่ในการระบุตำแหน่งของสัญญาณอ้างอิงแล้วทำการสร้างสัญญาณรีเซต (Reset) และสัญญาณเขียน เพื่อใช้ในการเก็บข้อมูลสัญญาณ

อ้างอิงลงหน่วยความจำแบบเข้าก่อนออกก่อน โดยลักษณะของสัญญาณต่างๆ ที่สำคัญแสดงไว้ดังรูปที่ 4.5 ซึ่งมีหลักการทำงานดังนี้

เมื่อตัวประมวลผลสัญญาณดิจิตอลอนุมัติให้เก็บข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี (สัญญาณ GCR Enable มีค่าเป็น '1') ตัวตรวจหาสัญญาณอ้างอิงสำหรับกำจัดผีจะคอยจนถึงเส้นที่ 2 พร้อมกับสัญญาณซิงก์แนวนอน ซึ่งบอกถึงจุดเริ่มต้นของเส้นที่ 3 แล้วทำการสร้างสัญญาณรีเซต (Reset) เพื่อให้หน่วยความจำแบบเข้าก่อนออกก่อนว่าง (Empty) จากนั้นทำการหน่วงเวลาไป 54.17 μs แล้วทำการสร้างสัญญาณเขียน (Write) เพื่อเขียนข้อมูลจำนวน 768 ไบต์ลงหน่วยความจำแบบเข้าก่อนออกก่อน โดยที่ข้อมูลทั้งหมดจะแบ่งเป็น 2 ชุดคือ 256 ไบท์ และ 512 ไบท์ ซึ่งจะได้กล่าวถึงรายละเอียดในบทที่ 5



รูปที่ 4.5 สัญญาณแสดงการทำงานของตัวตรวจหาสัญญาณอ้างอิงสำหรับกำจัดผี

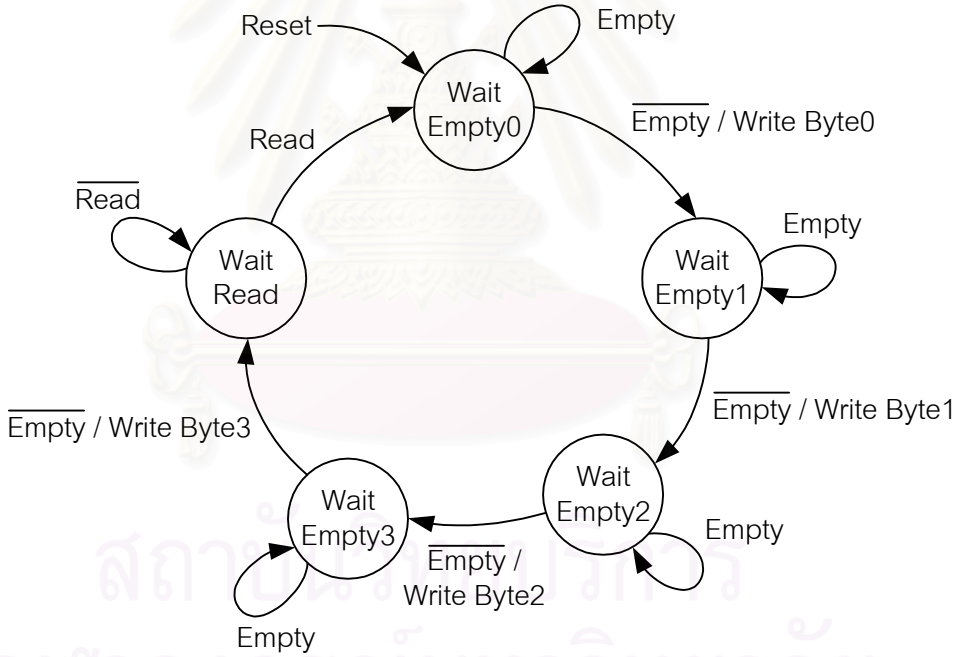
4.2.3 <u>หน่วยความจำแบบเข้าก่อนออกก่อน (FIFO Memory)</u>

หน่วยความจำแบบเข้าก่อนออกก่อนถูกนำมาใช้ในการเก็บข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผีจำนวน 768 ไบต์ เพื่อเตรียมไว้ให้กับตัวประมวลผลสัญญาณดิจิตอลนำไปใช้ในการ

คำนวณต่อไป หน่วยความจำแบบเข้าก่อนออกก่อนนี้สามารถสร้างได้โดยใช้ซอฟท์แวร์ Core Generator [26] ที่มีมาพร้อมกับซอฟท์แวร์ Xilinx Foundation ซึ่งภายในซอฟท์แวร์ Core Generator สามารถกำหนดให้ใช้บล็อกหน่วยความจำที่มีอยู่ภายในเอฟพีจีเอเบอร์ XCV300E-6 ในการสร้างหน่วยความจำแบบเข้าก่อนออกก่อนได้ จะทำให้ไม่สิ้นเปลืองเนื้อที่ในส่วนแอลยูที (LUT) ของเอฟพีจีเอ โดยรายละเอียดการกำหนดค่าต่างๆ แสดงไว้ในภาคผนวก ก

4.2.4 ตัวแปลงขนาดข้อมูล 8 บิตเป็น 32 บิต (8-to-32 bits Data Converter)

เนื่องจากตัวประมวลผลสัญญาณดิจิตอลจะควบคุม อ่านและเขียนข้อมูลบนเอฟพีจีเอผ่านบัสข้อมูล (Data Bus) ขนาด 32 บิต (ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อ 4.5) และเพื่อลดเวลาในการอ่านข้อมูลสัญญาณอ้างอิงของตัวประมวลผลสัญญาณดิจิตอล จึงต้องทำการแปลงข้อมูลสัญญาณอ้างอิงที่เก็บในหน่วยความจำแบบเข้าก่อนออกก่อนขนาด 8 บิต เป็น 32 บิต ซึ่งมีการทำงานแสดงได้ดังรูปที่ 4.6 และสามารถอธิบายการทำงานได้ดังนี้



รูปที่ 4.6 แผนภาพสถานะการทำงานของตัวแปลงขนาดข้อมูล 8 บิตเป็น 32 บิต

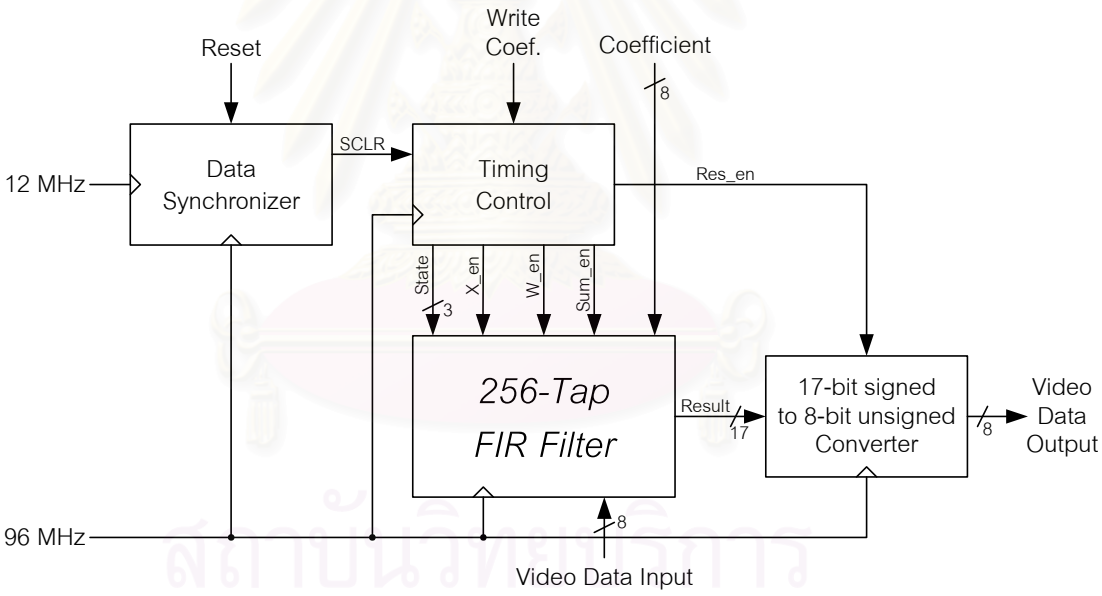
ตัวแปลงขนาดข้อมูลจะถูกรีเซทพร้อมกับการรีเซทหน่วยความจำแบบเข้าก่อนออกก่อนให้ว่าง (สถานะ Wait Empty) แล้วคอยจนกระทั่งมีข้อมูลเข้ามาในหน่วยความจำ จึงทำการอ่านข้อมูลจากหน่วยความจำออกมาเก็บไว้ที่ไบต์ 0 ในรีจิสเตอร์ขนาด 32 บิต (สถานะ Write Byte0) และกลับไปคอยข้อมูลในหน่วยความจำเพื่ออ่านออกมาเก็บไว้ที่ไบต์ 1 (สถานะ Write Byte1) ทำอย่างนี้ซ้ำเรื่อยไปจนครบ 4 ไบต์ (32 บิต) แล้วจึงส่งสัญญาณพร้อม (GCR Ready) เป็น ‘1’ และ

คอยให้ตัวประมวลผลสัญญาณดิจิตอลมาอ่านข้อมูลในรีจิสเตอร์ขนาด 32 บิตออกไป (สถานะ Wait Read) เมื่อข้อมูลถูกอ่านออกไปแล้วก็วนกลับมาอ่านข้อมูลจากหน่วยความจำอีก

4.3 วงจรกรองเอฟไอออาร์

ตารางที่ 4.1 ประสิทธิ์ภาพของวงจรกรองเอฟไอออาร์ขนาด 32 แท็ป

เทคนิคที่ใช้	ทรัพยากรที่ใช้ไป	อัตราเร็วของวงจร
วงจรคูณแบบธรรมดา	54%	75.7 MHz
วงจรคูณแบบไปป์ไลน์ 5 ระดับ	60%	144.4 MHz



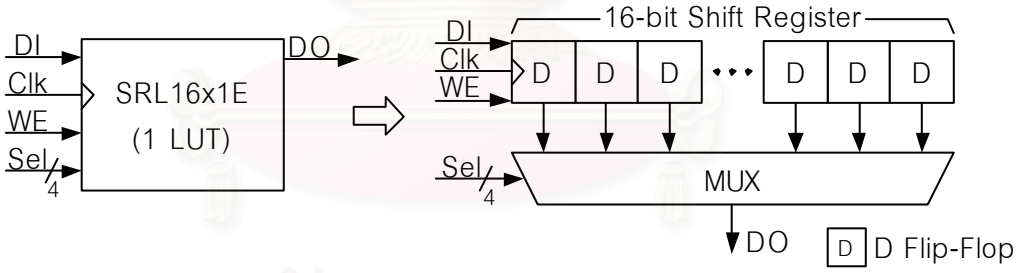
รูปที่ 4.7 ส่วนประกอบของวงจรกรองเอฟไอออาร์

วงจรกรองเอฟไอออาร์เป็นส่วนประกอบที่มีความสำคัญมากส่วนหนึ่งภายในเครื่องกำจัดผี และเป็นส่วนที่ใช้ทรัพยากรภายในชิพเอฟพีจีเอมากที่สุด เนื่องจากมีองค์ประกอบจากวงจรคูณและวงจรบวกจำนวนมาก และจากการทดลองสร้างวงจรกรองเอฟไอออาร์ขนาด 32 แท็ป เพื่อเปรียบเทียบกันระหว่างใช้วงจรคูณแบบธรรมดา กับวงจรคูณแบบไปป์ไลน์ [27] (Pipeline Multiplier) ได้ผลดังตารางที่ 4.1 พบว่าวงจรคูณแบบไปป์ไลน์ 5 ระดับใช้ทรัพยากรภายในเอฟพีจี-เอเพิ่มขึ้นเล็กน้อยจากวงจรคูณปกติ แต่สามารถเพิ่มความเร็วของวงจรคูณได้สูงกว่า 140 MHz ซึ่ง

สูงกว่าอัตราสุ่มข้อมูลที่ 12 ล้านตัวอย่างต่อวินาทีอยู่มาก จึงได้นำเทคนิคการใช้ทรัพยากรร่วม (Resource Sharing Technique) เข้ามาช่วยในการประหยัดทรัพยากร กล่าวคือ วงจรคูณและวงจรบวกอย่างละ 1 ชุด จะใช้ในการคำนวณข้อมูล 8 ชุดแต่คนละเวลา กล่าวได้ว่ามีการประหยัดทรัพยากรได้เกือบแปดเท่า โครงสร้างของวงจรกรองเอฟไอาร์แสดงได้ดังรูปที่ 4.7 โดยการทำงานของส่วนต่างๆ ภายในวงจรอธิบายได้ดังนี้

4.3.1 วงจรกรองเอฟไอาร์ขนาด 256 แท็ป (256-Tap FIR Filter)

การนำเทคนิคการใช้ทรัพยากรร่วมมาใช้กับวงจรกรองเอฟไอาร์นั้น ทำได้โดยการเพิ่มวงจรมัลติเพลกซ์เข้าไป เพื่อใช้เลือกข้อมูลจากวงจรเลื่อนบิตภายในวงจรกรอง และเนื่องจากโครงสร้างภายในของเอฟพีจีเอที่เป็นแบบแอลยูที (LUT : Look-Up Table) ขนาด 4 อินพุต จึงสามารถออกแบบให้ภายในแอลยูทีมีลักษณะการทำงานเป็นวงจรเลื่อนบิตข้อมูล (Shift Register) กับวงจรมัลติเพลกซ์ (Multiplexer) ซึ่งมีชื่อย่อว่า SRL16x1E ได้ดังรูปที่ 4.8 โดย 1 แอลยูทีจะใช้เก็บข้อมูลที่มีความกว้าง 1 บิต และมีความลึกได้ 16 ขั้น สำหรับวงจรกรองเอฟไอาร์ที่ใช้ทรัพยากรร่วม 8 ครั้งนี้ ต้องการวงจรเลื่อนบิตที่มีความกว้าง 8 บิต และมีความลึก 8 ขั้น สามารถสร้างได้โดยนำวงจร SRL16x1E จำนวน 8 ตัวมาต่อขนานกัน และป้อนค่า '0' ให้กับบิตบนสุดของสัญญาณเลือกข้อมูล (Sel) เรียกว่า SRL8x8E

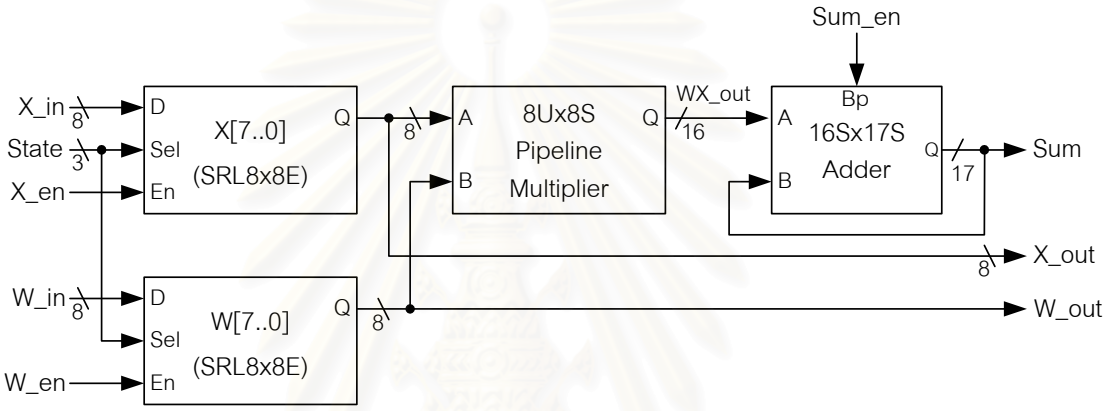


รูปที่ 4.8 วงจรแอลยูทีในลักษณะ SRL16x1E

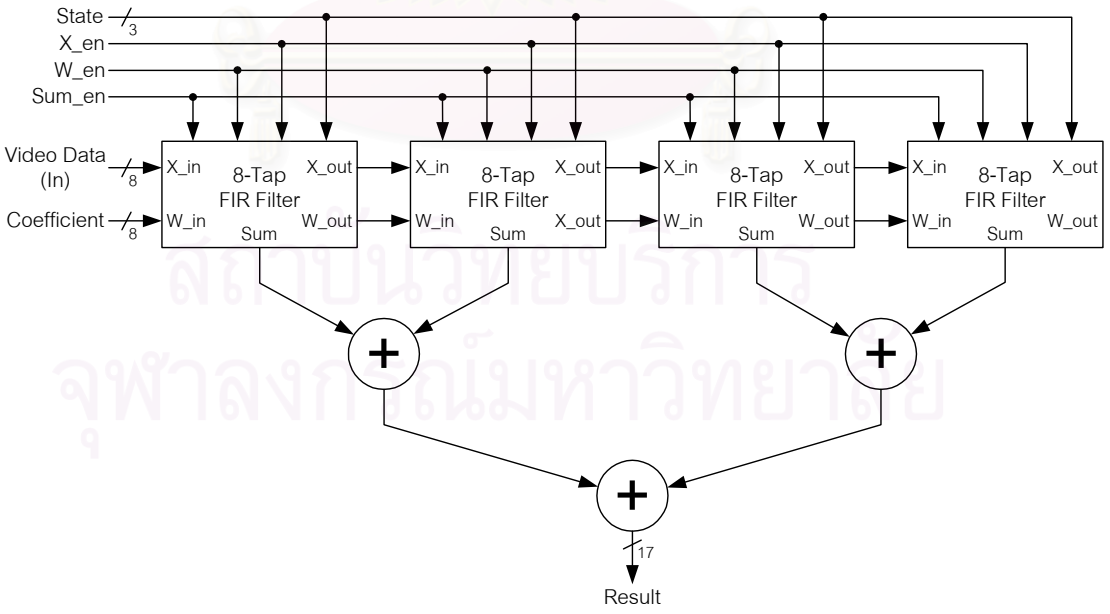
อีกส่วนหนึ่งที่สำคัญในการสร้างวงจรกรองเอฟไอาร์ที่ใช้ทรัพยากรร่วม 8 ครั้งคือ วงจรคูณแบบไปป์ไลน์ และวงจรบวกที่สามารถทำงานได้ที่ความถี่สูงกว่า 96 MHz (เนื่องจากอัตราสุ่มข้อมูลอยู่ที่ 12 ล้านตัวอย่างต่อวินาที และใช้ทรัพยากรร่วม 8 ครั้ง) สร้างได้โดยใช้ซอฟท์แวร์ IP Core Generator ที่มาพร้อมกับซอฟท์แวร์ Xilinx Foundation 3.3i โดยการกำหนดค่าต่างๆ ในการสร้างวงจรคูณแบบไปป์ไลน์ และวงจรบวกนั้น สามารถอ่านรายละเอียดเพิ่มเติมได้ในภาคผนวก ก

วงจรกรองเอฟไอาร์สามารถแบ่งได้ 2 แบบตามลักษณะโครงสร้างของวงจรกรองคือ โครงสร้างแบบดังเดิม และโครงสร้างแบบทรานซ์โพส (Transposed Form) [28] วงจรกรองแบบ

ดังเดิมที่ใช้ทรัพยากรร่วม 8 ครั้ง ความยาว 8 แท็ป มีโครงสร้างเป็นดังรูปที่ 4.9 ซึ่งใช้วงจร SRL8x8E 2 ชุดในการเก็บข้อมูลสัญญาณภาพ และค่าสัมประสิทธิ์ของวงจรกรอง จากนั้นเอาต์พุตของวงจร SRL8x8E ทั้งสองวงจรถูกป้อนให้กับวงจรคูณไปป์ไลน์ ที่ทำการคูณระหว่างข้อมูลสัญญาณภาพขนาด 8 บิตแบบไม่มีเครื่องหมาย กับค่าสัมประสิทธิ์ของวงจรกรอง 8 บิตแบบมีเครื่องหมาย ทำให้ได้ผลลัพธ์ของวงจรคูณเป็นค่า 16 บิตแบบมีเครื่องหมาย ซึ่งจะถูกป้อนให้กับวงจรบวกข้อมูล โดยสามารถเลือกให้ผ่านค่าที่พอร์ทขาเข้า A ออกไปที่พอร์ทขาออก Q ได้ ด้วยสัญญาณควบคุมการเริ่มหาผลรวม (Sum_en) หรือทำการบวกค่าผลคูณกับผลรวมครั้งก่อนหน้าได้ ทำให้สามารถหาผลรวมของการคูณข้อมูลทั้ง 8 ชุดได้

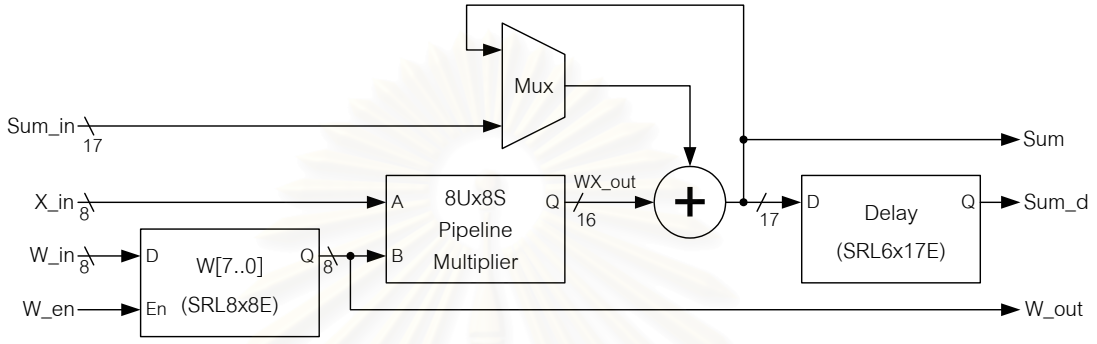


รูปที่ 4.9 วงจรกรองเอฟไอาร์ที่ใช้ทรัพยากรร่วมความยาว 8 แท็ป



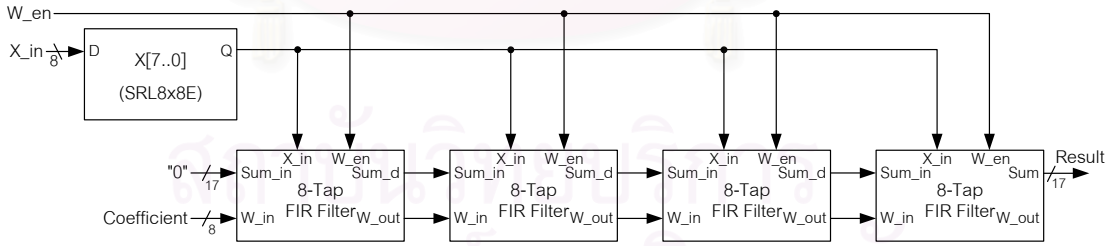
รูปที่ 4.10 วงจรกรองเอฟไอาร์ที่ใช้ทรัพยากรร่วมความยาว 32 แท็ป

และเมื่อต้องการวงจรกรองเอฟไอาร์ที่มีความยาวมากกว่า 8 แท็ป สามารถสร้างได้โดยการนำวงจรกรองเอฟไอาร์ความยาว 8 แท็ปดังกล่าว มาต่อกันแบบอนุกรมดังรูปที่ 4.10 เป็นวงจรกรองเอฟไอาร์ความยาว 32 แท็ป คือข้อมูลสัญญาณภาพและค่าสัมประสิทธิ์ขาออกของวงจรกรองชุดก่อน จะป้อนต่อไปให้เป็นข้อมูลและค่าสัมประสิทธิ์ขาเข้าของวงจรกรองชุดถัดไป สำหรับผลรวมของวงจรกรองแต่ละชุดจะนำมาผ่านวงจรบวก เพื่อหาผลรวมทั้งหมดอีกครั้งหนึ่ง



รูปที่ 4.11 วงจรกรองเอฟไอาร์แบบทรานส์โพสที่ใช้ทรัพยากรร่วมความยาว 8 แท็ป

วงจรกรองเอฟไอาร์แบบทรานซ์โพสที่ใช้ทรัพยากรร่วม 8 ครั้ง ความยาว 8 แท็ป มีโครงสร้างเป็นดังรูปที่ 4.11 ซึ่งใช้วงจร SRL8x8E เพียงชุดเดียว แต่จะมีวงจร SRL6x17E ใช้ในการหน่วงผลลัพธ์ของวงจรกรองเพิ่มขึ้นมาแทน อีกทั้งยังมีตัวมัลติเพลกซ์ที่ใช้เลือกผลรวมของวงจรกรองชุดก่อนหน้า กับผลรวมปัจจุบันเพิ่มขึ้นมา ซึ่งจะทำให้อัตราเร็วของวงจรลดลง เนื่องจากเส้นทางในส่วนของวงจรบวกนั้นยาวขึ้น



รูปที่ 4.12 วงจรกรองเอฟไอาร์แบบทรานส์โพสที่ใช้ทรัพยากรร่วมความยาว 32 แท็ป

ในการสร้างวงจรกรองที่มีความยาวสูงขึ้น ทำได้โดยการนำวงจรกรองแบบทรานส์โพส ความยาว 8 แท็ปดังกล่าวมาต่อกันดังรูปที่ 4.12 เป็นวงจรกรองเอฟไอาร์แบบทรานส์โพสที่มีความยาว 32 แท็ป จะสังเกตว่าวงจรกรองแบบทรานส์โพสนี้ มีโครงสร้างที่ง่ายกว่าแบบดั้งเดิม กล่าวคือผลรวมของวงจรกรองชุดหนึ่ง จะถูกส่งไปคำนวณต่อในชุดถัดไป ทำให้ไม่ต้องสร้างวงจรบวกเพิ่ม

ในการหาผลรวมทั้งหมดของวงจรกรองเหมือนกับวงจรกรองแบบดั้งเดิม จึงเป็นข้อดีของโครงสร้างแบบทรานส์โพสที่ง่ายต่อการเพิ่มความยาวของวงจรกรอง

ผลการเปรียบเทียบประสิทธิภาพของวงจรกรองแบบดั้งเดิม กับแบบทรานส์โพส เป็นดังตารางที่ 4.2 ถึงแม้ว่าที่ความยาวของวงจรกรองเท่ากัน โครงสร้างแบบทรานส์โพสจะใช้ทรัพยากรน้อยกว่าแบบดั้งเดิม แต่อัตราเร็วของวงจรกรองแบบทรานส์โพสก็ต่ำกว่าของวงจรกรองแบบดั้งเดิมอยู่มาก ดังนั้นหากคำนึงถึงเวลาที่รวมวงจรทั้งหมดภายในเอฟพีจีเอเข้าด้วยกันแล้ว จะทำให้พื้นที่ว่างภายในเอฟพีจีเอน้อยลง และอัตราเร็วของวงจรก็จะลดลงตามไปด้วย ซึ่งทำให้โครงสร้างแบบทรานส์โพสไม่สามารถทำงานที่ความถี่มากกว่า 96 MHz ตามที่ต้องการได้ ดังนั้นในงานวิจัยนี้จึงใช้โครงสร้างแบบดั้งเดิมในการสร้างวงจรกรองแบบเอฟไออาร์

ตารางที่ 4.2 เปรียบเทียบประสิทธิภาพของวงจรกรองแบบดั้งเดิมกับแบบทรานส์โพส

ลักษณะของวงจรกรอง	ความยาว (แท็ป)	ทรัพยากรที่ใช้	อัตราเร็วสูงสุดของ	
			วงจรกรอง (MHz)	การสุ่มข้อมูล (MSamples/sec)
โครงสร้างแบบดั้งเดิม	32	60%	144.4	144.4
โครงสร้างแบบดั้งเดิม โดยใช้ทรัพยากรร่วม 8 ครั้ง	32	9%	147.1	18.3
โครงสร้างแบบทรานส์โพส โดยใช้ทรัพยากรร่วม 8 ครั้ง	32	9%	124.3	15.6
โครงสร้างแบบดั้งเดิม โดยใช้ทรัพยากรร่วม 8 ครั้ง	256	73%	141.9	17.7
โครงสร้างแบบทรานส์โพส โดยใช้ทรัพยากรร่วม 8 ครั้ง	256	68%	112.8	14.1

ในการกำจัดผีในสัญญาณโทรทัศน์นั้น ความยาวของวงจรกรองเอฟไออาร์เป็นตัวบ่งบอกถึงความหน่วงเวลาของผีที่สามารถกำจัดได้ นั่นคือยิ่งวงจรกรองเอฟไออาร์มีความยาวมากขึ้น ก็จะสามารถกำจัดผีที่มีความหน่วงเวลามากขึ้นได้ ซึ่งเอฟพีจีเอเบอร์ XCV300E-6 นี้ สามารถสร้างวงจรกรองเอฟไออาร์ให้มีความยาวได้สูงสุดถึง 256 แท็ป คิดเป็นความหน่วงเวลา 21.33 μs และยังคงสามารถทำงานที่อัตราสุ่มข้อมูลได้มากกว่า 12 ล้านตัวอย่างต่อวินาที ดังตารางที่ 4.2

4.3.2 วงจรแปลงข้อมูล 17 บิตแบบมีเครื่องหมายเป็น 8 บิตแบบไม่มีเครื่องหมาย (17-bit signed to 8-bit unsigned converter)

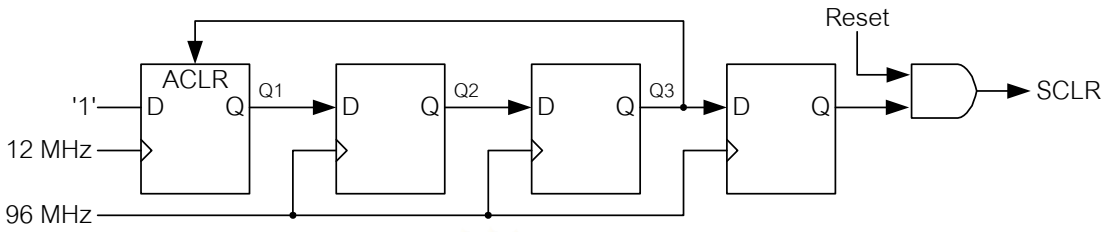
เนื่องจากข้อมูลสัญญาณภาพต้องมีขนาดเป็น 8 บิตแบบไม่มีเครื่องหมาย ก่อนส่งให้กับวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก ดังนั้นวงจรแปลงข้อมูล 17 บิตแบบมีเครื่องหมายเป็น 8 บิตแบบไม่มีเครื่องหมาย ทำหน้าที่ตัด 7 บิตล่างของผลลัพธ์ที่ออกจากวงจรกรองเอฟไออาร์ ซึ่งเป็นข้อมูล 17 บิตให้เหลือเพียง 10 บิตบน จากนั้นจะพิจารณาปัดให้เหลือ 8 บิต โดยหากค่าน้อยกว่า 0 จะให้ผลลัพธ์เป็น 0, หากค่าอยู่ระหว่าง 0 ถึง 255 จะให้ผลลัพธ์มีค่าตามนั้น และหากค่ามากกว่า 255 จะให้ผลลัพธ์มีค่าเป็น 255

4.3.3 วงจรเข้าจังหวะข้อมูล (Data Synchronizer)

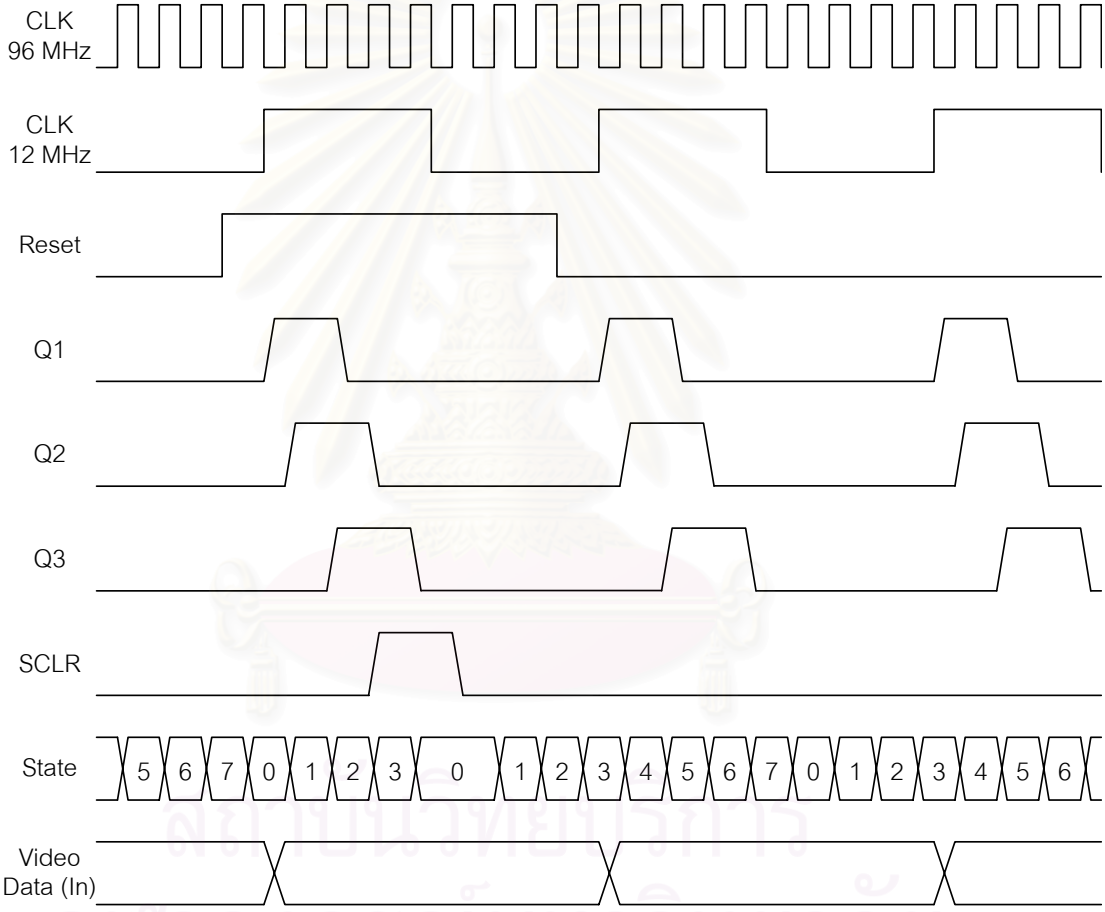
เนื่องจากวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลทำงานตามสัญญาณนาฬิกาความถี่ 12 MHz ทำให้ช่วงหลังจากขอบขาขึ้นของสัญญาณนาฬิกา 12 MHz นี้ ข้อมูลใหม่จะถูกส่งออกมาทำให้ค่าในสายสัญญาณไม่มีเสถียรภาพอยู่ช่วงเวลาหนึ่ง ดังนั้นหากค่าสถานะ (State) ที่ควบคุมการเก็บข้อมูลของวงจรกรองเอฟไออาร์ที่สถานะที่ 7 ตรงกันกับช่วงที่ข้อมูลสัญญาณภาพไม่มีเสถียรภาพนี้ ก็จะทำให้วงจรกรองได้รับข้อมูลที่ผิดพลาดไป ดังนั้นจึงต้องมีวงจรเข้าจังหวะข้อมูลทำหน้าที่สร้างสัญญาณ SCLR ไปรีเซทค่าสถานะให้กลับมาเป็น 0 ในช่วงที่ข้อมูลสัญญาณภาพมีเสถียรภาพมากที่สุด คือช่วงขอบขาลงของสัญญาณนาฬิกาความถี่ 12 MHz โดยวงจรเข้าจังหวะข้อมูลนี้จะส่งสัญญาณ SCLR เป็น ‘1’ ออกไปเมื่อตัวประมวลผลสัญญาณดิจิตอลส่งสัญญาณรีเซทมาด้วยเท่านั้น วงจรเข้าจังหวะข้อมูลมีลักษณะเป็นดังรูปที่ 4.13 และลักษณะของสัญญาณต่างๆ ที่สำคัญแสดงไว้ดังรูปที่ 4.14 สามารถอธิบายการทำงานได้ดังนี้

เมื่อเกิดขอบขาขึ้นของสัญญาณนาฬิกาความถี่ 12 MHz สัญญาณ Q1 จะมีค่าเป็น ‘1’ และเมื่อเกิดขอบขาขึ้นของสัญญาณนาฬิกาความถี่ 96 MHz สัญญาณ Q2 และ Q3 จะมีค่าเป็น ‘1’ ต่อเนื่องกัน ซึ่งเมื่อสัญญาณ Q3 ขึ้นเป็น ‘1’ ส่งผลให้สัญญาณ Q1 ถูกเคลียร์ค่ากลับเป็น ‘0’ อีกครั้ง และในคาบของสัญญาณนาฬิกาความถี่ 96 MHz ถัดมา สัญญาณ SCLR ก็จะขึ้นเป็น ‘1’ (ถ้า

สัญญาณรีเซทมีค่าเป็น ‘1’ ด้วย) ทำให้ค่าสถานะถูกรีเซทกลับเป็น 0 ซึ่งจะตรงกับช่วงขอบขาลงของสัญญาณนาฬิกาความถี่ 12 MHz คือช่วงที่ข้อมูลสัญญาณภาพมีเสถียรภาพมากที่สุด



รูปที่ 4.13 วงจรเข้าจังหวะข้อมูล



รูปที่ 4.14 ลักษณะของสัญญาณที่สำคัญของวงจรเข้าจังหวะข้อมูล

4.3.4 วงจรควบคุมจังหวะการทำงาน (Timing Control)

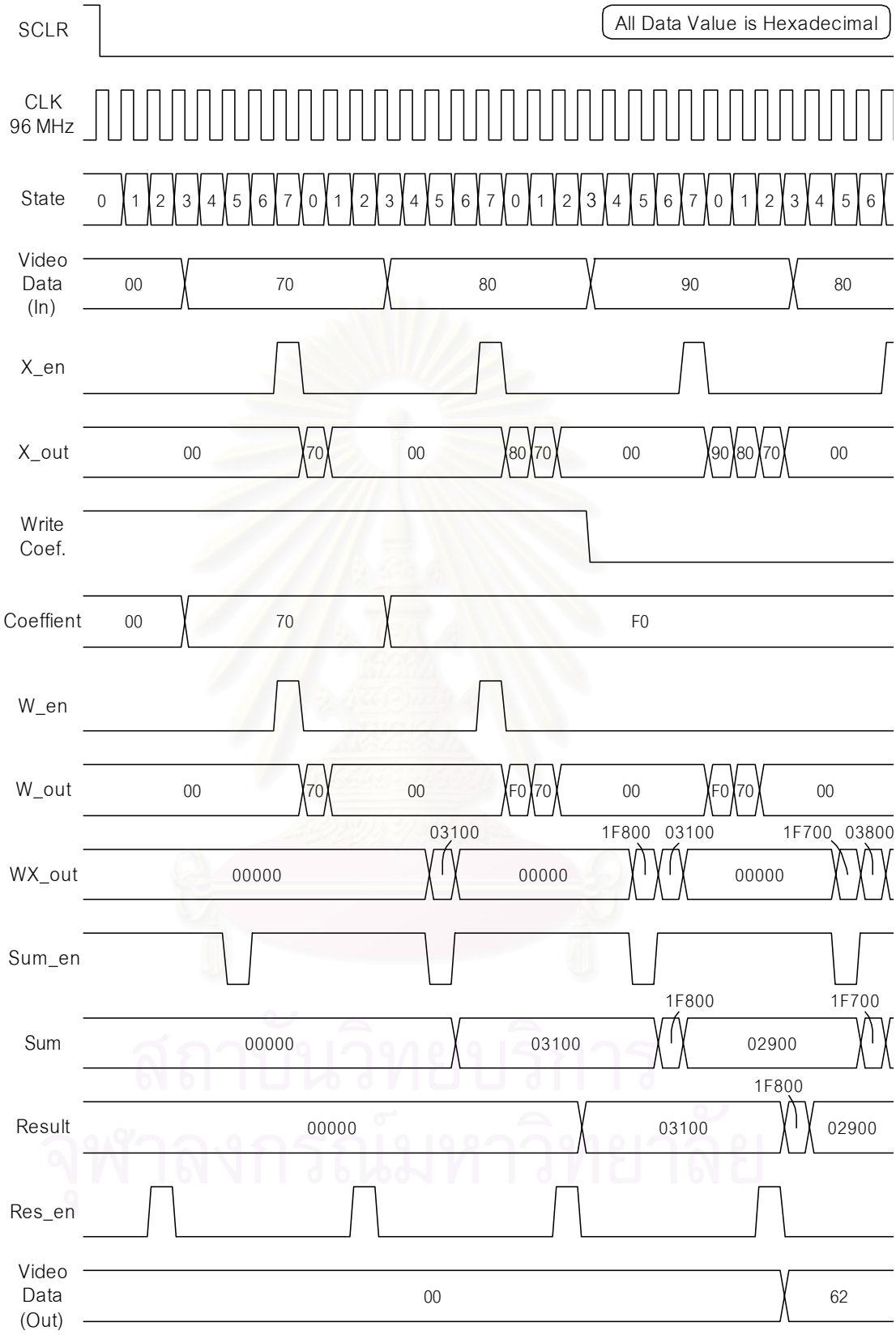
วงจรควบคุมจังหวะการทำงาน ทำหน้าที่สร้างสัญญาณควบคุมต่างๆ อันได้แก่ สัญญาณควบคุมสถานะ (State), สัญญาณควบคุมการเก็บข้อมูลภาพ (X_en), สัญญาณควบคุมการเก็บค่าสัมประสิทธิ์ (W_en), สัญญาณควบคุมการเริ่มหาผลรวม (Sum_en) และสัญญาณควบคุมการ

เก็บผลลัพธ์ (Res_en) เพื่อควบคุมการทำงานภายในวงจรกรองเอฟไอาร์ โดยมีลักษณะของสัญญาณต่างๆ ที่สำคัญดังรูปที่ 4.15 และสามารถอธิบายการทำงานได้ดังนี้

1. *สัญญาณควบคุมสถานะ (State)* เป็นสัญญาณควบคุมขนาด 3 บิต ใช้ในการเลือกข้อมูลทั้ง 8 ชุดที่เก็บอยู่ใน SRL8x8E ซึ่งอยู่ภายในวงจรกรองเอฟไอาร์ให้ออกมาทำการคำนวณทีละชุด
2. *สัญญาณควบคุมการเก็บข้อมูลภาพ (X_en)* เป็นสัญญาณที่ควบคุมให้วงจร SRL8x8E ทำการเลื่อนข้อมูลภาพไป 1 ขั้น พร้อมกับเก็บข้อมูลภาพใหม่เข้ามาในสถานะที่ 7 ของทุกรอบการทำงาน เมื่อสัญญาณควบคุมการเก็บข้อมูลภาพ มีค่าเป็น ‘1’
3. *สัญญาณควบคุมการเก็บค่าสัมประสิทธิ์ (W_en)* เป็นสัญญาณที่ควบคุมให้วงจร SRL8x8E ทำการเลื่อนค่าสัมประสิทธิ์ไป 1 ขั้น พร้อมกับเก็บค่าสัมประสิทธิ์ใหม่เข้ามาในสถานะที่ 7 ของรอบการทำงานที่สัญญาณเขียนค่าสัมประสิทธิ์ (Write Coef.) มีค่าเป็น ‘1’
4. *สัญญาณควบคุมการเริ่มหาผลรวม (Sum_en)* เป็นสัญญาณที่ควบคุมให้ผ่านค่าผลลัพธ์ของการคูณออกไปเมื่อมีค่าเป็น ‘0’ และให้ผลรวมระหว่างผลลัพธ์ของการคูณกับผลรวมครั้งก่อนหน้าผ่านออกไปเมื่อมีค่าเป็น ‘1’ ซึ่งสัญญาณควบคุมนี้จะมีค่าเป็น ‘0’ ที่สถานะที่ 5 เพียงสถานะเดียวเท่านั้น เนื่องจากวงจรคูณมีเวลาแฝง (Latency time) 5 คาบสัญญาณนาฬิกา ดังนั้นผลลัพธ์ของการคูณข้อมูลชุดที่ 0 จึงปรากฏที่สถานะที่ 5 และผลรวมของข้อมูลทั้ง 8 ชุด ก็จะปรากฏที่สถานะที่ 5 นี้เช่นกัน
5. *สัญญาณควบคุมการเก็บผลลัพธ์ (Res_en)* เป็นสัญญาณควบคุมให้เก็บค่าผลลัพธ์ของวงจรกรองเอฟไอาร์หลังจากยุบให้เหลือ 8 บิตแล้ว ไว้ในรีจิสเตอร์ขาออก ซึ่งสัญญาณควบคุมนี้จะมีค่าเป็น ‘1’ ที่สถานะที่ 2 เพียงสถานะเดียวเท่านั้น เนื่องจากวงจรกรองเอฟ-ไอาร์ที่เราสร้างมีความยาว 256 แท็ป คือต้องใช้วงจรกรองความยาว 8 แท็ป ทั้งหมด 32 ชุด ซึ่งทำให้วงจรบวกที่ใช้ในการหาผลรวมของวงจรกรองทั้ง 32 ชุด มีเวลาแฝง (Latency time) ถึง 5 คาบสัญญาณนาฬิกา ดังนั้นผลรวมของวงจรกรองทั้ง 32 ชุด จึงปรากฏที่สถานะที่ 2

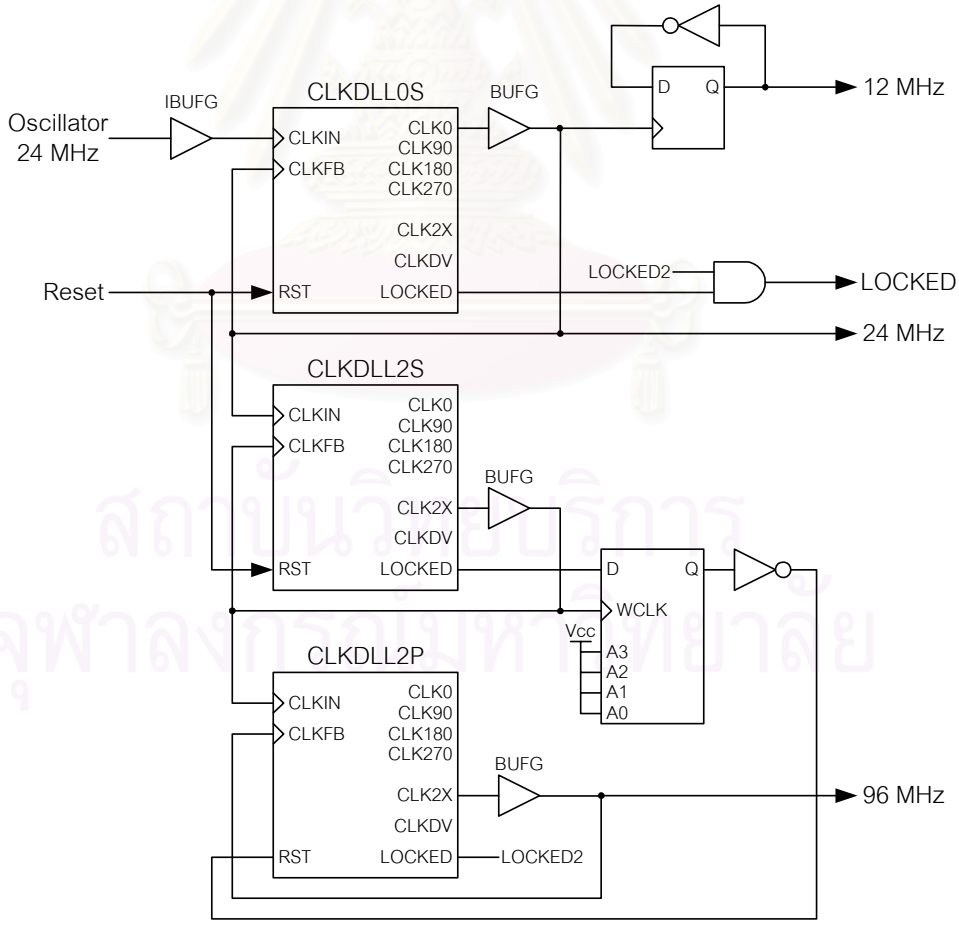
4.4 วงจรสร้างสัญญาณนาฬิกาของระบบ

วงจรสร้างสัญญาณนาฬิกาของระบบ (System Clock Generator) เป็นวงจรที่จะสร้างสัญญาณนาฬิกา ที่มีความถี่แตกต่างกัน 3 ความถี่ ให้กับวงจรต่างๆ ที่อยู่ภายในเอฟพีจีเอคือ



รูปที่ 4.15 สัญญาณต่างๆ ที่สำคัญของวงจรควบคุมจังหวะการทำงาน

1. *ความถี่ 12 MHz* เป็นความถี่ที่ใช้ป้อนให้กับวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล และวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก เพื่อใช้ในการสุ่มข้อมูลสัญญาณภาพรวม
2. *ความถี่ 96 MHz* เป็นความถี่ที่ใช้ป้อนให้กับวงจรกรองเอฟไออาร์ เนื่องจากการใช้เทคนิคการใช้ทรัพยากรร่วม 8 ครั้ง ทำให้วงจรกรองเอฟไออาร์ต้องทำงานที่ความถี่สูงกว่าอัตราสุ่มข้อมูลสัญญาณภาพถึง 8 เท่า นั่นคือวงจรกรองเอฟไออาร์ต้องทำงานที่ความถี่ 96 MHz
3. *ความถี่ 24 MHz* เป็นความถี่หลักของวงจรส่วนต่างๆ นอกเหนือจากวงจรที่ใช้ความถี่ทั้งสองข้างต้น และยังใช้เป็นความถี่หลักในการให้กำเนิดสัญญาณนาฬิกาทั้งสองความถี่ดังกล่าวข้างต้นด้วย โดยอาศัยวงจร CLKDLL ภายในเอฟพีจีเอ นำมาต่อกันดังรูปที่ 4.16 ซึ่งนอกจากวงจร CLKDLL จะสามารถให้กำเนิดความถี่ของสัญญาณนาฬิกาเป็น 4 เท่าได้แล้ว วงจร CLKDLL ยังช่วยลดปัญหาการเลื่อนไปของสัญญาณนาฬิกา (Clock Skew) อันเนื่องมาจากระยะห่างระหว่างวงจรภายในเอฟพีจีเอที่อยู่ห่างกันมากๆ ได้



รูปที่ 4.16 วงจรสร้างสัญญาณนาฬิกาของระบบ

เนื่องจากภายในเอฟพีจีเอเบอร์ XCV300E-6 มีวงจร CLKDLL และตัว BUFG อยู่เพียง 8 และ 4 ตัวตามลำดับ และในการสร้างวงจรความถี่ 4 เท่า (96 MHz) จำเป็นต้องใช้ CLKDLL ที่เป็นคู่ S และ P เดียวกัน ทำให้ในการสร้างวงจรจะต้องกำหนดตำแหน่งของวงจร CLKDLL และตัว BUFG เอง โดยอ้างอิงชื่อของอุปกรณ์ภายในวงจรที่เราออกแบบ กับตำแหน่งภายในเอฟพีจีเอ ในแฟ้มเงื่อนไขบังคับ (Constrain File) ซึ่งมีนามสกุลเป็น ".ucf" ดังในภาคผนวก ข

4.5 วงจรแปลงการอ้างอิงหน่วยความจำเป็นสัญญาณควบคุม

การอ้างอิงหน่วยความจำของตัวประมวลผลสัญญาณดิจิตอลในช่วง 0xA0000000 ถึง 0xA03FFFFF เป็นช่วงหนึ่งที่ตัวประมวลผลสัญญาณดิจิตอลใช้สำหรับติดต่อกับอุปกรณ์ภายนอก ซึ่งใช้ต่ออยู่กับเอฟพีจีเอ วงจรแปลงการอ้างอิงหน่วยความจำเป็นสัญญาณควบคุมจะทำหน้าที่แปลงสัญญาณการอ่าน และเขียนข้อมูลจากตัวประมวลผลสัญญาณดิจิตอล ให้เป็นสัญญาณควบคุมวงจรส่วนต่างๆ ภายในเอฟพีจีเอดังตารางที่ 4.3

ตารางที่ 4.3 สัญญาณควบคุมที่สัมพันธ์กับการอ้างอิงหน่วยความจำตำแหน่งต่างๆ

ตำแหน่งที่อ้างอิง	สัญญาณควบคุมเมื่ออ่าน	สัญญาณควบคุมเมื่อเขียน
0xA0000000	bit 0 : LOCKED	bit 0 : Reset
0xA0000004	bit 0-9 : Line	bit 0 : Bypass
0xA0000010	bit 0-31 : GCR Data	-
0xA0000014	bit 0 : GCR Ready	bit 0 : GCR Enable
0xA0000020	-	bit 0-7 : Coefficient

ดังนั้นตัวประมวลผลสัญญาณดิจิตอลสามารถควบคุมการทำงานของเอฟพีจีเอได้ ด้วยการอ่าน และเขียนค่าไปยังหน่วยความจำตำแหน่งต่างๆ เช่น การโปรแกรมค่าสัมประสิทธิ์ให้กับวงจรกรองเอฟไออาร์ ทำโดยการเขียนค่าสัมประสิทธิ์ไปยังตำแหน่ง 0xA0000020 หรือการสั่งให้ผ่านสัญญาณภาพขาเข้าไปที่ขาออกทำโดยการเขียนค่า 0x01 ไปยังตำแหน่ง 0xA0000004 เป็นต้น

4.6 สรุปท้ายบท

ในบทนี้ ได้กล่าวถึงรายละเอียดของวงจรที่ออกแบบให้ทำงานบนเอฟพีจีเอ อันได้แก่ วงจรแยกข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี, วงจรกรองเอฟไอาร์, วงจรสร้างสัญญาณนาฬิกาของระบบ, วงจรแปลงการอ้างอิงหน่วยความจำเป็นสัญญาณควบคุม และวงจรมัลติเพลกซ์ เมื่อทำการสังเคราะห์ และสร้างวงจรเพื่อโปรแกรมลงเอฟพีจีเอเบอร์ XCV300E-6 ของบริษัท Xilinx ซึ่งมีทรัพยากรภายในทั้งหมดประมาณ 300,000 เกต ซึ่งถือว่าค่อนข้างมาก แต่อย่างไรก็ดีทรัพยากรเกือบทั้งหมดจะต้องถูกใช้ไปในการสร้างวงจรกรองเอฟไอาร์ขนาดใหญ่ เพื่อให้เครื่องกำจัดผีสามารถกำจัดผีที่มีเวลาประวิงสูงที่สุด ดังนั้นทรัพยากรที่ใช้ไปทั้งหมดภายในชิพเอฟพีจีเอเป็นดังตารางที่ 4.4

ตารางที่ 4.4 รายงานการใช้ทรัพยากรของเอฟพีจีเอ

	No. Used	Max Available	% Used
Slices	3,070	3,072	99%
Flip Flops	5,547	6,144	90%
LUTs	4,576	6,144	74%
bounded IOBs	108	158	68%
Block RAMs	2	32	6%
GCLKs	3	4	75%
GCLKIOBs	1	4	25%
DLLs	3	8	37%

จะสังเกตว่าทรัพยากรภายในเอฟพีจีเอถูกใช้ไปเกือบทั้งหมดคือ ใช้ไป 3,070 Slices จากทั้งหมด 3,072 Slices แต่อย่างไรก็ดี จำนวนที่รายงานออกมานี้ เป็นจำนวนที่รวมวงจรที่ไม่เกี่ยวข้องกับวงจรที่ออกแบบอยู่ด้วย คือวงจรบางส่วนได้ถูกนำไปใช้ช่วยในการเชื่อมต่อวงจร ดังนั้นในการแก้ไขวงจร เพิ่มหรือลดวงจรภายในเพียงเล็กน้อย จึงไม่เกิดปัญหาทรัพยากรไม่เพียงพอ สำหรับวงจรในเอฟพีจีเอสามารถดูรายละเอียดได้ในภาคผนวก ค

เนื่องจากทรัพยากรเกือบทั้งหมดถูกใช้ไป ทำให้วงจรที่ได้มีความเร็วในการทำงานที่ต่ำลง ดังนั้นเพื่อให้วงจรสามารถทำงานได้ที่ความถี่สัญญาณนาฬิกาที่ต้องการ จึงต้องมีการกำหนดค่าเวลาประวิงมากที่สุด ที่วงจรยังสามารถทำงานได้ไว้ในแฟ้มเงื่อนไขบังคับ โดยได้กำหนดเผื่อไว้จากความถี่ที่ต้องการเล็กน้อยคือ ความถี่ 96 MHz ได้เผื่อเป็น 100 MHz (เวลาประวิง 10 ns), ความถี่ 24 MHz ได้เผื่อเป็น 25 MHz (เวลาประวิง 40 ns) และความถี่ 12 MHz ได้เผื่อเป็น 12.5 MHz (เวลาประวิง 80 ns) ซึ่งผลหลังจากการสร้างวงจรพบว่า เวลาของเส้นทางที่มากที่สุดรายงานออกมายังคงน้อยกว่าค่าที่กำหนดดังตารางที่ 4.5

ตารางที่ 4.5 รายงานเวลาประวิงของเส้นทางที่ยาวที่สุดของเอฟพีจีเอ

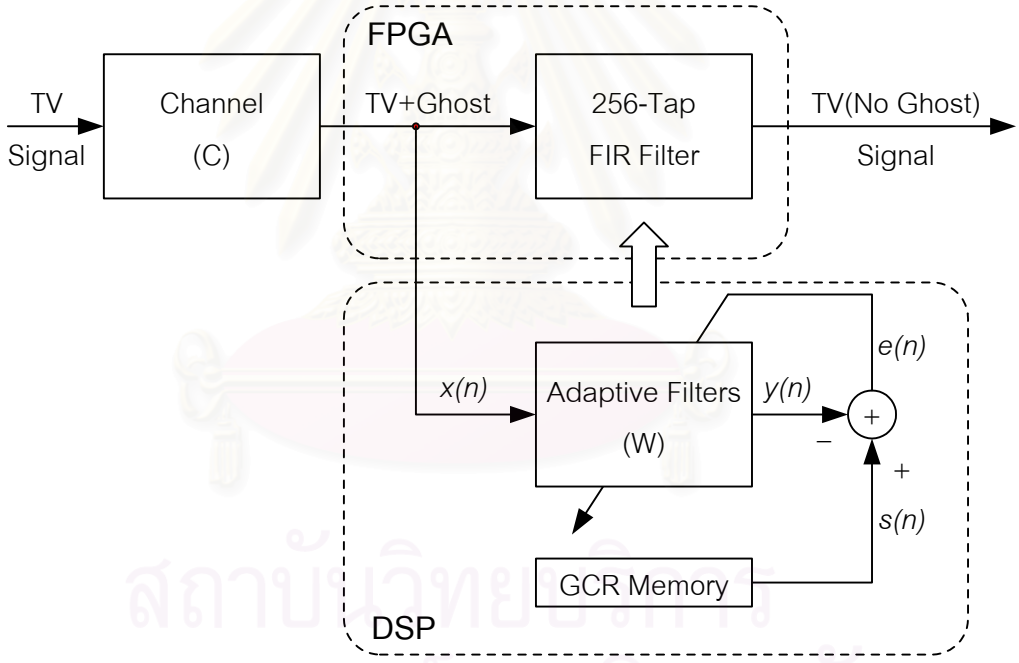
ความถี่ที่ต้องการ	เวลาประวิงที่กำหนด	เวลาประวิงของเส้นทางที่ยาวที่สุด
12 MHz	80 ns	12.750 ns
24 MHz	40 ns	18.362 ns
96 MHz	10 ns	9.820 ns

บทที่ 5

โปรแกรมกำจัดผี

5.1 กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด (LMS : Least Mean Square Algorithm)

เนื่องจากเครื่องกำจัดผีต้องทำงานแบบเวลาจริง (Real Time) และเครื่องกำจัดผีที่ดีควรจะใช้เวลาในการกำจัดผีน้อย ดังนั้นกระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุดจึงเป็นกระบวนวิธีที่เหมาะสมมาก เพราะเป็นกระบวนวิธีที่ง่าย มีการคำนวณไม่ซับซ้อน และมีเวลาการลู่เข้าที่เร็ว [2] สำหรับขั้นตอนในการคำนวณหาค่าสัมประสิทธิ์ที่ใช้กำจัดผีแสดงได้ดังรูปที่ 5.1 อธิบายการทำงานได้ดังนี้



รูปที่ 5.1 ขั้นตอนการคำนวณหาค่าสัมประสิทธิ์ที่ใช้กำจัดผี

สัญญาณโทรทัศน์ที่แพร่ภาพสัญญาณออกมาทุกทิศทาง ผ่านช่องสัญญาณ (Channel) ที่ทำให้เกิดสัญญาณผีปะปนเข้ามา ตัวประมวลผลสัญญาณดิจิตอลจะทำการอ่านข้อมูลสัญญาณอ้างอิงที่มาพร้อมกับสัญญาณภาพ เข้ามาคำนวณผ่านวงจรกรองแบบปรับตัวซึ่งมีค่าสัมประสิทธิ์ **W(n)** ที่คาดว่าจะสามารถกำจัดผีได้ดังสมการที่ 5.1 จากนั้นนำผลลัพธ์ $y(n)$ ที่ได้ไปเปรียบเทียบกับค่าของสัญญาณอ้างอิงที่เก็บไว้ในหน่วยความจำ $s(n)$ เพื่อหาค่าผิดพลาด $e(n)$

ดังสมการที่ 5.2 แล้วนำไปใช้ในการปรับค่าสัมประสิทธิ์ในตัววงจรกรองแบบปรับตัวให้มีค่าผิดพลาดที่น้อยลง ดังสมการที่ 5.3 และเมื่อค่าผิดพลาดน้อยจนถึงระดับที่คิดว่าสามารถกำจัดผีได้แล้วก็ทำการโปรแกรมค่าสัมประสิทธิ์ที่ได้ให้กับวงจรกรองภายในเอฟพีจีเอต่อไป

$$y(n) = \mathbf{W^T(n)} * \mathbf{X(n)} \text{ หรือ } y(n) = \sum_{i=0}^{N-1} w_i(n) \cdot x(n-i) \quad \text{.....สมการที่ 5.1}$$

$$e(n) = s(n) - y(n) \quad \text{.....สมการที่ 5.2}$$

$$\mathbf{W(n+1)} = \mathbf{W(n)} + \mu \cdot e(n) \cdot \mathbf{X(n)} \quad \text{.....สมการที่ 5.3}$$

เมื่อ $\mathbf{X(n)}$, $\mathbf{W(n)}$, $\mathbf{W(n+1)}$ เป็นเวกเตอร์สัญญาณภาพ และเวกเตอร์สัมประสิทธิ์ที่ n และ n+1 ตามลำดับ

$x(n-i)$ เป็นข้อมูลสัญญาณภาพตัวที่ i ในอดีต

$w_i(n)$ เป็นค่าสัมประสิทธิ์ตัวที่ i

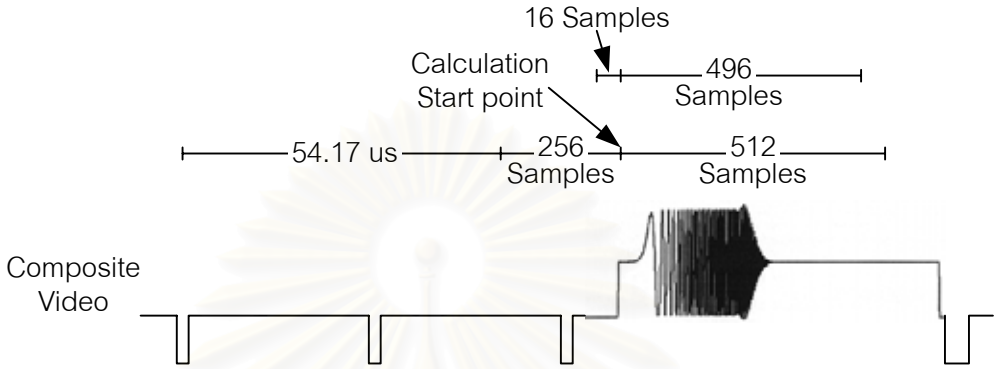
จากการทดสอบปรับค่าสัมประสิทธิ์ (μ) เพื่อหาค่าที่เหมาะสมในการกำจัดผี พบว่าค่า 0.125 เป็นค่าที่มีความเหมาะสมมาก กล่าวคือค่าสัมประสิทธิ์ที่มากกว่า 0.125 ไม่สามารถกำจัดผีได้หมดเนื่องจากการปรับค่าสัมประสิทธิ์ของวงจรกรองเปลี่ยนแปลงเร็วเกินไป และถ้าใช้ค่าสัมประสิทธิ์ที่น้อยกว่า 0.125 นี้ เวลาลู่เข้าของกระบวนการกำจัดผีจะนาน

5.2 ข้อมูลสัญญาณอ้างอิงสำหรับกำจัดผี

ในการหาค่าของข้อมูลสัญญาณอ้างอิงที่จะนำมาใช้ในการกำจัดผีนั้น ทำโดยใช้เครื่องสร้างและแทรกรูปแบบสัญญาณโทรทัศน์รุ่น PM 5655 (PM 5655 VITS Generator & Inserter) ป้อนสัญญาณวีดิทัศน์ที่มีสัญญาณอ้างอิงแทรกอยู่เส้นภาพที่ 318 ให้กับเครื่องกำจัดผีที่พัฒนาขึ้นโดยตรง จากนั้นพัฒนาโปรแกรมในส่วนของตัวประมวลผลสัญญาณดิจิตอล ให้ทำการอ่านค่าสัญญาณอ้างอิงเข้ามา แล้วหาค่าเฉลี่ยของสัญญาณดังกล่าว จากนั้นทำการโลดค่าเฉลี่ยของสัญญาณอ้างอิงที่ได้ขึ้นมาเก็บไว้บนคอมพิวเตอร์ส่วนบุคคล (Personal Computer) เพื่อใช้เป็นสัญญาณอ้างอิงมาตรฐานให้กับโปรแกรมกำจัดผีที่จะพัฒนาขึ้น

โดยปกติสัญญาณที่มีความเข้มมากที่สุดจะใช้เวลาเดินทางมาถึงเครื่องรับน้อยที่สุด ซึ่งทำให้เกิดผีปรากฏบนจอภาพทางขวาของภาพหลัก แต่ในบางกรณีสัญญาณที่ใช้เวลาเดินทางน้อยที่สุดก็อาจมีความเข้มของสัญญาณน้อยกว่าสัญญาณที่มาทีหลังได้ ซึ่งทำให้เกิดผีปรากฏบนจอภาพทางซ้ายของภาพหลัก เพื่อให้เครื่องกำจัดผีสามารถกำจัดผีที่เกิดขึ้นทางซ้ายของภาพหลักนี้ได้ด้วย ในการคำนวณจึงต้องใช้ข้อมูลสัญญาณอ้างอิงที่เกิดขึ้นก่อนจุดที่เราคำนวณมาใช้ด้วย

และเนื่องจากตัวแยกข้อมูลสัญญาณอ้างอิงในเอฟพีจีเอ ถูกออกแบบให้เก็บข้อมูลสัญญาณอ้างอิงทั้งหมด 768 ไบท์ โดยการคำนวณเพื่อปรับค่าสัมประสิทธิ์ของวงจรกรองจะเริ่มที่ข้อมูลสัญญาณอ้างอิงไบท์ที่ 257 ดังนั้นจึงใช้ข้อมูลสัญญาณอ้างอิงตั้งแต่ไบท์ที่ 241 ถึงไบท์ที่ 752 เป็นข้อมูลสัญญาณอ้างอิงตั้งต้นเก็บในหน่วยความจำ เพื่อใช้ในการคำนวณ ดังรูปที่ 5.2



รูปที่ 5.2 ลักษณะของสัญญาณที่เกี่ยวข้องกับข้อมูลสัญญาณอ้างอิง

เนื่องจากเราใช้ข้อมูลสัญญาณอ้างอิงก่อนจุดที่เริ่มคำนวณ 16 ตัวอย่าง ทำให้ในการคำนวณเสมือนเห็นข้อมูลสัญญาณอ้างอิงที่อ่านจากตัวแยกสัญญาณอ้างอิงในเอฟพีจีเอ เกิดเร็วขึ้น 16 ตัวอย่าง ดังนั้นค่าสัมประสิทธิ์ของวงจรกรองเริ่มต้นจึงควรมีค่าของสมาชิกตัวที่ 16 เป็น 127 (ค่าสูงสุดของข้อมูล 8 บิตแบบมีเครื่องหมาย) และสมาชิกตัวอื่นๆ เป็น 0 นั่นคือวงจรกรอง 16 แท็ปแรกจะใช้ในการกำจัดผีที่เกิดขึ้นก่อนสัญญาณภาพหลัก และวงจรกรอง 240 แท็ปหลังจะใช้กำจัดผีที่เกิดขึ้นหลังจากสัญญาณภาพหลัก

5.3 การทำงานของโปรแกรมกำจัดผี

จะขอบรรยายแนวความคิดในการออกแบบโปรแกรมด้วยภาษาบรรยายโปรแกรม (PDL : Program Description Language) โดยจะเริ่มบรรยายจากแนวความคิดกว้างๆ และค่อยๆ เพิ่มรายละเอียดลงไปทีละระดับ จนถึงระดับที่มีรายละเอียดพอที่จะนำไปพัฒนาเป็นโปรแกรมได้ ดังนี้

5.3.1 ภาษาบรรยายโปรแกรมระดับ 1 (PDL Level 1)

```
MODULE : Main
        System Initialization
        REPEAT
                Get GCR Data
                Calculate Coefficient for 256-Tap FIR Filter
```

Program Coefficient to 256-Tap FIR Filter in FPGA

UNTIL Forever

END Main

แนวความคิดหลักของโปรแกรมก็คือ โปรแกรมจะทำงานเป็น 2 ช่วง ในช่วงแรกจะทำการกำหนดค่าเริ่มต้นขอบระบบ (System Initialization) หลังจากนั้นจะเป็นช่วงการทำงานที่จะวนรอบคอยอ่านข้อมูลสัญญาณอ้างอิงเข้ามา แล้วนำไปใช้ในการคำนวณหาค่าสัมประสิทธิ์ที่เหมาะสม จากนั้นก็โปรแกรมค่าสัมประสิทธิ์ที่ได้ให้กับวงจรกรองเอฟไอาร์ที่อยู่ภายในเอฟพีจีเอ

5.3.2 ภาษาบรรยายโปรแกรมระดับ 2 (PDL Level 2)

MODULE : Main

System Initialization :

Setup EMIF CE Space Control Register

Setup Xilinx part

Setup Parameters for calculation

REPEAT

Get GCR Data :

Enable GCR

Wait until GCR Data Ready is true

Disable GCR

Read GCR Data 256 Bytes for Initial Xi

Read GCR Data 512 Bytes for Calculation

Calculate Coefficient of 256-Tap FIR Filter :

IF (GCR Data exist) THEN

Check GCR Data Style

Adjust Coefficient by LMS Algorithm

END IF

Program Coefficient to 256-Tap FIR Filter in FPGA :

Set Bypass is true

Write Coefficient to 256-Tap FIR Filter

Set Bypass is false

UNTIL Forever

END Main

ภาษาบรรยายโปรแกรมระดับ 2 (PDL Level 2) นี้ กล่าวถึงรายละเอียดในการทำงานแต่ละส่วนมากขึ้นคือ ในส่วนการกำหนดค่าเริ่มต้นจะต้องกำหนดค่าโมดการติดต่อกับหน่วยความจำภายนอกตัวประมวลผลสัญญาณดิจิตอล (EMIF) จากนั้นทำการกำหนดค่าสถานะต่างๆ ภายในเอฟพีจีเอ และกำหนดค่าเริ่มต้นของตัวแปรต่างๆ ที่ใช้ในการคำนวณภายในตัวประมวลผลสัญญาณดิจิตอล ในส่วนการอ่านข้อมูลสัญญาณอ้างอิงจะต้องสั่งให้ตัวแยกข้อมูลสัญญาณอ้างอิงทำการเก็บข้อมูลสัญญาณอ้างอิงลงหน่วยความจำ เมื่อมีข้อมูลในหน่วยความจำแล้วก็ทำการอ่านข้อมูลออกมา โดยแบ่งเป็น 2 ส่วน ส่วนแรก 256 ไบท์ใช้กำหนดค่าเริ่มต้นของเวกเตอร์สัญญาณภาพ (Xi) ส่วนที่สอง 512 ไบท์ใช้สำหรับการคำนวณ ในส่วนของการคำนวณจะมีการตรวจสอบสัญญาณข้อมูลอ้างอิงก่อนว่าได้รับมาถูกต้องหรือไม่ จากนั้นจึงตรวจสอบรูปแบบข้อมูลสัญญาณอ้างอิงว่าเป็นแบบบวกหรือแบบลบ แล้วจึงทำการคำนวณหาค่าสัมประสิทธิ์ และส่วนสุดท้ายคือการโปรแกรมค่าสัมประสิทธิ์ที่ได้ให้กับวงจรกรองเอฟไอาร์ จะต้องสั่งให้เอฟพีจีเอปล่อยสัญญาณขาเข้าผ่านออกไปก่อนเพื่อป้องกันสัญญาณภาพที่ผิดปกติในช่วงการโปรแกรม จากนั้นจึงโปรแกรมค่าสัมประสิทธิ์ลงไป แล้วจึงยกเลิกการปล่อยผ่าน

จะเห็นว่ารายละเอียดในภาษาบรรยายโปรแกรมระดับ 2 นี้ยังไม่เพียงพอที่จะนำไปเขียนโปรแกรมได้ เนื่องจากยังขาดรายละเอียดของค่าต่างๆ ที่ใช้ และรายละเอียดในส่วนของการคำนวณที่ซับซ้อน ซึ่งภาษาบรรยายโปรแกรมระดับ 3 จะมีรายละเอียดส่วนต่างๆ เหล่านี้เพิ่มเข้ามา

5.3.3 <u>ภาษาบรรยายโปรแกรมระดับ 3 (PDL Level 3)</u>

MODULE : Main

System Initialization :

Set EMIF CE Space Control Register :

EMIF_CE2 = 0x23D2CF21 (32-bit-wide asynchronous interface)

Setup Xilinx part :

REPEAT

Reset Xilinx

Check Locked

UNTIL Locked is true

Set Bypass is true

Disable GCR

```
Setup Parameters for calculation :
    Initial Coefficient Wi[0-255] = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 127 0…0]
    Initial Mean Square Error (MSE) = 0
    Adjust Flag (AFlag) is false
REPEAT
  Get GCR Data :
    Enable GCR
    Wait until GCR Data Ready is true
    Disable GCR
    Read GCR Data 256 Bytes for Initial Xi :
      FOR i=0 to 63 LOOP
        Read GCR Data 1 word (4 bytes)
        Write GCR Data each byte to Xi
      END LOOP
    Read GCR Data 512 Bytes for Calculation :
      FOR i=0 to 128 LOOP
        Read GCR Data 1 word (4 bytes)
        Write GCR Data each byte to GData
      END LOOP
  Calculate Coefficient of 256-Tap FIR Filter :
    IF (GCR Data exist) THEN
      Check GCR Data Style :
        IF (plus style) THEN
          Set Index to plus style
        ELSE
          Set Index to minus style
        END IF
      Adjust Coefficient by LMS Algorithm :
        Sum of Square Error (SSE) = 0
        FOR i=0 to 511 LOOP
          Find output of FIR Filter (Yi) by Equation 5.1
```

```
                    Find Error = Ref[Index][i] – Yi by Equation 5.2
                    Find SSE = SSE + (Error * Error)
                    IF (AFlag = true) THEN Adjust Coefficient by Equation 5.3
                    Shift Xi 1 byte and Load Xi[0] = GData[i]
                END LOOP
                Find MSE = SSE / 512
                IF (MSE more than Upper Bound) THEN
                    AFlag = true
                ELSE IF (MSE less than Lower Bound) THEN
                    AFlag = false
                END IF
            END IF
        Program Coefficient to 256-Tap FIR Filter in FPGA :
            Wait until Line No. more than 610
            Set Bypass is true
            Write Coefficient to 256-Tap FIR Filter :
                FOR i=0 to 255 LOOP
                    Program Coefficient = Wi[255-i]
                END LOOP
            Set Bypass is false
    UNTIL Forever
END Main
```

จะเห็นว่าภาษาบรรยายโปรแกรมระดับ 3 (PDL Level 3) นี้ ได้บรรยายรายละเอียดต่างๆ ของการทำงานมากใกล้เคียงกับภาษาคอมพิวเตอร์ระดับสูง จึงสามารถนำไปสร้างเป็นโปรแกรมกำจัดผีด้วยภาษาซี (C Language) ได้โดยง่าย ดังภาคผนวก ง

5.4 สรุปท้ายบท

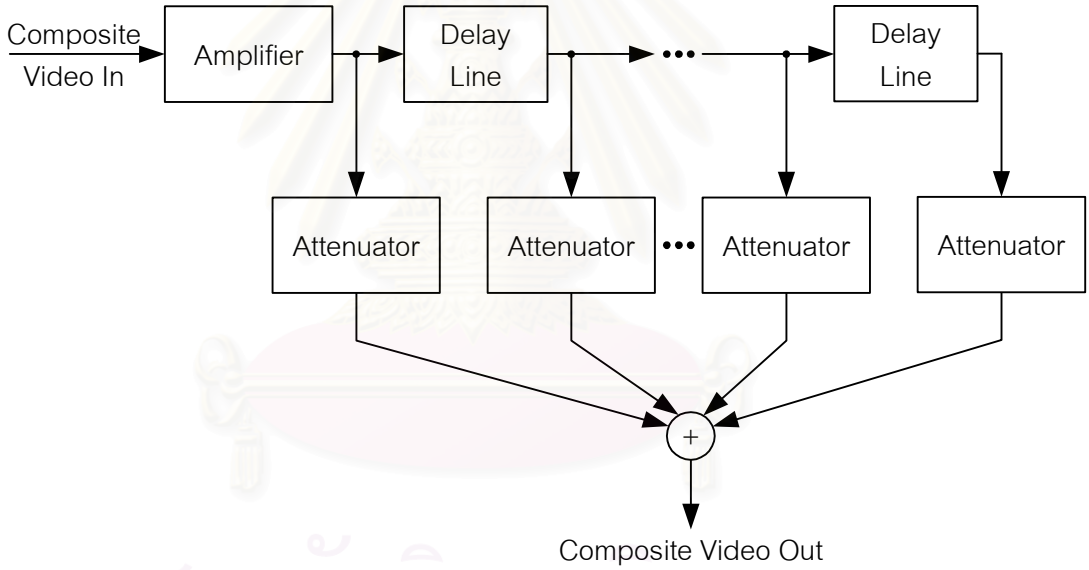
ในบทนี้ได้กล่าวถึงขั้นตอนการประยุกต์ใช้กระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด ในการหาค่าสัมประสิทธิ์ของวงจรกรอง ข้อมูลสัญญาณอ้างอิงที่เก็บในหน่วยความจำเพื่อใช้ในการคำนวณ และสุดท้ายเป็นรายละเอียดของโปรแกรมกำจัดผีแสดงในรูปของภาษาบรรยายโปรแกรม 3 ระดับ

บทที่ 6

การทดสอบ และสรุปผล

6.1 เครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์

การทดสอบการทำงานของเครื่องกำจัดผีในสัญญาณโทรทัศน์ ไม่สามารถทดสอบกับระบบจริงได้ เนื่องจากในระบบจริงไม่สามารถควบคุมปัจจัยต่างๆ อีกทั้งยังต้องอาศัยความช่วยเหลือจากสถานีแพร่ภาพสัญญาณ ให้แทรกสัญญาณอ้างอิงมากับสัญญาณภาพด้วย ซึ่งเป็นไปได้ยาก จึงได้สร้างเครื่องจำลองการเกิดผีอย่างง่ายขึ้นมา เพื่อใช้จำลองผลที่เกิดขึ้นจากช่องสัญญาณ โดยเครื่องจำลองการเกิดผีมีโครงสร้างวงจรเป็นดังรูปที่ 6.1 มีหลักการทำงานดังนี้

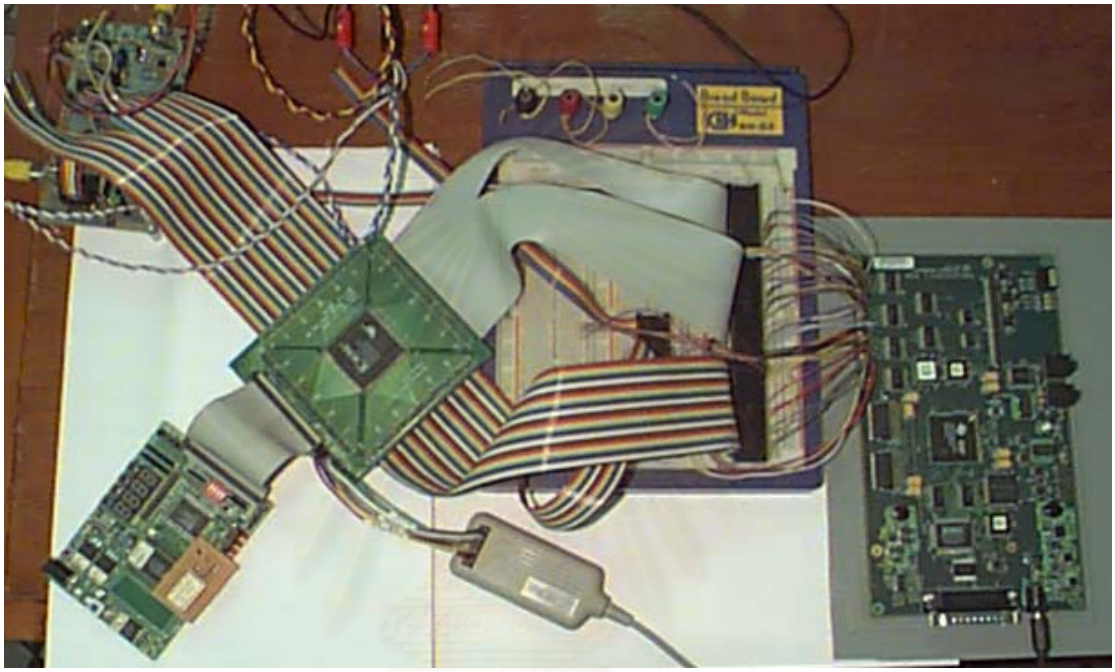


รูปที่ 6.1 โครงสร้างวงจรของเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์

สัญญาณวีดิทัศน์ที่เข้ามาจะถูกป้อนผ่านวงจรขยายสัญญาณ แล้วผ่านตัวแท็ปสายประวิง (Delay Line) เป็นขั้นๆ โดยในแต่ละขั้นจะมีความหน่วง 270 ns จากนั้นสัญญาณที่มีความหน่วงค่าต่างๆ ที่เราต้องการจะถูกเลือกให้ผ่านวงจรลดทอนขนาดของสัญญาณ โดยสามารถปรับให้แต่ละสัญญาณถูกลดทอนขนาดไปไม่เท่ากัน แล้วนำสัญญาณเหล่านี้มาผ่านวงจรรวมสัญญาณ ทำให้ได้สัญญาณวีดิทัศน์ที่มีสัญญาณผีปะปนออกมาด้วย

6.2 การทดสอบการทำงาน

เมื่อออกแบบส่วนต่างๆ ของเครื่องกำจัดผีในสัญญาณโทรทัศน์ และแก้ไขเป็นที่เรียบร้อยแล้ว นำส่วนประกอบต่างๆ อันได้แก่ แผงวงจรแปลงสัญญาณแอนาลอก-ดิจิตอล, ชุดทดลองของตัวประมวลผลสัญญาณดิจิตอล และวงจรเอฟพีจีเอ ดังที่ได้กล่าวไว้ในบทที่ 3 มาประกอบเข้าด้วยกัน เพื่อทำการทดสอบต่อไป ส่วนประกอบต่างๆ ในเครื่องกำจัดผีแสดงไว้ในรูปที่ 6.2

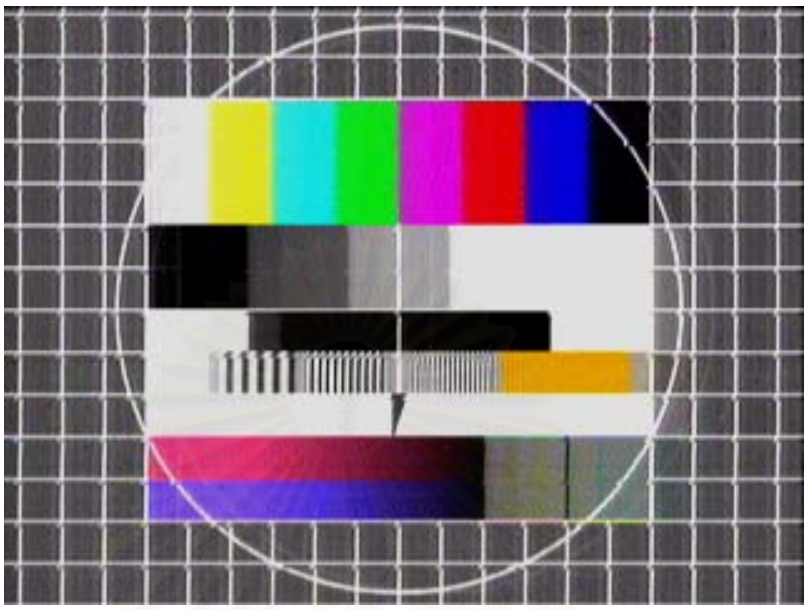


รูปที่ 6.2 ส่วนประกอบต่างๆ ในเครื่องกำจัดผีในสัญญาณโทรทัศน์

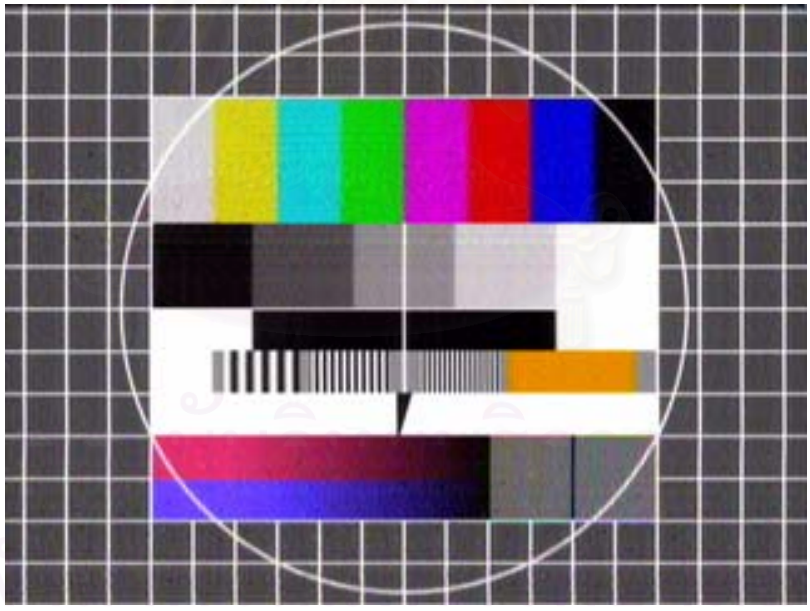
การทดสอบจะทำโดยใช้เครื่องสร้างและแทรกรูปแบบสัญญาณโทรทัศน์รุ่น PM 5655 (PM 5655 VITS Generator & Inserter) ในการสร้างสัญญาณวีดิทัศน์ที่มีสัญญาณอ้างอิงสำหรับกำจัดผีตามมาตรฐาน ITU-R BT.1124 แทรกมาที่เส้น 318 ด้วย ป้อนให้กับเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์ แล้วนำสัญญาณวีดิทัศน์ที่มีผีรวมอยู่ด้วยนี้ ป้อนให้กับเครื่องกำจัดผีที่ได้พัฒนาขึ้น แล้วนำสัญญาณหลังผ่านกระบวนการกำจัดผีป้อนให้กับเครื่องรับโทรทัศน์ต่อไป

ในการทดสอบการทำงานได้ทำการปรับความหน่วง และความเข้มของสัญญาณผีให้มีขนาดต่างๆ กัน โดยรูปที่ 6.3 และรูปที่ 6.4 เป็นตัวอย่างของสัญญาณภาพก่อนและหลังการกำจัดผี ซึ่งกำหนดสัญญาณผีไว้ 2 สัญญาณ มีความหน่วงเป็น 2.16 μs กับ 4.05 μs และมีความเข้มเป็น –12 dB กับ –20 dB ของสัญญาณหลัก จากการสังเกตสัญญาณภาพก่อนการกำจัดผี พบว่าสัญญาณผีที่ปะปนอยู่ทำให้เกิดผลใน 2 ลักษณะปรากฏให้เห็นอย่างชัดเจนบนจอภาพ ลักษณะแรกคือสัญญาณซิงก์มีคุณภาพแย่ลง ส่งผลให้ภาพเกิดการบิดเบี้ยวไป อีกลักษณะหนึ่งคือการเกิด

เงาของภาพซ้อนอยู่ในภาพหลัก ซึ่งจะเห็นว่าหลังจากผ่านการกำจัดผีแล้ว สามารถแก้ไขปัญหาทั้ง 2 ลักษณะที่เกิดขึ้นได้



รูปที่ 6.3 สัญญาณภาพก่อนการกำจัดผี



รูปที่ 6.4 สัญญาณภาพหลังการกำจัดผี

เพื่อหาขีดความสามารถของเครื่องกำจัดผี ได้ทำการทดสอบเครื่องกำจัดผีแบ่งเป็นสามลักษณะตามความหน่วงของสัญญาณผี ดังนี้

1. *สัญญาณผีมีความหน่วงอยู่ในช่วง -1.33 µs ถึง 0 µs* พบว่าสามารถกำจัดผีที่มีความเข้มของสัญญาณไม่เกิน -12 dB เมื่อเทียบกับสัญญาณหลักได้เป็นอย่างดี แต่ถ้าสัญญาณมี

ความเข้มมากกว่า -12 dB พบว่าไม่สามารถกำจัดผีได้หมด กล่าวคือภาพที่ปรากฏบนจอภาพจะมีคุณภาพที่ดีขึ้นแต่ยังคงเห็นผีปรากฏอยู่เล็กน้อย

2. *สัญญาณผีมีความหน่วงอยู่ในช่วง 0 µs ถึง +10 µs* พบว่าสามารถกำจัดผีที่มีความเข้มของสัญญาณไม่เกิน -6 dB เมื่อเทียบกับสัญญาณหลักได้ภายในเวลาเพียง 3-4 วินาที แต่หากสัญญาณผีมีความเข้มของสัญญาณอยู่ในช่วง -6 dB ถึง -3 dB จะต้องใช้เวลาในการกำจัดผีมากขึ้น ทั้งนี้เนื่องจากความเข้มของสัญญาณผีที่ไม่เกิน -6 dB เครื่องกำจัดผีสามารถทำงานได้อย่างเต็มประสิทธิภาพ กล่าวคือสัญญาณซิงก์จะยังมีลักษณะดีพอที่ทำให้การเก็บข้อมูลสัญญาณอ้างอิงเป็นไปอย่างถูกต้องทุกเฟรมภาพ แต่หากสัญญาณผีมีความเข้มสูงกว่า -6 dB ก็จะทำให้การนับเส้นผิดพลาด ส่งผลให้ไม่สามารถเก็บข้อมูลสัญญาณอ้างอิงได้อย่างถูกต้อง ตัวประมวลผลก็จะไม่ได้คำนวณปรับค่าสัมประสิทธิ์ในเฟรมนั้นๆ ทำให้เวลาที่ใช้ในการคำนวณเพิ่มมากขึ้น และสัญญาณผีที่มีความแรงมากกว่า -3 dB จะไม่สามารถกำจัดผีได้หมด
3. *สัญญาณผีมีความหน่วงมากกว่า +10 µs* พบว่าสามารถกำจัดผีที่มีความเข้มของสัญญาณผีไม่เกิน -12 dB เมื่อเทียบกับสัญญาณหลักได้เป็นอย่างดี แต่ถ้าสัญญาณมีความเข้มมากกว่า -12 dB พบว่าไม่สามารถกำจัดผีได้หมด

6.3 ปัญหาในการทำงาน

1. เนื่องจากเอฟพีจีเอที่ใช้เป็นรุ่นที่ค่อนข้างใหม่ ซึ่งวงจรภายในบางตัวยังไม่มีข้อมูลบางอย่างที่จำเป็นต่อการจำลองการทำงาน (Simulation) เช่นไม่มีข้อมูลเวลาประวิง (Delay Time) เป็นต้น ทำให้ไม่สามารถจำลองการทำงานที่มีผลของเวลาประวิงหลังจากผ่านการสังเคราะห์วงจร (Synthesis) ได้ จึงทำให้การตรวจสอบข้อผิดพลาดของวงจรใช้เวลามากขึ้น

2. เครื่อง Logic Analyzer ที่มีใช้ภายในห้องปฏิบัติการเป็นรุ่นเก่า มีอัตราการสุ่มตัวอย่างและหน่วยความจำภายในน้อย ทำให้การตรวจสอบการทำงานที่ผิดพลาดของเอฟพีจีเอ และตัวประมวลผลสัญญาณดิจิตอล ซึ่งทำงานที่ความถี่ค่อนข้างสูงเป็นไปโดยยาก

6.4 สรุป

เครื่องกำจัดผีในสัญญาณโทรทัศน์ระบบ PAL ได้ถูกพัฒนาขึ้น ด้วยการสังเคราะห์วงจรกรองเอฟไออาร์ลงบนเอฟพีจีเอ โดยนำเทคนิคการใช้ทรัพยากรร่วมมาช่วย ทำให้ประหยัดทรัพยากรในส่วนของวงจรกรองไปได้มากกว่า 6 เท่า และผลการทดสอบทำให้เห็นว่ากระบวนวิธีค่าเฉลี่ยกำลังสองน้อยที่สุด เป็นกระบวนวิธีที่เหมาะสมกับกระบวนการกำจัดผี ซึ่งทำให้เครื่อง

กำจัดผีที่พัฒนาขึ้นสามารถกำจัดผีที่มีความหน่วงไม่เกิน 10 μs ความเข้มไม่เกิน -6 dB เมื่อเทียบกับสัญญาณหลักได้ภายในเวลา 3-4 วินาที

การพัฒนาเครื่องกำจัดผีในสัญญาณโทรทัศน์ขึ้นมา น่าจะเป็นส่วนช่วยให้เกิดการตื่นตัวที่จะพัฒนาทางด้านคุณภาพของสัญญาณโทรทัศน์ภายในประเทศไทย ให้มีคุณภาพของสถานีส่ง และเครื่องรับโทรทัศน์ที่ดียิ่งขึ้น

6.5 ข้อเสนอแนะ

1. เนื่องจากการทดสอบการทำงานของเครื่องกำจัดผีในสัญญาณโทรทัศน์ ทำอยู่ภายในห้องปฏิบัติการ ซึ่งมีสัญญาณรบกวนน้อย ดังนั้นในอนาคตจึงควรทำการทดสอบในระบบที่มีสัญญาณรบกวนมากขึ้น เพื่อทดสอบความทนทานต่อสัญญาณรบกวน
2. เนื่องจากเครื่องจำลองการเกิดผีที่พัฒนาขึ้น เพื่อใช้ในการทดสอบการทำงานของเครื่องกำจัดผีนั้น เป็นอย่างง่ายอาจจะไม่ครอบคลุมผลของช่องสัญญาณที่เกิดขึ้นจริงตามธรรมชาติ ดังนั้นในอนาคตน่าจะได้ทำการทดสอบการทำงานกับระบบจริง ซึ่งต้องขอความร่วมมือกับทางสถานีแพร่ภาพสัญญาณโทรทัศน์ให้ส่งสัญญาณอ้างอิงมากับสัญญาณภาพด้วย หรือถ้าสามารถสร้างเครื่องจำลองการเกิดผีที่ให้ผลการทำงานใกล้เคียงกับช่องสัญญาณจริงมากๆ ก็จะสามารถทดสอบความสามารถของเครื่องกำจัดผีได้มากขึ้น
3. เนื่องจากเครื่องกำจัดผีที่พัฒนาขึ้น ใช้อุปกรณ์ราคาแพง เช่นชุดทดลอง TMS320C6211 DSP STARTER KIT ทำให้ยังไม่สามารถผลิตในเชิงอุสาหกรรมได้ ดังนั้นในอนาคตจึงควรจะพัฒนาตัวประมวลผลสัญญาณดิจิตอล ที่สามารถทำงานแทนตัวประมวลผลที่ใช้อยู่ ในราคาที่ถูกกว่ามากๆ เพื่อลดต้นทุนในการผลิตเครื่องกำจัดผี
4. เนื่องจากเอฟพีจีเอที่ใช้มีขนาดใหญ่ ดังนั้นอาจจะสามารถพัฒนาให้เอฟพีจีเอทำหน้าที่ในการหาค่าสัมประสิทธิ์ของวงจรกรองเองด้วย โดยอาศัยตัวไมโครคอนโทรลเลอร์ตัวเล็กๆ ภายนอกในการโปรแกรมเอฟพีจีเอให้งานเป็นตัวหาค่าสัมประสิทธิ์ทุกครั้งที่เปลี่ยนช่อง และหลังจากได้ค่าสัมประสิทธิ์ที่เหมาะสมแล้ว ก็โปรแกรมให้เอฟพีจีเอทำงานเป็นวงจรกรองเอฟไอออาร์ทำหน้าที่กำจัดผีต่อไป วิธีนี้จะช่วยลดต้นทุนในส่วนของตัวประมวลผลสัญญาณดิจิตอลลงได้

รายการอ้างอิง

1. Huang, W.C. and Lo, C.S. A Novel TV Ghost Cancellation System. IEEE International Conference on Consumer Electronics (1993) : 134 –135.
2. Kim, K.B.; Oh, J.; Lee, M.H.; Hwang, H. and Song, D.I. A New Ghost Cancellation System. IEEE International Conference on Consumer Electronics (1994) : 288 - 289.
3. Roy, S.; Yang, J. and Patel C.B. Ghost Cancellation for Advanced Compatible TV System using Complementary Sequences. IEEE Transaction on Consumer Electronics Vol.38 No.4 (September 1992) : 767-777
4. Miyasawa, H.; Matsuura, S. and etc. Development of A Ghost Cancel Reference Signal for TV Broadcasting. IEEE Transaction on Broadcasting Vol.35 No.4 (December 1989) : 339-347.
5. Fiallos, E.; Jarmusz, J.; Caron, B. and Ledoux, B. An Enhanced Ghost Cancellation Reference Signal. IEEE Transaction on Consumer Electronics Vol.40 No.3 (August 1994) : 296-297
6. OLU, A.A. and Brahmaji, P. VLSI Echo Cancellation Filter. Fourth Annual IEEE International ASIC Conference and Exhibit (1991) : P3-2/1-4
7. Cole, E. and Sergio, K. A VLSI Chip Set for Ghost Cancellation and Waveform Equalization of Analog Television Signals. IEEE International Conference on Acoustics, Speech and Signal Processing Vol.3 (1992) : 541-544.
8. William, L.; Edward, C. and Paul, K. A Reconfigurable Video Ghost Cancelling Filter Chip. IEEE Conference on Custom Integrated Circuits (1993) : 15.3.1-15.3.3.
9. Tactaaki, T.; Reiichi, K. and etc. Ghost Cancelling System for NTSC TV Receiver. IEEE International Conference on Consumer Electronics (1989) : 68-69
10. Craig, B.G. Ghost Cancellation System for The US Startdard GCR. IEEE Transactions on Consumer Electornics Vol.39 No.4 (November 1993) : 928-933
11. Bhasker, J. A VHDL Primer. Prentice-Hall, 1992.
12. Haykin, S. Adaptive Filter Theory. 3rd ed. New Jersey : Prentice-Hall, 1996.

13. Bucher, M.L. Simulation of multipath fading/ghosting for analog and digital television transmission in broadcast channels. IEEE Transactions on broadcasting Vol.38 (December 1992) : 256-262.

14. Alan, V.O.; Ronald, W.S. and John, R.B. Discrete-Time Signal Processing. 2nd ed. New Jersey : Prentice-Hall, 1999.

15. Jinshi, H. A Ghost Cancelation System For The NTSC Television. IEEE Transactions on Consumer Electronics Vol.39 No.4 (November 1993) : 896-904.

16. Crawford, D. Adaptive Filters. [Online], Available from: http://www.spd.eee.strath.ac.uk/~david/adapt_filt/adapt_filt.html. [2002, Feb].

17. Sims, H.V. Principles of PAL Colour Television and Related Systems. 2nd ed. London lliffe books, 1969

18. Recommendation ITU-R BT.1124-3. Reference signals for ghost cancelling in analogue television system. 2001.

19. Report ITU-R BT.2018. Study of the system C ghost canceling reference signal for the evaluation and correction of linear distortion in the television chain. 1998.

20. National Semiconductor Incorporation. LM1881 Video Sync Separator. [Online], Available from: http://benedikt.lphard.cz/PRVKY.EL/LM1881.PDF. [2002, Feb].

21. Texas Instruments Incorporation. THS1031 3-V TO 5.5-V, 10-bit, 30 MSPS CMOS ANALOG-TO-DIGITAL CONVERTER. [Online], Available from: http://www-s.ti.com/sc/ds/ths1031.pdf. [2002, Feb].

22. Xilinx Incorporation. Xilinx Foundation 3.3i. [Online], Available from: http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+Foundation. [2002, Feb].

23. Xilinx Incorporation. The Programmable Logic Data Book 2000. United State of America, 2000.

24. Texas Instruments Incorporation. TMS320C6211 DSP STARTER KIT. [Online], Available from: http://focus.ti.com/docs/tool/toolfolder.jhtml?PartNumber=TMDX320006211. [2002, Feb].

25. Texas Instruments Incorporation. THS5641 8-BIT, 100 MSPS, CommsDAC™ DIGITAL-TO-ANALOG CONVERTER. [Online], Available from: http://www-s.ti.com/sc/ds/ths5641.pdf. [2002, Feb].

26. Xilinx Incorporation. CORE Generator Cores & IP Updates. [Online], Available from: http://www.xilinx.com/ ipcenter/coregen/3_x/updates4.htm. [2002, Feb].

27. Xilinx Incorporation. Multiply Generator. [Online], Available from: http://www.xilinx.com/ipcenter/catalog/logicore/docs/mult_gen.pdf. [2002, Feb].

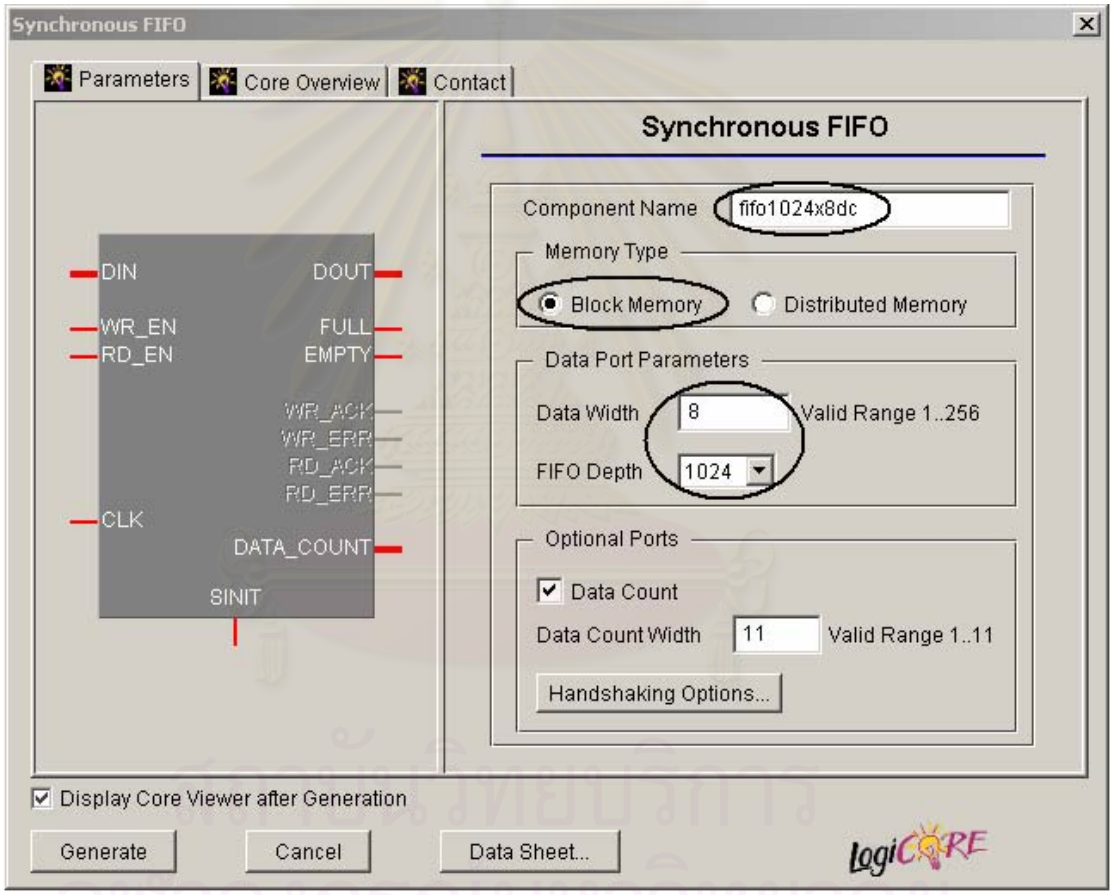
28. Xilinx Incorporation. Transposed Form FIR Filters. [Online], Available from: http://www.xilinx.com/xapp/xapp219.pdf. [2002, Feb].

ภาคผนวก

ภาคผนวก ก
การสร้างวงจรต่าง ๆ ด้วยซอฟท์แวร์ IP Core Generator

Synchronous FIFO

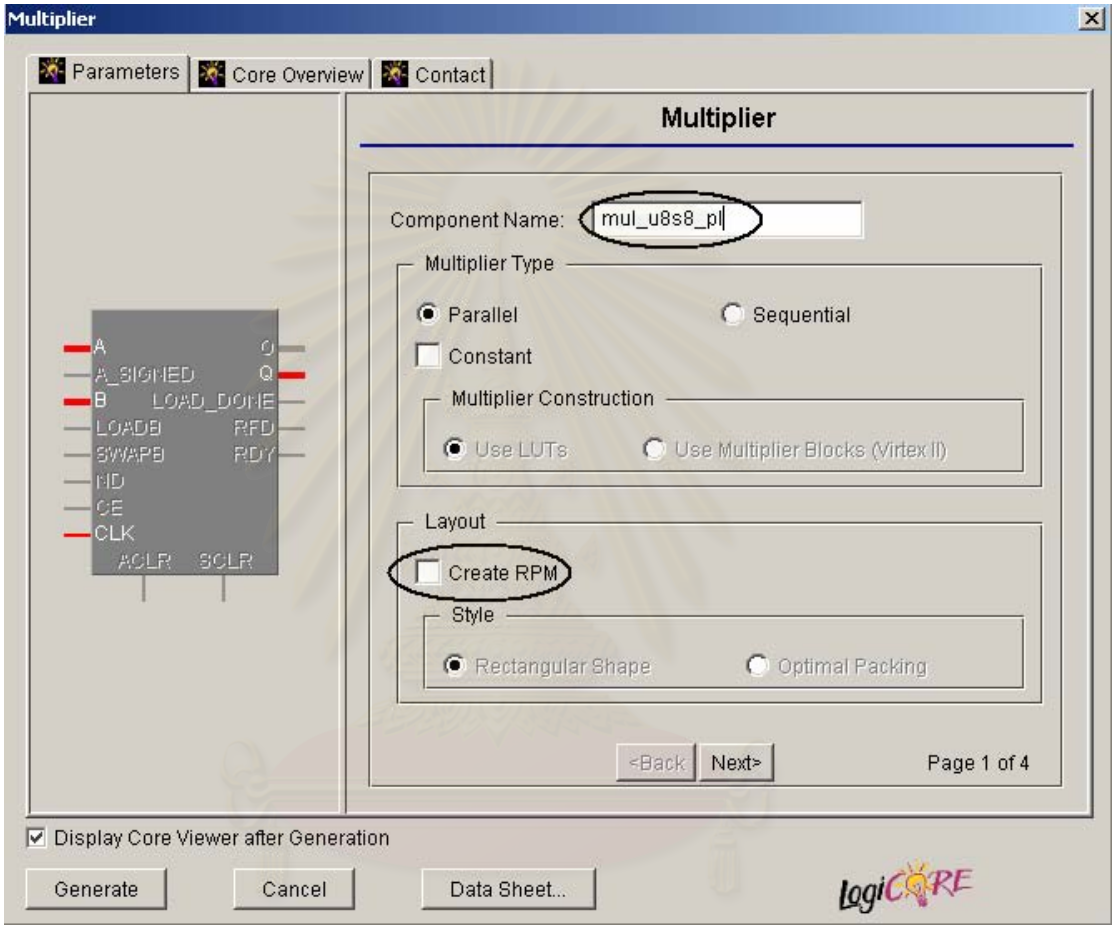
การกำหนดค่าต่างๆ ที่สำคัญในการสร้างหน่วยความจำแบบเข้าก่อนออกก่อน ได้แก่ การกำหนดชื่อให้กับวงจรที่จะสร้างขึ้น, กำหนดให้สร้างหน่วยความจำด้วยการใช้ทรัพยากรในส่วนบล็อกหน่วยความจำ, กำหนดขนาดความกว้างของข้อมูลเป็น 8 บิต และกำหนดความลึกของหน่วยความจำแบบเข้าก่อนออกก่อนเป็น 1024 ไบต์ ดังรูป



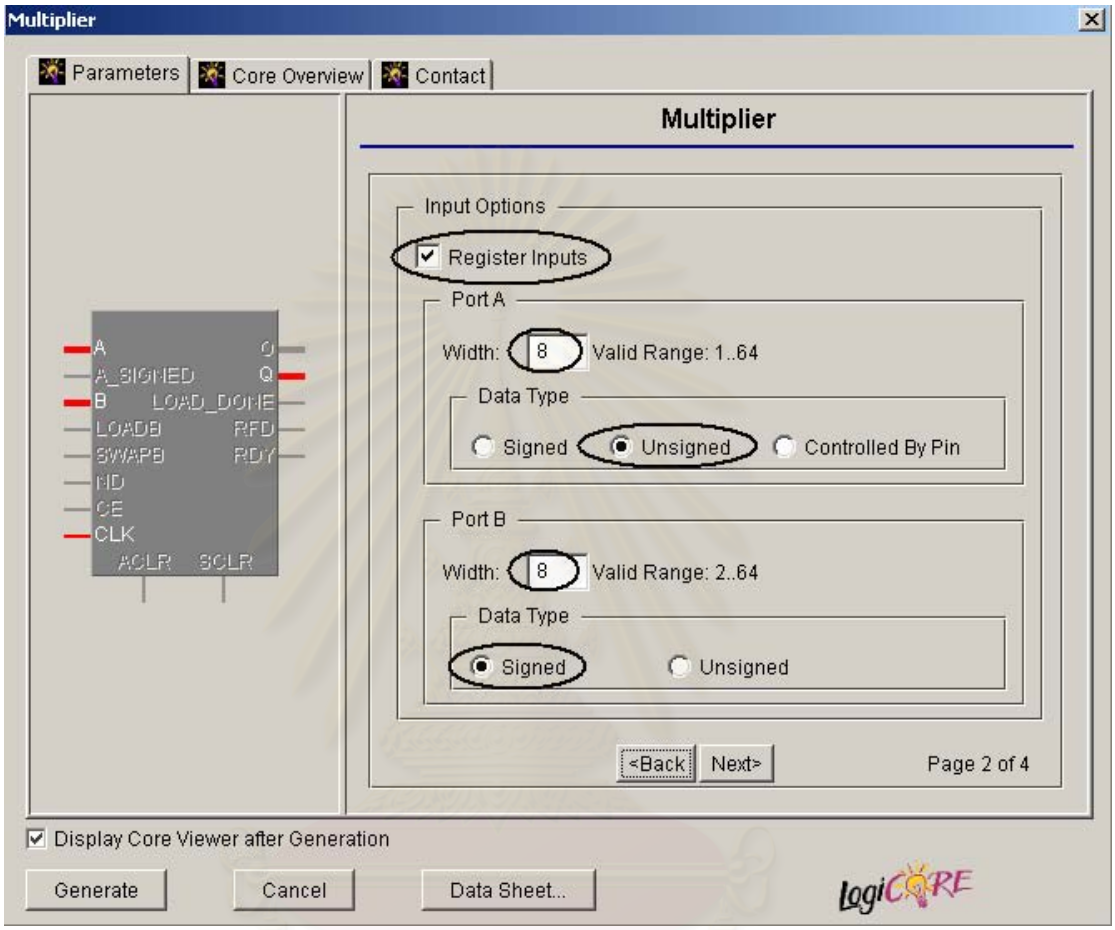
Multiplier

การกำหนดค่าต่างๆ ที่สำคัญในการสร้างวงจรคูณจะแบ่งออกเป็น 4 หน้า ดังนี้

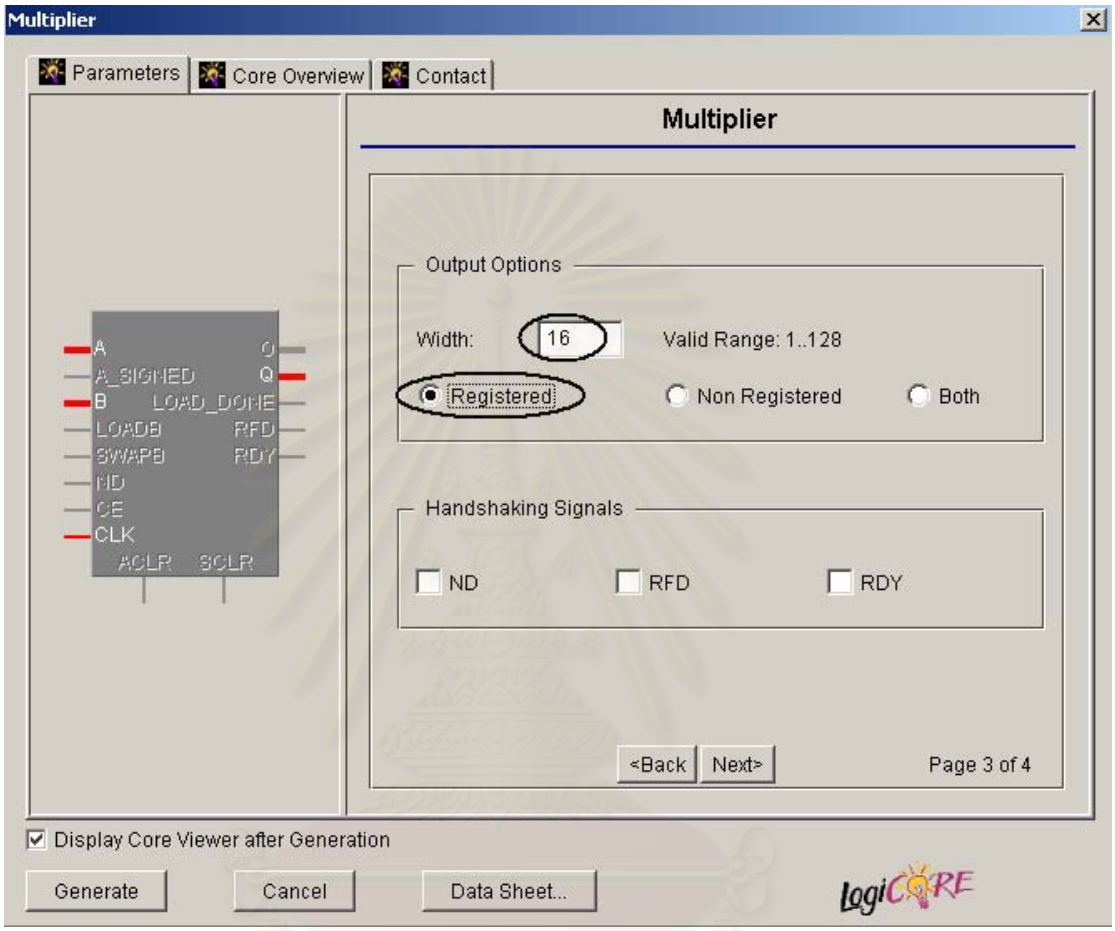
หน้าแรก ต้องกำหนดชื่อของวงจรที่จะสร้างขึ้น และต้องไม่เลือกตัวเลือก Create RPM เพื่อให้วงจรมีความยืดหยุ่นในการจัดเรียงตัวลงในเอฟพีจีเอ



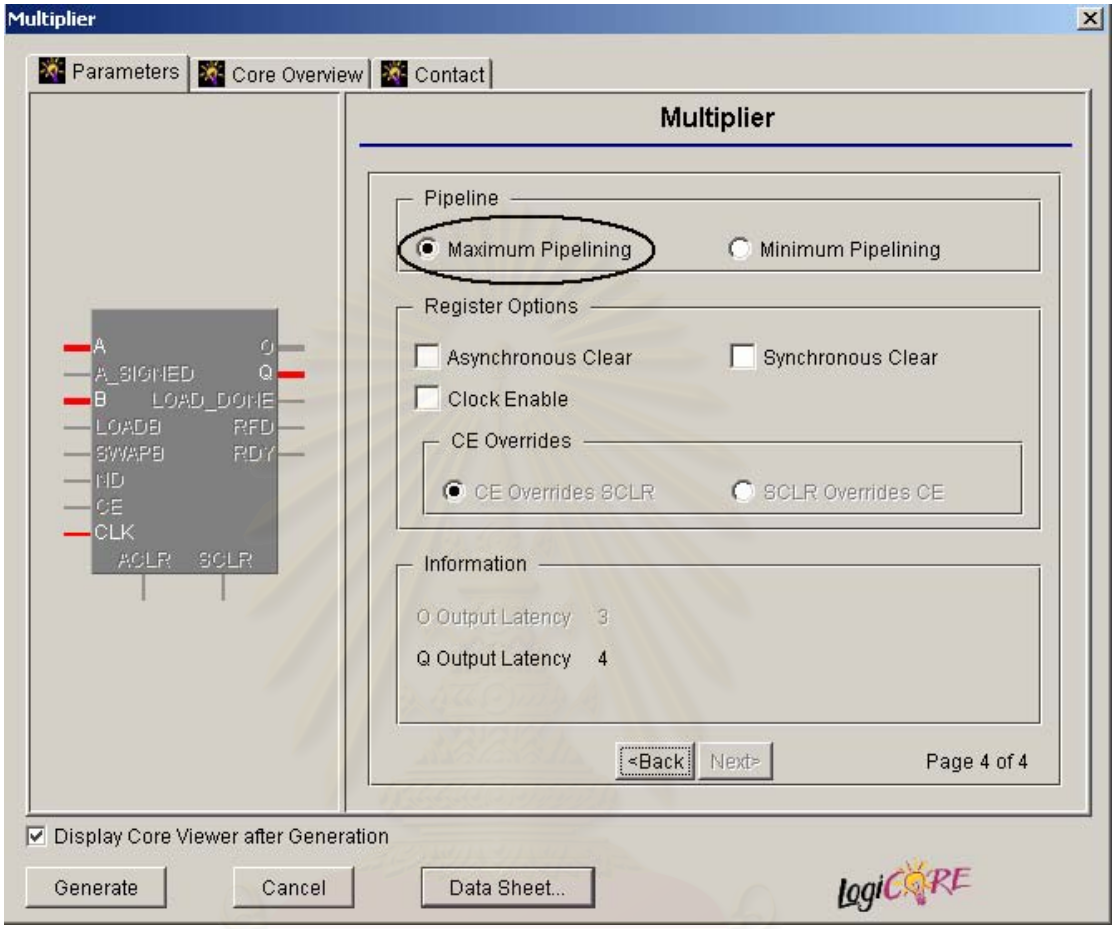
หน้าที่สอง เป็นการกำหนดเกี่ยวกับสัญญาณขาเข้าของวงจรคูณ โดยเริ่มจากกำหนดให้ใช้รีจิสเตอร์ขาเข้าด้วย จากนั้นก็กำหนดความกว้างของข้อมูลขาเข้าที่พอร์ท A เป็น 9 แบบมีเครื่องหมาย และความกว้างของข้อมูลที่พอร์ท B เป็น 8 แบบมีเครื่องหมายเช่นกัน



หน้าที่สาม เป็นการกำหนดเกี่ยวกับข้อมูลขาออก โดยปกติความกว้างของข้อมูลขาออกจะเป็นผลรวมของข้อมูลขาเข้าอยู่แล้วคือ 17 บิต และกำหนดให้มีรีจิสเตอร์ขาออกด้วย เพื่อเพิ่มประสิทธิภาพการทำงาน

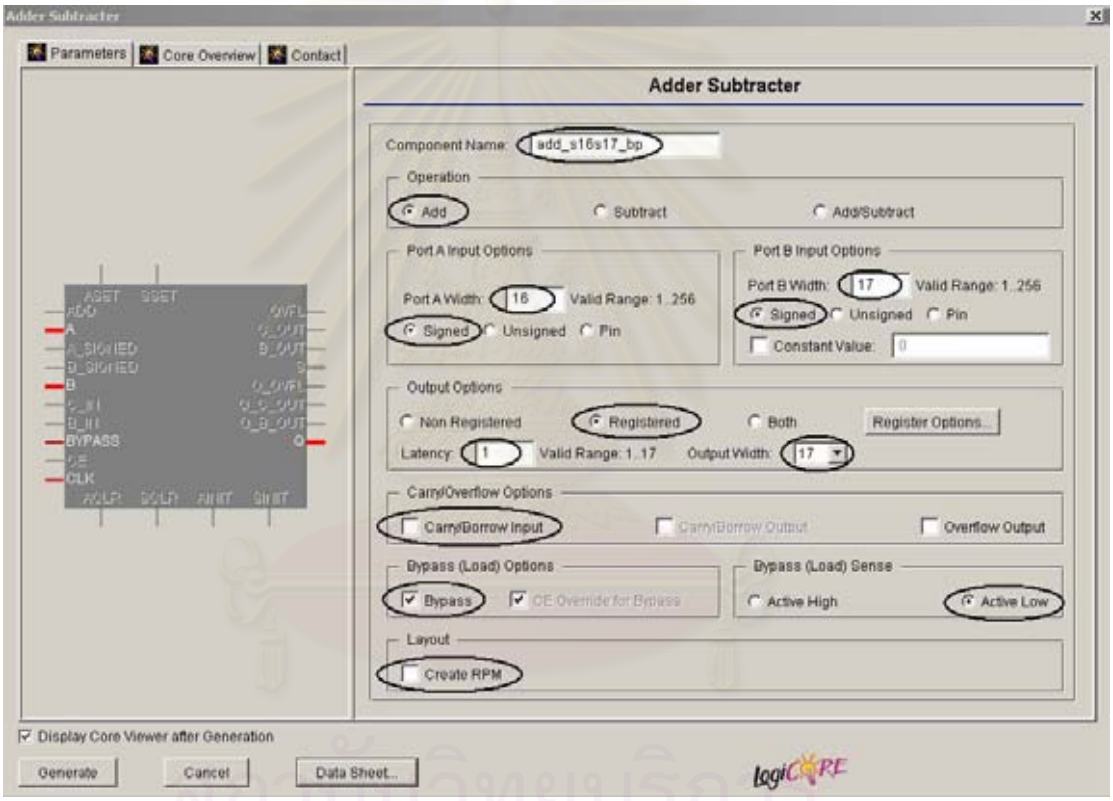


หน้าที่สี่ เป็นการกำหนดเกี่ยวกับรีจิสเตอร์ภายในวงจรคูณ โดยในงานวิจัยนี้เราต้องการความเร็วสูงที่สุดจากวงจรคูณจึงต้องกำหนดให้ใช้ Maximum Pipelining



Adder

การกำหนดค่าต่างๆ ที่สำคัญในการสร้างวงจรบวก ได้แก่ การกำหนดชื่อให้กับวงจรที่จะสร้างขึ้นมา, กำหนดให้เป็นวงจรบวก, กำหนดขนาดของข้อมูลขาเข้าทั้งพอร์ท A และ B ให้เป็น 17 บิต แบบมีเครื่องหมาย, กำหนดข้อมูลขาออกเป็นแบบรีจิสเตอร์ โดยมีเวลาแฝงเป็น 1 ความกว้างของข้อมูลเป็น 17 บิต, ไม่ต้องเลือกตัวเลือก Carry/Borrow Input, เลือกตัวเลือก Bypass เพื่อให้สามารถปล่อยผ่านข้อมูลขาเข้าด้านพอร์ท A ออกไปได้ โดยกำหนดให้ขาสัญญาณ Bypass เป็นแบบ Active Low และที่สำคัญไม่ต้องเลือกตัวเลือก Create RPM เพื่อให้วงจรมีความยืดหยุ่นในการเรียงตัวกันภายในเอฟพีจีเอ



ภาคผนวก ข
แฟ้มเงื่อนไขบังคับ

MAIN.UCF

```
############################################################
#                I/F DSP                   #
############################################################
NET "addr_bus<2>" LOC = "P220";
NET "addr_bus<3>" LOC = "P218";
NET "addr_bus<4>" LOC = "P216";
NET "addr_bus<5>" LOC = "P215";
NET "addr_bus<6>" LOC = "P206";
NET "addr_bus<7>" LOC = "P205";
NET "addr_bus<8>" LOC = "P203";
NET "addr_bus<9>" LOC = "P202";
NET "addr_bus<10>" LOC = "P201";
NET "addr_bus<11>" LOC = "P200";
NET "addr_bus<12>" LOC = "P199";
NET "addr_bus<13>" LOC = "P195";
NET "addr_bus<14>" LOC = "P194";
NET "addr_bus<15>" LOC = "P193";
NET "addr_bus<16>" LOC = "P192";
NET "addr_bus<17>" LOC = "P191";
NET "addr_bus<18>" LOC = "P189";
NET "addr_bus<19>" LOC = "P188";
NET "addr_bus<20>" LOC = "P187";
NET "addr_bus<21>" LOC = "P186";
NET "data_bus<0>" LOC = "P49";
NET "data_bus<1>" LOC = "P48";
NET "data_bus<2>" LOC = "P47";
NET "data_bus<3>" LOC = "P46";
NET "data_bus<4>" LOC = "P42";
NET "data_bus<5>" LOC = "P41";
NET "data_bus<6>" LOC = "P40";
NET "data_bus<7>" LOC = "P39";
NET "data_bus<8>" LOC = "P36";
NET "data_bus<9>" LOC = "P35";
NET "data_bus<10>" LOC = "P34";
NET "data_bus<11>" LOC = "P33";
NET "data_bus<12>" LOC = "P28";
NET "data_bus<13>" LOC = "P27";
NET "data_bus<14>" LOC = "P26";
NET "data_bus<15>" LOC = "P24";
NET "data_bus<16>" LOC = "P23";
NET "data_bus<17>" LOC = "P21";
NET "data_bus<18>" LOC = "P20";
NET "data_bus<19>" LOC = "P19";
NET "data_bus<20>" LOC = "P18";
NET "data_bus<21>" LOC = "P17";
NET "data_bus<22>" LOC = "P13";
NET "data_bus<23>" LOC = "P12";
NET "data_bus<24>" LOC = "P11";
NET "data_bus<25>" LOC = "P10";
NET "data_bus<26>" LOC = "P9";
NET "data_bus<27>" LOC = "P7";
NET "data_bus<28>" LOC = "P6";
NET "data_bus<29>" LOC = "P5";
NET "data_bus<30>" LOC = "P4";
```

```
NET "data_bus<31>" LOC = "P3";
NET "n_ce" LOC = "P54";
NET "n_oe" LOC = "P53";
NET "n_rd" LOC = "P50";
NET "n_wr" LOC = "P52";
##############################################################
#                  I/F Demo Board                  #
##############################################################
NET "n_sw2<0>" LOC = "P238";
NET "n_sw2<1>" LOC = "P237";
NET "n_sw2<2>" LOC = "P236";
NET "n_sw2<3>" LOC = "P235";
NET "n_sw2<4>" LOC = "P234";
NET "n_sw2<5>" LOC = "P231";
NET "n_sw2<6>" LOC = "P230";
NET "n_sw2<7>" LOC = "P229";
#NET "n_sw2<8>" LOC = "P228";
NET "n_SW<0>" LOC = "P145";
NET "n_SW<1>" LOC = "P138";
NET "n_SW<2>" LOC = "P134";
NET "n_SW<3>" LOC = "P124";
NET "n_LED<0>" LOC = "P177";
NET "n_LED<1>" LOC = "P167";
NET "n_LED<2>" LOC = "P163";
NET "n_LED<3>" LOC = "P156";
NET "n_SEG<0>" LOC = "P160";
NET "n_SEG<1>" LOC = "P153";
NET "n_SEG<2>" LOC = "P141";
NET "n_SEG<3>" LOC = "P149";
NET "n_SEG<4>" LOC = "P155";
NET "n_SEG<5>" LOC = "P162";
NET "n_SEG<6>" LOC = "P142";
NET "n_SEG<7>" LOC = "P152";
NET "SEG_EN<0>" LOC = "P128";
NET "SEG_EN<1>" LOC = "P125";
NET "SEG_EN<2>" LOC = "P139";
NET "SEG_EN<3>" LOC = "P127";
##############################################################
#                     Clock                     #
##############################################################
#NET "clk_p" LOC = "P92";
#NET "adc_clk_p" LOC = "P89";
NET "clkdv_p" LOC = "P94";
INST "div2_mul4_1/dll2x" LOC = "DLL2S";
INST "div2_mul4_1/dll4x" LOC = "DLL2P";
INST "div2_mul4_1/dlldv" LOC = "DLL0S";
#INST "div2_mul4_1_clkdvg" LOC = "GCLKBUF1";
INST "div2_mul4_1/clk0g" LOC = "GCLKBUF0";
INST "div2_mul4_1/clk4xg" LOC = "GCLKBUF3";
INST "div2_mul4_1/clk2xg" LOC = "GCLKBUF2";
INST "div2_mul4_1/clkpad" LOC = "GCLKPAD0";
INST "div2_mul4_1/adc_clk_bufg" LOC = "GCLKBUF1";
INST "div2_mul4_1/adc_clk_ibufg" LOC = "GCLKPAD1";
NET "adc_clk_p" TNM_NET = "adc_clk_p";
TIMESPEC "TS_adc_clk_p" = PERIOD "adc_clk_p" 80 ns HIGH 50 %;
NET "clk_p" TNM_NET = "clk_p";
TIMESPEC "TS_clk_p" = PERIOD "clk_p" 40 ns HIGH 50 %;
##############################################################
#                I/F Analog Board                  #
##############################################################
NET "comp_sync_p" LOC = "P84";
NET "dac_data_p<9>" LOC = "P70";
NET "dac_data_p<8>" LOC = "P71";
```

```
NET "dac_data_p<7>" LOC = "P72";
NET "dac_data_p<6>" LOC = "P73";
NET "dac_data_p<5>" LOC = "P74";
NET "dac_data_p<4>" LOC = "P78";
NET "dac_data_p<3>" LOC = "P79";
NET "dac_data_p<2>" LOC = "P80";
#NET "dac_data_p<1>" LOC = "P81";
#NET "dac_data_p<0>" LOC = "P82";
NET "adc_data_p<9>" LOC = "P99";
NET "adc_data_p<8>" LOC = "P100";
NET "adc_data_p<7>" LOC = "P101";
NET "adc_data_p<6>" LOC = "P102";
NET "adc_data_p<5>" LOC = "P103";
NET "adc_data_p<4>" LOC = "P107";
NET "adc_data_p<3>" LOC = "P108";
NET "adc_data_p<2>" LOC = "P109";
#NET "adc_data_p<1>" LOC = "P110";
#NET "adc_data_p<0>" LOC = "P111";
##############################################################
#                  Debug                     #
##############################################################
NET "debug<0>" LOC = "P174";
NET "debug<1>" LOC = "P173";
NET "debug<2>" LOC = "P171";
NET "debug<3>" LOC = "P170";
NET "debug<4>" LOC = "P169";
NET "debug<5>" LOC = "P168";
```

ภาคผนวก ค
การออกแบบวงจรด้วยภาษาวีเอชดีแอล

p_ghost.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Package P_GHOST is
    -- Demo Board
    type HEX4 is array(3 downto 0) of std_logic_vector(3 downto 0);-- L => R

    -- Filter
    Constant WordSize         : integer := 8;
    Constant TapSize          : integer := 128;
    Constant ShareSize        : integer := 3;
    Constant SharingFactor    : integer := 2**ShareSize;
    Constant ComponentSize : integer := TapSize/SharingFactor;
    Constant PipeLineLevel   : integer := 2;

    subtype UnsignedShareType is unsigned(ShareSize-1 downto 0);
    subtype UnsignedWordType is unsigned(WordSize-1 downto 0);
    subtype UnsignedDWordType is unsigned(2*WordSize-1 downto 0);
    type UnsignedPipeLineType is array(PipeLineLevel-1 downto 0) of UnsignedDWordType;
    type ArrayUnsignedWordType is array(ComponentSize-1 downto 0) of UnsignedWordType;
    type ArrayUnsignedDWordType is array(ComponentSize-1 downto 0) of UnsignedDWordType;-- 256
    type ArrayUnsignedDWordType2 is array(ComponentSize/2-1 downto 0) of UnsignedDWordType;-- 128
    type ArrayUnsignedDWordType4 is array(ComponentSize/4-1 downto 0) of UnsignedDWordType;-- 64
    type ArrayUnsignedDWordType8 is array(ComponentSize/8-1 downto 0) of UnsignedDWordType;-- 32
    subtype SignedShareType is signed(ShareSize-1 downto 0);
    subtype SignedWordType is signed(WordSize-1 downto 0);
    subtype SignedWordType_1 is signed(WordSize downto 0);
    subtype SignedWordType_3 is signed(WordSize+2 downto 0);
    subtype SignedDWordType is signed(2*WordSize-1 downto 0);
    subtype SignedDWordType_1 is signed(2*WordSize downto 0);
    subtype SignedDWordType_3 is signed(2*WordSize+2 downto 0);
    type SignedPipeLineType is array(PipeLineLevel-1 downto 0) of SignedDWordType;
    type SignedPipeLineType_1 is array(PipeLineLevel-1 downto 0) of SignedDWordType_1;
    type SignedPipeLineType_3 is array(PipeLineLevel-1 downto 0) of SignedDWordType_3;
    type ArraySignedWordType is array(ComponentSize-1 downto 0) of SignedWordType;
    type ArraySignedDWordType is array(ComponentSize-1 downto 0) of SignedDWordType;      -- 256
    type ArraySignedDWordType_3 is array(ComponentSize-1 downto 0) of SignedDWordType_3;-- 256
    type ArraySignedDWordType2 is array(ComponentSize/2-1 downto 0) of SignedDWordType;-- 128
    type ArraySignedDWordType2_3 is array(ComponentSize/2-1 downto 0) of SignedDWordType_3;-- 128
    type ArraySignedDWordType4 is array(ComponentSize/4-1 downto 0) of SignedDWordType;  -- 64
    type ArraySignedDWordType4_3 is array(ComponentSize/4-1 downto 0) of SignedDWordType_3;-- 64
    type ArraySignedDWordType8 is array(ComponentSize/8-1 downto 0) of SignedDWordType;  -- 32
    type ArraySignedDWordType8_3 is array(ComponentSize/8-1 downto 0) of SignedDWordType_3;-- 32
    subtype ShareType is std_logic_vector(ShareSize-1 downto 0);
    subtype WordType is std_logic_vector(WordSize-1 downto 0);
    subtype WordType_1 is std_logic_vector(WordSize downto 0);
    subtype WordType_3 is std_logic_vector(WordSize+2 downto 0);
    subtype DWordType is std_logic_vector(2*WordSize-1 downto 0);
    subtype DWordType_1 is std_logic_vector(2*WordSize downto 0);
    subtype DWordType_3 is std_logic_vector(2*WordSize+2 downto 0);
    type ArrayWordType is array(ComponentSize-1 downto 0) of WordType;
    type ArrayDWordType is array(ComponentSize-1 downto 0) of DWordType;-- 256
    type ArrayDWordType_3 is array(ComponentSize-1 downto 0) of DWordType_3;     -- 256
```

```
    type ArrayDWordType2 is array(ComponentSize/2-1 downto 0) of DWordType;       -- 128
    type ArrayDWordType2_3 is array(ComponentSize/2-1 downto 0) of DWordType_3;-- 128
    type ArrayDWordType4 is array(ComponentSize/4-1 downto 0) of DWordType;       -- 64
    type ArrayDWordType4_3 is array(ComponentSize/4-1 downto 0) of DWordType_3;-- 64
    type ArrayDWordType8 is array(ComponentSize/8-1 downto 0) of DWordType;       -- 32
    type ArrayDWordType8_3 is array(ComponentSize/8-1 downto 0) of DWordType_3;-- 32
End P_GHOST;

Package body P_GHOST is
        --Body Part
End P_GHOST;
```

show7seg.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.p_ghost.all;

Entity SHOW7SEG is
-- generic parameters
-- port list
        port(   clk         : in std_logic;
                -- In Chip
                HEX     : in HEX4;
                DOT     : in std_logic_vector(3 downto 0);
                LED     : in std_logic_vector(3 downto 0);
                SW      : out std_logic_vector(3 downto 0);
                sw2     : out std_logic_vector(7 downto 0);
                -- Out Chip
                SEG_EN: out std_logic_vector(3 downto 0);
                n_SEG   : out std_logic_vector(7 downto 0);     -- ABCEDFG.
                n_LED   : out std_logic_vector(3 downto 0);
                n_SW    : in std_logic_vector(3 downto 0);
                n_sw2   : in std_logic_vector(7 downto 0)
                );
end SHOW7SEG;

Architecture Behavior of SHOW7SEG is
-- declarations
        signal state        : unsigned(1 downto 0);
        signal HEX1         : std_logic_vector(3 downto 0);
        signal SEG1         : std_logic_vector(6 downto 0);
        signal cnt10b       : unsigned(9 downto 0);
        signal mDOT         : std_logic;
begin
-- concurrent statements
        Process(clk)
        begin
                if rising_edge(clk) then
                        cnt10b  <= cnt10b+1;
                end if;
        end process;

        Process(clk)
        begin
                if rising_edge(clk) then
                        if cnt10b=1023 then
                                state       <=      state+1;
                                n_SEG(6 downto 0)       <=      SEG1;
                                case state is
                                        when "00"       =>
                                                SEG_EN <=       "0001";
                                                mDOT    <=      DOT(0);
                                        when "01"       =>
                                                SEG_EN <=       "0010";
                                                mDOT    <=      DOT(1);
                                        when "10"       =>
                                                SEG_EN <=       "0100";
                                                mDOT    <=      DOT(2);
                                        when others     =>
                                                SEG_EN <=       "1000";
                                                mDOT    <=      DOT(3);
                                end case;
```

```
                    end if;
            end if;
        end process;

        n_SEG(7)        <=      not mDOT;

        with state select
            HEX1    <=      HEX(0) when "00",
                            HEX(1) when "01",
                            HEX(2) when "10",
                            HEX(3) when others;

        with HEX1 select
            SEG1    <=      "1000000" when "0000",
                            "1111001" when "0001",
                            "0100100" when "0010",
                            "0110000" when "0011",
                            "0011001" when "0100",
                            "0010010" when "0101",
                            "0000010" when "0110",
                            "1111000" when "0111",
                            "0000000" when "1000",
                            "0010000" when "1001",
                            "0001000" when "1010",
                            "0000011" when "1011",
                            "0100111" when "1100",
                            "0100001" when "1101",
                            "0000110" when "1110",
                            "0001110" when others;

        n_LED   <=      not LED;
        SW              <=      not n_SW;
        sw2             <=      not n_sw2;

end Behavior;
```

div2_mul4.vhd

```
-- XAPP132
--
-- DLL 2X and 4X Example
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
-- pragma translate_off
library unisim;
use unisim.vcomponents.all;
-- pragma translate_on
entity DIV2_MUL4 is
    port (   CLKIN              : in  std_logic;
             RESET              : in  std_logic;
             CLK                : out std_logic;
             CLKDV              : out std_logic;
             CLK2X              : out std_logic;
             CLK4X              : out std_logic;
             LOCKED1            : out std_logic;
             LOCKED2            : out std_logic;
             adc_clk_p          : in std_logic;
             adc_clk            : out std_logic );
end DIV2_MUL4;

architecture structural of DIV2_MUL4 is

component IBUFG
-- synopsys translate_off
  generic(
     TimingChecksOn: Boolean := DefaultTimingChecksOn;
     InstancePath: STRING := "*";
     Xon: Boolean := DefaultXon;
     MsgOn: Boolean := DefaultMsgOn;
     tpd_I_O          : VitalDelayType01 := (0.100 ns, 0.100 ns);
     tipd_I           : VitalDelayType01 := (0.000 ns, 0.000 ns));

-- synopsys translate_on
  port(    O          : out   STD_ULOGIC;
           I          : in    STD_ULOGIC);
end component;

component IBUF
-- synopsys translate_off
  generic(
     TimingChecksOn: Boolean := DefaultTimingChecksOn;
     InstancePath: STRING := "*";
     Xon: Boolean := DefaultXon;
     MsgOn: Boolean := DefaultMsgOn;
     tpd_I_O               : VitalDelayType01 := (0.100 ns, 0.100 ns);
     tipd_I                : VitalDelayType01 := (0.000 ns, 0.000 ns));

-- synopsys translate_on
  port(    O          : out   STD_ULOGIC;
           I          : in    STD_ULOGIC);
end component;

component CLKDLL
-- synopsys translate_off
    generic ( TimingChecksOn : Boolean := DefaultTimingChecksOn;
              InstancePath : STRING := "*";
```

```
            Xon  : Boolean := DefaultXon;
            MsgOn : Boolean := DefaultMsgOn;

            tipd_CLKIN   : VitalDelayType01 := (0.000 ns, 0.000 ns);
            tipd_CLKFB  : VitalDelayType01 := (0.000 ns, 0.000 ns);
            tipd_RST      : VitalDelayType01 := (0.000 ns, 0.000 ns);

            tpd_CLKIN_LOCKED  : VitalDelayType01 := (0.100 ns, 0.100 ns);

            tperiod_CLKIN_POSEDGE     : VitalDelayType  := 0.010 ns;
            MAXPERCLKIN       : time := 100 ns;

            tpw_CLKIN_posedge : VitalDelayType  := 0.010 ns;
            tpw_CLKIN_negedge : VitalDelayType  := 0.010 ns;

            tpw_RST_posedge   : VitalDelayType  := 0.010 ns;

             DUTY_CYCLE_CORRECTION : Boolean := TRUE;
             CLKDV_DIVIDE : real := 2.0);

-- synopsys translate_on
    port ( CLKIN      : in  std_ulogic := '0';
           CLKFB      : in  std_ulogic := '0';
           RST        : in  std_ulogic := '0';
           CLK0       : out std_ulogic := '0';
           CLK90      : out std_ulogic := '0';
           CLK180     : out std_ulogic := '0';
           CLK270     : out std_ulogic := '0';
           CLK2X      : out std_ulogic := '0';
           CLKDV      : out std_ulogic := '0';
           LOCKED     : out std_ulogic := '0');
end component;

component BUFG
-- synopsys translate_off
  generic(
     TimingChecksOn: Boolean := DefaultTimingChecksOn;
     InstancePath: STRING := "*";
     Xon: Boolean := DefaultXon;
     MsgOn: Boolean := DefaultMsgOn;
     tpd_I_O                 : VitalDelayType01 := (0.100 ns, 0.100 ns);
     tipd_I                  : VitalDelayType01 := (0.000 ns, 0.000 ns));

-- synopsys translate_on
  port(    O          : out   STD_ULOGIC;
           I          : in    STD_ULOGIC);
end component;
component OBUF
-- synopsys translate_off
  generic(
     TimingChecksOn: Boolean := DefaultTimingChecksOn;
     InstancePath: STRING := "*";
     Xon: Boolean := DefaultXon;
     MsgOn: Boolean := DefaultMsgOn;
     tpd_I_O                 : VitalDelayType01 := (0.100 ns, 0.100 ns);
     tipd_I                  : VitalDelayType01 := (0.000 ns, 0.000 ns));

-- synopsys translate_on
  port(
           O          : out   STD_ULOGIC;
           I          : in    STD_ULOGIC);
end component;
```

```
component SRL16
-- synopsys translate_off
  generic (
      TimingChecksOn: Boolean := DefaultTimingChecksOn;
      InstancePath: STRING := "*";
      Xon: Boolean := DefaultXon;
      MsgOn: Boolean := DefaultMsgOn;

      -- VITAL input wire delays

      tipd_A0   : VitalDelayType01 := (0.0 ns,  0.0 ns);
      tipd_A1   : VitalDelayType01 := (0.0 ns,  0.0 ns);
      tipd_A2   : VitalDelayType01 := (0.0 ns,  0.0 ns);
      tipd_A3   : VitalDelayType01 := (0.0 ns,  0.0 ns);

      tipd_D    : VitalDelayType01 := (0.0 ns,  0.0 ns);
      tipd_CLK  : VitalDelayType01 := (0.0 ns,  0.0 ns);

      -- VITAL pin-to-pin propagation delays

      tpd_A0_Q  : VitalDelayType01 := (0.1 ns, 0.1 ns);
      tpd_A1_Q  : VitalDelayType01 := (0.1 ns, 0.1 ns);
      tpd_A2_Q  : VitalDelayType01 := (0.1 ns, 0.1 ns);
      tpd_A3_Q  : VitalDelayType01 := (0.1 ns, 0.1 ns);
      tpd_CLK_Q : VitalDelayType01 := (0.1 ns, 0.1 ns);


      -- VITAL setup and hold times

      tsetup_D_CLK_posedge_posedge  : VitalDelayType := 0.01 ns;
      tsetup_D_CLK_negedge_posedge  : VitalDelayType := 0.01 ns;

      thold_D_CLK_posedge_posedge   : VitalDelayType := 0.01 ns;
      thold_D_CLK_negedge_posedge   : VitalDelayType := 0.01 ns;


      -- VITAL minimum pulse width

      tpw_CLK_posedge : VitalDelayType := 0.01 ns;
      tpw_CLK_negedge : VitalDelayType := 0.01 ns;

      INIT : bit_vector := X"0000");

-- synopsys translate_on
  port (D   : in STD_ULOGIC;
        CLK             : in STD_ULOGIC;
        A0   : in STD_ULOGIC;
        A1   : in STD_ULOGIC;
        A2   : in STD_ULOGIC;
        A3   : in STD_ULOGIC;
        Q    : out STD_ULOGIC);
end component;

-- RESET_w,
signal CLK0_dll, CLK0_g, CLKIN_w, CLK2X_dll, CLK2X_g, CLK4X_dll, CLK4X_g : std_logic;
signal LOCKEDDV_dll, LOCKED2X, LOCKED2X_delay, RESET4X, LOCKED4X_dll : std_logic;
signal logic1 : std_logic;
--signal count : unsigned(3 downto 0);
signal DIVCLK      :              std_logic;

signal adc_clk_g,adc_clk_in  :              std_logic;

begin
```

```
logic1 <= '1';

clkpad : IBUFG  port map (I=>CLKIN, O=>CLKIN_w);

dlldv  : CLKDLL port map (  CLKIN=>CLKIN_w,  CLKFB=>CLK0_g, RST=>RESET,
                            CLK0=>CLK0_dll,   CLK90=>open, CLK180=>open, CLK270=>open,
                            CLK2X=>open, CLKDV=>open, LOCKED=>LOCKEDDV_dll);

LOCKED1 <= LOCKEDDV_dll;

clk0g  : BUFG   port map (I=>CLK0_dll,   O=>CLK0_g);
CLK       <=          CLK0_g;

dll2x  : CLKDLL port map (  CLKIN=>CLK0_g,   CLKFB=>CLK2X_g, RST=>RESET,
                            CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                            CLK2X=>CLK2X_dll, CLKDV=>open, LOCKED=>LOCKED2X);

clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);
CLK2X <= CLK2X_g;

rstsrl : SRL16  port map (      D=>LOCKED2X, CLK=>CLK2X_g, Q=>LOCKED2X_delay,
                            A3=>logic1, A2=>logic1, A1=>logic1, A0=>logic1);

RESET4X <= not LOCKED2X_delay;

dll4x  : CLKDLL port map (  CLKIN=>CLK2X_g,  CLKFB=>CLK4X_g, RST=>RESET4X,
                            CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                            CLK2X=>CLK4X_dll, CLKDV=>open, LOCKED=>LOCKED4X_dll);

LOCKED2 <= LOCKED4X_dll;

clk4xg : BUFG   port map (I=>CLK4X_dll,  O=>CLK4X_g);
CLK4X <= CLK4X_g;

        DivClockProcess:
        Process(CLK0_g)
        begin
                if rising_edge(CLK0_g) then
                        DIVCLK <=          not DIVCLK;
                end if;
        end process DivClockProcess;

        CLKDV  <=          DIVCLK;

        adc_clk_ibufg       : IBUFG port map  ( I => adc_clk_p, O => adc_clk_in );
        adc_clk_bufg        : BUFG port map   ( I => adc_clk_in, O => adc_clk_g );
        adc_clk   <=        adc_clk_g;
end structural;
```

fifo8todsp32.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity FIFO8TODSP32 is
-- generic parameters
-- port list
        port(   reset           : in std_logic;
                clk             : in std_logic;
                -- Read 8-bit data from FIFO
                fifo_rd         : out std_logic;
                fifo_dout       : in std_logic_vector(7 downto 0);
                fifo_empty      : in std_logic;
                -- Send 32-bit data to DSP
                data_rd         : in std_logic;
                data_ready      : out std_logic;
                data_out        : out std_logic_vector(31 downto 0)
               );
end FIFO8TODSP32;

Architecture rtl of FIFO8TODSP32 is
-- declarations
        signal rstate       : std_logic_vector(3 downto 0);
        constant rs00       : std_logic_vector(3 downto 0) := "0000";
        constant rs01       : std_logic_vector(3 downto 0) := "0001";
        constant rs02       : std_logic_vector(3 downto 0) := "0010";
        constant rs03       : std_logic_vector(3 downto 0) := "0101";
        constant rs04       : std_logic_vector(3 downto 0) := "0110";
        constant rs05       : std_logic_vector(3 downto 0) := "1101";
        constant rs06       : std_logic_vector(3 downto 0) := "1110";
        constant rs07       : std_logic_vector(3 downto 0) := "1001";
        constant rs08       : std_logic_vector(3 downto 0) := "1010";
        constant rs09       : std_logic_vector(3 downto 0) := "1000";
        signal rsdelay      : std_logic_vector(1 downto 0);
        signal dout         : std_logic_vector(31 downto 0);
begin
-- concurrent statements
        Read8to32StateProcess:
        Process(reset,clk)
        begin
                if reset='1' then
                        rstate  <=      rs00;
                elsif rising_edge(clk) then
                        case rstate is
                                when rs00 =>
                                        if fifo_empty='0' then
                                                rstate  <=      rs01;
                                        else
                                                rstate  <=      rs00;
                                        end if;

                                when rs01 =>
                                        if rsdelay(1)='1' then
                                                rstate  <=      rs02;
                                        else
                                                rstate  <=      rs01;
                                        end if;

                                when rs02 =>
```

```
                        if fifo_empty='0' then
                                rstate  <=      rs03;
                        else
                                rstate  <=      rs02;
                        end if;

                when rs03 =>
                        if rsdelay(1)='1' then
                                rstate  <=      rs04;
                        else
                                rstate  <=      rs03;
                        end if;

                when rs04 =>
                        if fifo_empty='0' then
                                rstate  <=      rs05;
                        else
                                rstate  <=      rs04;
                        end if;

                when rs05 =>
                        if rsdelay(1)='1' then
                                rstate  <=      rs06;
                        else
                                rstate  <=      rs05;
                        end if;

                when rs06 =>
                        if fifo_empty='0' then
                                rstate  <=      rs07;
                        else
                                rstate  <=      rs06;
                        end if;

                when rs07 =>
                        if rsdelay(1)='1' then
                                rstate  <=      rs08;
                        else
                                rstate  <=      rs07;
                        end if;

                when rs08 =>
                        if data_rd='1' then
                                rstate  <=      rs09;
                        else
                                rstate  <=      rs08;
                        end if;

                when rs09 =>
                        if data_rd='0' then
                                rstate  <=      rs00;
                        else
                                rstate  <=      rs09;
                        end if;

                when others =>
                        rstate  <=      rs00;
        end case;
    end if;
end process Read8to32StateProcess;

ReadDelayProcess:
Process(clk)
```

```
	begin
		if rising_edge(clk) then
			rsdelay(0)		<=	rstate(0);
			rsdelay(1)		<=	rsdelay(0) and rstate(0);
		end if;
	end process ReadDelayProcess;

	OutputProcess:
	Process(clk)
	begin
		if rising_edge(clk) then
			if rstate(3 downto 1)="001" then
				dout(7 downto 0)	<=	fifo_dout;
			else
				dout(7 downto 0)	<=	dout(7 downto 0);
			end if;

			if rstate(3 downto 1)="011" then
				dout(15 downto 8)	<=	fifo_dout;
			else
				dout(15 downto 8)	<=	dout(15 downto 8);
			end if;

			if rstate(3 downto 1)="111" then
				dout(23 downto 16)	<=	fifo_dout;
			else
				dout(23 downto 16)	<=	dout(23 downto 16);
			end if;

			if rstate(3 downto 1)="101" then
				dout(31 downto 24)	<=	fifo_dout;
			else
				dout(31 downto 24)	<=	dout(31 downto 24);
			end if;
		end if;
	end process OutputProcess;

	fifo_rd		<=	rstate(0) and not(rsdelay(1));
	data_ready	<=	'1' when rstate=rs08 else '0';
	data_out	<=	dout;

end rtl;
```

ifdsp.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity IFDSP is
-- generic parameters
-- port list
        port(   -- Inner XilinX Side
                clk                 : in std_logic;
                locked              : in std_logic;
                reset               : out std_logic;
                dsp_bypass          :   out std_logic;
                -- DDLL
                ver_sync            : in std_logic;
                oddeven             : in std_logic;
                line_no    : in std_logic_vector(9 downto 0);
                 -- FIFO I/F
                 fifo_rst           : in std_logic;
                 fifo_rd            : out std_logic;
                 fifo_dout          : in std_logic_vector(7 downto 0);
                 fifo_empty         : in std_logic;
                 fifo_full          : in std_logic;
                 -- Filter Side
                 wi_wr              : out std_logic;
                 wi_in              : out std_logic_vector(7 downto 0);
                 select_wi          : out std_logic_vector(7 downto 0);
                 wi_out             : in std_logic_vector(7 downto 0);
                 -- A/D D/A Side
                 gcr_en             : out std_logic;
                 -- DSP Side
                 n_ce               : in std_logic;
                 n_oe               : in std_logic;
                 n_rd               : in std_logic;
                 n_wr               : in std_logic;
                 addr_bus           : in std_logic_vector(21 downto 2);
                 data_bus           : inout std_logic_vector(31 downto 0));
end IFDSP;

Architecture Behavior of IFDSP is
-- declarations
        signal addr_lat             : std_logic_vector(21 downto 2);
        signal datain_lat,dataout_lat  : std_logic_vector(31 downto 0);
        signal wr_lat,rd_lat        : std_logic;
        signal rst                  : std_logic_vector(2 downto 0);
        signal bypass               : std_logic;
   signal bypass_l                  : std_logic;
        signal f_rd                 : std_logic;
        signal f_wr                 : std_logic_vector(2 downto 0);
        signal gcr_en_l             : std_logic;
        signal select_wi_l          : std_logic_vector(7 downto 0);

        component FIFO8TODSP32
                -- LocalGenericClause
                -- LocalPortClause
                port(   reset               : in std_logic;
                        clk                 : in std_logic;
                        -- Read 8-bit data from FIFO
                        fifo_rd             : out std_logic;
                        fifo_dout           : in std_logic_vector(7 downto 0);
                        fifo_empty          : in std_logic;
```

```
                        -- Send 32-bit data to DSP
                        data_rd         : in std_logic;
                        data_ready      : out std_logic;
                        data_out        : out std_logic_vector(31 downto 0));
        end component;
        signal data32_ready     : std_logic;
        signal data32_out       : std_logic_vector(31 downto 0);

begin
        clk             =>      clk,
        fifo_rd         =>      fifo_rd,
        fifo_dout       =>      fifo_dout,
        fifo_empty      =>      fifo_empty,
        data_rd         =>      f_rd,
        data_ready      =>      data32_ready,
        data_out        =>      data32_out);

        RdWrProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        if n_wr='0' and n_ce='0' and n_rd='1' then
                                wr_lat  <=      '1';
                        else
                                wr_lat  <=      '0';
                        end if;

                        if n_rd='0' and n_ce='0' and n_wr='1' then
                                rd_lat  <=      '1';
                        else
                                rd_lat  <=      '0';
                        end if;
                end if;
        end process RdWrProcess;

        AddrDataProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        if n_ce='0' then
                                addr_lat  <=    addr_bus;
                                datain_lat<=    data_bus;
                        else
                                addr_lat  <=    addr_lat;
                                datain_lat<=    datain_lat;
                        end if;
                end if;
        end process AddrDataProcess;

        RstProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        rst(2)  <=      rst(1);
                        rst(1)  <=      rst(0);
                        if addr_lat="0000000000000000000" and wr_lat='1' then
                                rst(0)  <=      '1';
                        else
                                rst(0)  <=      '0';
                        end if;
                        if addr_lat="0000000000000000001" and wr_lat='1' then
                                 bypass  <=  '1';
                        else
```

```
                        bypass  <=  '0';
                end if;

                if (bypass='1' and wr_lat='1') then
                        bypass_l <=     datain_lat(0);
                else
                        bypass_l <=     bypass_l;
                end if;
        end if;
end process RstProcess;

WrProcess:
Process(clk)
begin
        if rising_edge(clk) then
                f_wr(2)  <=     f_wr(1);
                f_wr(1)  <=     f_wr(0);
                if addr_lat="0000000000000001000" and wr_lat='1' then
                        f_wr(0)  <=     '1';
                else
                        f_wr(0)  <=     '0';
                end if;

                if addr_lat="0000000000000001001" and wr_lat='1' then
                        select_wi_l     <=      datain_lat(7 downto 0);
                else
                        select_wi_l     <=      select_wi_l;
                end if;
        end if;
end process WrProcess;

RdProcess:
Process(clk)
begin
        if rising_edge(clk) then
                if addr_lat="0000000000000000100" and rd_lat='1' then
                        f_rd    <=      '1';
                else
                        f_rd    <=      '0';
                end if;
        end if;
end process RdProcess;

GCRProcess:
Process(clk)
begin
        if rising_edge(clk) then
                if addr_lat="0000000000000000101" and wr_lat='1' then
                        gcr_en_l <=     datain_lat(0);
                else
                        gcr_en_l <=     gcr_en_l;
                end if;
        end if;
end process GCRProcess;

DataOutProcess:
Process(clk)
begin
        if rising_edge(clk) then
                case addr_lat is
                        when "0000000000000000000"      =>
                                dataout_lat(31)         <=  locked;
                                dataout_lat(30 downto 24)       <= "1110000";
```

```
                                        dataout_lat(23 downto 16)    <= "11000011";
                                        dataout_lat(15 downto  8)    <= "10100101";
                                        dataout_lat( 7 downto  0)    <= "10010110";
                                    when "0000000000000000001"       =>
                                        dataout_lat(31 downto 24)    <= "00000000";
                                        dataout_lat(23 downto 16)    <= "00000000";
                                        dataout_lat(15 downto 12)    <= "0000";
                                        dataout_lat(11)              <=  ver_sync;
                                        dataout_lat(10)              <=  oddeven;
                                        dataout_lat( 9 downto  0)    <= line_no;
                                    when "0000000000000000100"       =>
                                        dataout_lat(31 downto  0)    <= data32_out;
                                    when "0000000000000000101"       =>
                                        dataout_lat(31 downto 24)    <= "00000000";
                                        dataout_lat(23 downto 16)    <= "00000000";
                                        dataout_lat(15 downto  8)    <= "00000000";
                                        dataout_lat( 7 downto  3)    <= "00000";
                                        dataout_lat(2)               <= data32_ready;
                                        dataout_lat(1)               <= fifo_full;
                                        dataout_lat(0)               <= fifo_empty;
                                    when "0000000000000001001"       =>
                                        dataout_lat(31 downto 24)    <= "00000000";
                                        dataout_lat(23 downto 16)    <= "00000000";
                                        dataout_lat(15 downto  8)    <= "00000000";
                                        dataout_lat( 7 downto  0)    <= wi_out;
                                    when others =>
                                        dataout_lat(31 downto 24)    <= "00000000";
                                        dataout_lat(23 downto 16)    <= "00000000";
                                        dataout_lat(15 downto  8)    <= "00000000";
                                        dataout_lat( 7 downto  0)    <= "00000000";
                                end case;
                            end if;
                    end process DataOutProcess;

                    data_bus  <= dataout_lat when (n_ce='0' and n_oe='0') else
"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

                    reset           <= '1' when (rst(0)='1' or rst(1)='1') and rst(2)='0' else '0';
                    dsp_bypass      <= bypass_l;
                    wi_wr           <='1' when (f_wr(0)='1' or f_wr(1)='1') and f_wr(2)='0' else '0';
                    wi_in           <= datain_lat(7 downto 0);
                    select_wi       <= select_wi_l;
                    gcr_en          <= gcr_en_l;

end Behavior;
```

ddll.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity DDLL is
-- generic parameters
-- port list
        port(
                clk             : in std_logic;
                comp_sync       : in std_logic;
                line_pulse      : out std_logic;
                line_no         : out std_logic_vector(9 downto 0);
                oddeven         : out std_logic;
                ver_sync        : out std_logic;
                 lost_sync      : out std_logic);
end DDLL;

Architecture Behevioral of DDLL is
-- declarations
        signal comp_sync_1          : std_logic_vector(1 downto 0);
        signal fall_edge,fall_edge1 : std_logic;
        signal l_pulse              : std_logic;
        signal div24r               : unsigned(5 downto 0);
        signal div64r               : unsigned(5 downto 0);
        signal clr1,clr2,clr3       : std_logic;
         signal lost_sync_d         : unsigned(2 downto 0);

        signal c_cnt                : unsigned(3 downto 0);
        signal c_cnt_full           : std_logic;
        signal c_cnt_empty          : std_logic;
        signal v_sync,v_sync_1      : std_logic;
        signal l_no                 : unsigned(9 downto 0);
        signal l_no_625             : std_logic;
        signal oe_toggle            : std_logic;
        signal oe_latch             : std_logic;
        signal oe_latch_1           : std_logic;
        signal hl_cnt               : unsigned(6 downto 0);
        signal halfline             : std_logic;
        signal fullline             : std_logic;
begin
-- concurrent statements
        LinePulseProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        comp_sync_1(0)    <= comp_sync;
                        comp_sync_1(1)    <= comp_sync_1(0);

                        if comp_sync_1="10" then
                                fall_edge <= '1';
                        else
                                fall_edge <= '0';
                        end if;
                        fall_edge1 <= fall_edge;
                end if;
        end process LinePulseProcess;

        DivProcess:
        Process(clk)
        begin
```

```
if rising_edge(clk) then
        if clr1='1' or clr2='1' or clr3='1' then
                div24r    <="000000";
        else
                div24r    <= div24r+1;
        end if;

        if clr2='1' then
                div64r    <= "000000";
        elsif clr3='1' then
                div64r    <= "000001";
        elsif clr1='1' then
                div64r    <= div64r+1;
        else
                div64r    <= div64r;
        end if;

        if (div64r/=63 and div24r=22) then
                clr1      <= '1';
        else
                clr1      <= '0';
        end if;

        if (div64r=63 and fall_edge='1') then
                clr2      <= '1';
        else
                clr2      <= '0';
        end if;

        if (div64r=63 and div24r=46 and lost_sync_d(2)='0') then
                clr3      <= '1';
        else
                clr3      <= '0';
        end if;

--      if (div64r=63 and fall_edge='1') then
                if clr2='1' then
                        lost_sync_d <=  "000";
                elsif clr3='1' then
                        lost_sync_d <=  lost_sync_d+1;
                else
                        lost_sync_d <=  lost_sync_d;
                end if;

                l_pulse   <=      clr2 or clr3;

                if fall_edge='1' then
                        hl_cnt    <= "0000000";
                elsif clr1='1' then
                        hl_cnt    <= hl_cnt+1;
                else
                        hl_cnt    <= hl_cnt;
                end if;

                if fall_edge='1' then
                        if (hl_cnt(6 downto 4)=1 or hl_cnt(6 downto 4)=2) then
                                halfline  <= '1';
                        else
                                halfline  <= '0';
                        end if;
                else
                        halfline  <= halfline;
                end if;
```

```
                        if fall_edge='1' then
                                if hl_cnt(5 downto 4)=3 or hl_cnt(6 downto 4)=4 then
                                        fullline    <= '1';
                                else
                                        fullline    <= '0';
                                end if;
                        else
                                halfline    <= halfline;
                        end if;

                        if fall_edge1='1' then
                                if fullline='1' then
                                        oe_toggle<= '1';
                                elsif halfline='1' then
                                        oe_toggle<= not oe_toggle;
                                else
                                        oe_toggle<= oe_toggle;
                                end if;
                        else
                                oe_toggle<= oe_toggle;
                        end if;

                if v_sync='0' and v_sync_l='1' then
                        oe_latch  <= oe_toggle;
                else
                        oe_latch  <= oe_latch;
                end if;
                oe_latch_1 <= oe_latch;
        end if;
end process DivProcess;

CountProcess:
Process(clk)
begin
        if rising_edge(clk) then
                if clr1='1' or clr2='1' then
                        if comp_sync_l(0)='1' then
                                if c_cnt_full='0' then
                                        c_cnt       <=      c_cnt+1;
                                else
                                        c_cnt       <=      c_cnt;
                                end if;
                        else
                                if c_cnt_empty='0' then
                                        c_cnt       <=      c_cnt-1;
                                else
                                        c_cnt       <=      c_cnt;
                                end if;
                        end if;
                else
                        c_cnt       <=      c_cnt;
                end if;

                if c_cnt=15 then
                        c_cnt_full          <=      '1';
                else
                        c_cnt_full          <=      '0';
                end if;

                if c_cnt=0 then
                        c_cnt_empty         <=      '1';
                else
```

```
                    c_cnt_empty     <=      '0';
                end if;

                if fall_edge='1' then
                    v_sync  <=      std_logic(c_cnt(3));
                else
                    v_sync  <=      v_sync;
                end if;
                v_sync_l <=     v_sync;

                if l_no=625 then
                    l_no_625 <=     '1';
                else
                    l_no_625 <=     '0';
                end if;

                if (oe_latch='0' and oe_latch_l='1') or (l_no_625='1' and l_pulse='1') then -- Even
                    l_no    <=      "0000000000";
                elsif l_pulse='1' then
                    l_no    <=      l_no+1;
                else
                    l_no    <=      l_no;
                end if;
            end if;
        end process CountProcess;

        line_pulse      <=      l_pulse;
        ver_sync        <=      v_sync;
        oddeven         <=      oe_latch;
        line_no         <=      std_logic_vector(l_no);
    lost_sync   <=  lost_sync_d(2);

end Behevioral;
```

capturegcr.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity CAPTUREGCR is
-- generic parameters
-- port list
	port(
		reset		: in std_logic;
		clk		: in std_logic;
		-- Sync
		line_pulse	: in std_logic;
		line_no		: in std_logic_vector(9 downto 0);
		-- GCR
		gcr_en		: in std_logic;
		gcr_rst		: out std_logic;
		gcr_wr		: out std_logic);
end CAPTUREGCR;

Architecture rtl of CAPTUREGCR is
-- declarations
	type gcr_state is (s0,s1,s2,s3);
	signal s_gcr		: gcr_state;

	signal cnt_delay	: unsigned(8 downto 0);
	signal cnt_wr		: unsigned(10 downto 0);
	signal delay_to_gcr	: std_logic;
	signal cnt_wr_1023	: std_logic;
begin
-- concurrent statements
	StateProcess:
	Process(reset,clk)
	begin
		if reset='1' then
			s_gcr	<=	s0;
		elsif rising_edge(clk) then
			case s_gcr is
				when s0	=>
					if gcr_en='1' and line_no="0000001101" then
						s_gcr	<=	s1;
					else
						s_gcr	<=	s0;
					end if;

				when s1	=>
					if line_pulse='1' then
						s_gcr	<=	s2;
					else
						s_gcr	<=	s1;
					end if;

				when s2	=>
					if delay_to_gcr='1' then
						s_gcr	<=	s3;
					else
						s_gcr	<=	s2;
					end if;

				when s3	=>
					if cnt_wr_1023='1' then
```

```
                                        s_gcr   <=      s0;
                                    else
                                        s_gcr   <=      s3;
                                    end if;

                        end case;
                end if;
        end process StateProcess;

        CountProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        if s_gcr=s3 then
                                cnt_wr  <=      cnt_wr+1;
                        else
                                cnt_wr  <=      "00000000000";
                        end if;

                        if cnt_wr=1278 then
                                cnt_wr_1023     <=      '1';
                        else
                                cnt_wr_1023     <=      '0';
                        end if;

                        if s_gcr=s2 then
                                cnt_delay       <=      cnt_delay+1;
                        else
                                cnt_delay       <=      "000000000";
                        end if;

                        if cnt_delay=21 then
                                delay_to_gcr    <=      '1';
                        else
                                delay_to_gcr    <=      '0';
                        end if;
                end if;
        end process CountProcess;

        gcr_wr  <=      '1' when s_gcr=s3 else '0';
        gcr_rst <=      '1' when s_gcr=s1 else '0';

end rtl;
```

ifadda.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
Entity IFADDA is
-- generic parameters
-- port list
        port(-- Inner Side
                clk4x           : in std_logic;
                adc_clk         : in std_logic;
                adc_data        : out std_logic_vector(7 downto 0);
                dac_data        : in std_logic_vector(7 downto 0);
                comp_sync       : out std_logic;
                filter_sync     : out std_logic;
                -- A/D D/A Side
                adc_data_p      : in std_logic_vector(9 downto 2);
                dac_data_p      : out std_logic_vector(9 downto 2);
                comp_sync_p     : in std_logic);
end IFADDA;
Architecture Behavior of IFADDA is
-- declarations
        signal adc_data_lat         :std_logic_vector(9 downto 2);
        signal adc_exist, logic1    : std_logic;
        signal adc_sync             : std_logic_vector(2 downto 0);
        signal adc_data_sync        : std_logic_vector(7 downto 0);
begin
-- concurrent statements
        logic1  <=      '1';
        Clk12Process:
        Process(adc_clk)
        begin
                if rising_edge(adc_clk) then
                        adc_data_lat    <=      adc_data_p;
                end if;
        end process Clk12Process;
        Clk12Process1:
        Process(adc_sync,adc_clk)
        begin
                if adc_sync(1)='1' then
                        adc_exist <=    '0';
                elsif rising_edge(adc_clk) then
                        adc_exist <=    logic1;
                end if;
        end process Clk12Process1;
        Clk96Process:
        Process(clk4x)
        begin
                if rising_edge(clk4x) then
                        adc_sync(0)     <=      adc_exist;
                        adc_sync(1)     <=      adc_sync(0);
                        adc_sync(2)     <=      adc_sync(1);
                        filter_sync     <=      adc_sync(2);
                        if adc_sync(1 downto 0)="01" then
                                adc_data_sync   <=      adc_data_lat(9 downto 2);
                        end if;
                end if;
        end process Clk96Process;
        adc_data <=     adc_data_sync;
        comp_sync       <=      comp_sync_p;
        dac_data_p      <=      dac_data;
end Behavior;
```

fir.vhd

```
Library   ieee;
use       ieee.std_logic_1164.all;
use       ieee.std_logic_arith.all;

Entity FIR is
-- generic parameters
-- port    list
          port(     reset              : in std_logic;
                    clk4x              : in std_logic;
                    -- Wi
                    wi_wr              : in std_logic;
                    wi_in              : in std_logic_vector(7 downto 0);
                    select_wi          : in std_logic_vector(7 downto 0);
                    wi_out             : out std_logic_vector(7 downto 0);
                    -- Xi
                    xi_in              : in std_logic_vector(7 downto 0);
                    xi_out             : out std_logic_vector(7 downto 0);
                    xi_delay_16        : out std_logic_vector(7 downto 0);
                    -- Result
                    sum                : out std_logic_vector(7 downto 0));
end FIR;

Architecture structural of FIR is
-- declarations
          component MAC8
                    -- LocalGenericClause
                    -- LocalPortClause
                    port(     sclr     :     in std_logic;
                              clk      :     in std_logic;
                              w_wr     :     in std_logic;
                              w_in     :     in std_logic_vector(7 downto 0);
                              x_in     :     in std_logic_vector(7 downto 0);
                              w_out    :     out std_logic_vector(7 downto 0);
                              x_out    :     out std_logic_vector(7 downto 0);
                              sum      :     out std_logic_vector(16 downto 0));
          end component;

          component ADD_S17S17
                    -- LocalGenericClause
                    -- LocalPortClause
                    port(     clk      :     in std_logic;
                              a        :     in std_logic_vector(16 downto 0);
                              b        :     in std_logic_vector(16 downto 0);
                              q        :     out std_logic_vector(16 downto 0));
          end component;

          component CNT3DEC8
                    -- LocalGenericClause
                    -- LocalPortClause
                    port(     clk      :     in std_logic;
                              sclr     :     in std_logic;
                              select_wi :    in std_logic_vector(2 downto 0);
                              x_en     :     out std_logic;
                              wi_en1   :     out std_logic;
                              wi_en2   :     out std_logic;
                              sum_en   :     out std_logic;
                              state    :     out std_logic_vector(2 downto 0));
          end component;

          component MUX8X4TO1CE
```

```
		port(	ce	:	in std_logic;
			clk	:	in std_logic;
			s	:	in std_logic_vector(1 downto 0);
			ma	:	in std_logic_vector(7 downto 0);
			mb	:	in std_logic_vector(7 downto 0);
			mc	:	in std_logic_vector(7 downto 0);
			md	:	in std_logic_vector(7 downto 0);
			q	:	out std_logic_vector(7 downto 0));
	end component;

	component MUX8X8TO1CE
		port(	ce	:	in std_logic;
			clk	:	in std_logic;
			s	:	in std_logic_vector(2 downto 0);
			ma	:	in std_logic_vector(7 downto 0);
			mb	:	in std_logic_vector(7 downto 0);
			mc	:	in std_logic_vector(7 downto 0);
			md	:	in std_logic_vector(7 downto 0);
			me	:	in std_logic_vector(7 downto 0);
			mf	:	in std_logic_vector(7 downto 0);
			mg	:	in std_logic_vector(7 downto 0);
			mh	:	in std_logic_vector(7 downto 0);
			q	:	out std_logic_vector(7 downto 0));
	end component;

	subtype	std8 is	std_logic_vector(7 downto 0);
	type arr32_std8	is array(31 downto 0) of std8;
	type arr8_std8 is array(7 downto 0) of std8;
	subtype	std17 is std_logic_vector(16 downto 0);
	type arr32_std17 is array(31 downto 0) of std17;
	type arr16_std17 is array(15 downto 0) of std17;
	type arr8_std17	is array(7 downto 0) of std17;
	type arr4_std17	is array(3 downto 0) of std17;
	type arr2_std17	is array(1 downto 0) of std17;
	signal x_in_i	:	arr32_std8;
	signal x_out_i	:	arr32_std8;
	signal xi_delay_16_l	:	std8;
	signal w_in_i	:	arr32_std8;
	signal w_out_i	:	arr32_std8;
	signal sum_i	:	arr32_std17;
	signal sum16_i	:	arr16_std17;
	signal sum8_i	:	arr8_std17;
	signal sum4_i	:	arr4_std17;
	signal sum2_i	:	arr2_std17;
	signal sum1_i	:	std17;
	signal sum_out	:	std8;
	signal x_en	:	std_logic;
	signal sum_en	:	std_logic;
	signal state	:	std_logic_vector(2 downto 0);
	signal w_out_l	:	std_logic_vector(7 downto 0);
	signal x_out_l	:	std_logic_vector(7 downto 0);
	signal w_mux_i	:	arr8_std8;
	signal w_en1	:	std_logic;
	signal w_en2	:	std_logic;
begin
-- concurrent statements

	CNT3DEC8_1 : CNT3DEC8 port map(clk		=>	clk4x,
					sclr		=>	reset,
					select_wi	=>	select_wi(2 downto 0),
					x_en		=>	x_en,
					wi_en1		=>	w_en1,
					wi_en2		=>	w_en2,
```

```
                                    sum_en          =>      sum_en,
                                    state           =>      state);
x_in_i(0)           <=      xi_in;
w_in_i(0)           <=      wi_in;

Process(x_out_i,w_out_i)
begin
        for i in 31downto 1 loop
                x_in_i(i) <= x_out_i(i-1);
                w_in_i(i) <= w_out_i(i-1);
        end loop;
end process;

FIR16: For i in 31 downto 0 generate
        FIR16_1  : MAC8 port map(  sclr    =>      reset,
                                   clk     =>      clk4x,
                                   w_wr    =>      wi_wr,
                                   w_in    =>      w_in_i(i),
                                   x_in    =>      x_in_i(i),
                                   w_out   =>      w_out_i(i),
                                   x_out   =>      x_out_i(i),
                                   sum     =>      sum_i(i));
                        end generate;
ADD16:  for i in 15 downto 0 generate
        ADD16_1: ADD_S17S17 port map(   clk     =>      clk4x,
                                        a       =>      sum_i(2*i),
                                        b       =>      sum_i(2*i+1),
                                        q       =>      sum16_i(i));
        end generate;
ADD8: for i in 7 downto 0 generate
        ADD8_1: ADD_S17S17 port map(    clk     =>      clk4x,
                                        a       =>      sum16_i(2*i),
                                        b       =>      sum16_i(2*i+1),
                                        q       =>      sum8_i(i));
        end generate;
ADD4: for i in 3 downto 0 generate
        ADD4_1 : ADD_S17S17 port map(   clk     =>      clk4x,
                                        a       =>      sum8_i(2*i),
                                        b       =>      sum8_i(2*i+1),
                                        q       =>      sum4_i(i));
        end generate;
ADD2: for i in 1 downto 0 generate
        ADD2_1: ADD_S17S17 port map(    clk     =>      clk4x,
                                        a       =>      sum4_i(2*i),
                                        b       =>      sum4_i(2*i+1),
                                        q       =>      sum2_i(i));
        end enerate;
ADD1 : ADD_S17S17 port map(             clk     =>      clk4x,
                                        a       =>      sum2_i(0),
                                        b       =>      sum2_i(1),
                                        q       =>      sum1_i);
SumProcess:
Process(clk4x)
begin
        if rising_edge(clk4x) then
                if sum_en='1' then
                        if sum1_i(16)='1' then
                                sum_out  <=     "00000000";
                        elsif sum1_i(15)='1' then
                                sum_out  <=     "11111111";
                        else
                                sum_out  <=     sum1_i(14 downto 7);
                        end     if;
```

```
                else
                        sum_out  <=      sum_out;
                end if;
        end if;
end process SumProcess;

MUX4: for i in 7 downto 0 generate
        MUX4_1 : MUX8X4TO1CE port map(ce      =>      w_en1,
                                      clk     =>      clk4x,
                                      s       =>      select_wi(4 downto 3),
                                      ma      =>      w_out_i(4*i),
                                      mb      =>      w_out_i(4*i+1),
                                      mc      =>      w_out_i(4*i+2),
                                      md      =>      w_out_i(4*i+3),
                                      q       =>      w_mux_i(i));
        end generate;

MUX8X8TO1_1    : MUX8X8TO1CE port map( ce      =>      w_en2,
                                      clk     =>      clk4x,
                                      s       =>      select_wi(7 downto 5),
                                      ma      =>      w_mux_i(0),
                                      mb      =>      w_mux_i(1),
                                      mc      =>      w_mux_i(2),
                                      md      =>      w_mux_i(3),
                                      me      =>      w_mux_i(4),
                                      mf      =>      w_mux_i(5),
                                      mg      =>      w_mux_i(6),
                                      mh      =>      w_mux_i(7),
                                      q       =>      w_out_l );

ReadWiProcess:
Process(clk4x)
begin
        if rising_edge(clk4x) then
                if x_en='1' then
                        x_out_l   <=      x_out_i(31);
                else
                        x_out_l   <=      x_out_l;
                end if;

                if x_en='1' then
                        xi_delay_16_l     <=      x_out_i(1);
                else
                        xi_delay_16_l     <=      xi_delay_16_l;
                end if;
        end if;
end process ReadWiProcess;

wi_out    <=      w_out_l;
xi_out    <=      x_out_l;
sum       <=      sum_out;
xi_delay_16       <=      xi_delay_16_l;

end structural;
```

testled.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use work.P_GHOST.all;

Entity TESTLED is
        port(   reset   :       in std_logic;
                clk     :       in std_logic;
                clkdv   :       in std_logic;
                clk2x   :       in std_logic;
                clk4x   :       in std_logic;
                locked  :       in std_logic;
                LED     :       out std_logic_vector(3 downto 0));
end TESTLED;

Architecture Behavior of TESTLED is
        signal Cnt4ba   :       unsigned(3 downto 0);
        signal div24a   :       unsigned(4 downto 0);
        signal div1ma   :       unsigned(15 downto 0);
        signal Cnt4bb   :       unsigned(3 downto 0);
        signal div24b   :       unsigned(4 downto 0);
        signal div1mb   :       unsigned(15 downto 0);
        signal Cnt4bc   :       unsigned(3 downto 0);
        signal div24c   :       unsigned(4 downto 0);
        signal div1mc   :       unsigned(15 downto 0);
        signal Cnt4bd   :       unsigned(3 downto 0);
        signal div24d   :       unsigned(4 downto 0);
        signal div1md   :       unsigned(15 downto 0);
begin
        aDivProcess:
        Process(clk)
        begin
                if rising_edge(clk) then
                        if reset='1' then
                                div24a  <=      "00000";
                                div1ma  <=      "0000000000000000";
                        elsif div24a=31 then
                                div24a  <=      "00000";
                                div1ma  <=      div1ma+1;
                        else
                                div24a  <=      div24a+1;
                                div1ma  <=      div1ma;
                        end if;
                        if reset='1' then
                                Cnt4ba  <=      "0000";
                        elsif (div1ma=65535 and div24a=31) then
                                Cnt4ba  <=      Cnt4ba+1;
                        end if;
                end if;
        end process aDivProcess;

        bDivProcess:
        Process(clkdv)
        begin
                if rising_edge(clkdv) then
                        if reset='1' then
                                div24b  <=      "00000";
                                div1mb  <=      "0000000000000000";
                        elsif div24b=31 then
                                div24b  <=      "00000";
```

```
                        div1mb  <=      div1mb+1;
                    else
                        div24b  <=      div24b+1;
                        div1mb  <=      div1mb;
                    end if;
                    if reset='1' then
                        Cnt4bb  <=      "0000";
                    elsif (div1mb=65535 and div24b=31) then
                        Cnt4bb  <=      Cnt4bb+1;
                    end if;
                end if;
        end process bDivProcess;

        cDivProcess:
        Process(clk2x)
        begin
                if rising_edge(clk2x) then
                    if reset='1' then
                        div24c  <=      "00000";
                        div1mc  <=      "0000000000000000";
                    elsif div24c=31 then
                        div24c  <=      "00000";
                        div1mc  <=      div1mc+1;
                    else
                        div24c  <=      div24c+1;
                        div1mc  <=      div1mc;
                    end if;

                    if reset='1' then
                        Cnt4bc  <=      "0000";
                    elsif (div1mc=65535 and div24c=31) then
                        Cnt4bc  <=      Cnt4bc+1;
                    end if;
                end if;
        end process cDivProcess;

        dDivProcess:
        Process(clk4x)
        begin
                if rising_edge(clk4x) then
                    if reset='1' then
                        div24d  <=      "00000";
                        div1md  <=      "0000000000000000";
                    elsif div24d=31 then
                        div24d  <=      "00000";
                        div1md  <=      div1md+1;
                    else
                        div24d  <=      div24d+1;
                        div1md  <=      div1md;
                    end if;

                    if reset='1' then
                        Cnt4bd  <=      "0000";
                    elsif (div1md=65535 and div24d=31) then
                        Cnt4bd  <=      Cnt4bd+1;
                    end if;
                end if;
        end process dDivProcess;
        LED(3)  <=      locked;
        LED(2)  <=      Cnt4bd(3);
        LED(1)  <=      Cnt4ba(3);
        LED(0)  <=      Cnt4bb(3);
end Behavior;
```

main.vhd

```
Library   ieee;
use       ieee.std_logic_1164.all;
use       ieee.std_logic_arith.all;
use       work.p_ghost.all;

Entity MAIN       is
-- generic parameters
-- port    list
          port(     clk_p               :         in std_logic;
                    -- DSP Side
                    n_ce                :         in std_logic;
                    n_oe                :         in std_logic;
                    n_rd                :         in std_logic;
                    n_wr                :         in std_logic;
                    addr_bus            :         in std_logic_vector(21 downto 2);
                    data_bus            :         inout std_logic_vector(31 downto 0);
                    -- A/D D/A Side
                    clkdv_p             :         out std_logic;
                    adc_clk_p           :         in std_logic;
                    adc_data_p          :         in std_logic_vector(9 downto 2);
                    dac_data_p          :         out std_logic_vector(9 downto 2);
                    comp_sync_p         :         in std_logic;
                    -- 7-Segment Side
                    n_sw2               :         in std_logic_vector(7 downto 0);
                    n_SW                :         in std_logic_vector(3 downto 0);
                    n_LED               :         out std_logic_vector(3 downto 0);
                    SEG_EN              :         out std_logic_vector(3 downto 0);
                    n_SEG               :         out std_logic_vector(7 downto 0);
                    -- Debug
                    debug               :         out std_logic_vector(5 downto 0));
end       MAIN;

Architecture Structural       of MAIN is
-- declarations

          signal HEX          :         HEX4;
          signal DOT          :         std_logic_vector(3 downto 0);
          signal LED          :         std_logic_vector(3 downto 0);
          signal SW           :         std_logic_vector(3 downto 0);
          signal sw2          :         std_logic_vector(7 downto 0);

          component SHOW7SEG
                    port(     clk       :         in std_logic;
                              -- In Chip
                              HEX       :         in HEX4;
                              DOT       :         in std_logic_vector(3 downto 0);
                              LED       :         in std_logic_vector(3 downto 0);
                              SW        :         out std_logic_vector(3 downto 0);
                              sw2       :         out std_logic_vector(7 downto 0);
                              -- Out Chip
                              SEG_EN:             out std_logic_vector(3 downto 0);
                              n_SEG     :         out std_logic_vector(7 downto 0);     -- ABCEDFG.
                              n_LED     :         out std_logic_vector(3 downto 0);
                              n_SW      :         in std_logic_vector(3 downto 0);
                              n_sw2     :         in std_logic_vector(7 downto 0));
          end component;

          signal clk,clk2x,clk4x,locked,locked1,locked2     :         std_logic;
          signal adc_clk                    :         std_logic;
```

```
component DIV2_MUL4
        port (      CLKIN    : in  std_logic;
                    RESET    : in  std_logic;
                    CLK      : out std_logic;
                    CLKDV    : out std_logic;
                    CLK2X    : out std_logic;
                    CLK4X    : out std_logic;
                    LOCKED1 : out std_logic;
                    LOCKED2 : out std_logic;
                    adc_clk_p: in std_logic;
                    adc_clk    : out std_logic);
end component;

component FIFO1024X8DC
        -- LocalGenericClause
        -- LocalPortClause
        port(       sinit       :       in std_logic;
                    clk         :       in std_logic;
                    rd_en       :       in std_logic;
                    wr_en       :       in std_logic;
                    din         :       in std_logic_vector(7 downto 0);
                    dout        :       out std_logic_vector(7 downto 0);
                    full        :       out std_logic;
                    empty       :       out std_logic;
                    data_count:         out std_logic_vector(9 downto 0));
end component;

signal dsp_bypass   :       std_logic;
signal reset,fifo_rd,fifo_full,fifo_empty :         std_logic;
signal fifo_rst     :       std_logic;
signal fifo_dout    :       std_logic_vector(7 downto 0);
signal fifo_n       :       std_logic_vector(9 downto 0);

component IFDSP
        port(--     Inner XilinX Side
                    clk                 :       in std_logic;
                    locked              :       in std_logic;
                    reset               :       out std_logic;
                    dsp_bypass          :       out std_logic;
                    -- DDLL
                    ver_sync            :       in std_logic;
                    oddeven             :       in std_logic;
                    line_no             :       in std_logic_vector(9 downto 0);
                    -- FIFO I/F
                    fifo_rst            :       in std_logic;
                    fifo_rd             :       out     std_logic;
                    fifo_dout           :       in std_logic_vector(7 downto 0);
                    fifo_empty          :       in std_logic;
                    fifo_full           :       in std_logic;
                    -- Filter Side
                    wi_wr               :       out std_logic;
                    wi_in               :       out std_logic_vector(7 downto 0);
                    select_wi           :       out std_logic_vector(7 downto 0);
                    wi_out              :       in std_logic_vector(7 downto 0);
                    -- A/D D/A Side
                    gcr_en              :       out std_logic;
                    -- DSP Side
                    n_ce                :       in std_logic;
                    n_oe                :       in std_logic;
                    n_rd                :       in std_logic;
                    n_wr                :       in std_logic;
                    addr_bus            :       in std_logic_vector(21 downto 2);
                    data_bus            :       inout std_logic_vector(31 downto 0));
```

```
end     component;
component FIR
        -- LocalGenericClause
        -- LocalPortClause
        port(
                reset           : in std_logic;
                clk4x           : in std_logic;
                -- Wi
                wi_wr           : in std_logic;
                wi_in           : in std_logic_vector(7 downto 0);
                select_wi : in std_logic_vector(7 downto 0);
                wi_out          : out std_logic_vector(7 downto 0);
                -- Xi
                xi_in           : in std_logic_vector(7 downto 0);
                xi_out          : out std_logic_vector(7 downto 0);
                xi_delay_16     : out std_logic_vector(7 downto 0);
                -- Result
                sum             : out std_logic_vector(7 downto 0));
end component;

signal fir_reset    :       std_logic;
signal xi_delay_16  :       std_logic_vector(7 downto 0);
signal wi_wr        :       std_logic;
signal wi_in        :       std_logic_vector(7 downto 0);
signal wi_out       :       std_logic_vector(7 downto 0);
signal select_wi    :       std_logic_vector(7 downto 0);
signal dsp_sel_wi   :       std_logic_vector(7 downto 0);
signal xi_out       :       std_logic_vector(7 downto 0);
signal filter_sum   :       std_logic_vector(7 downto 0);

component IFADDA
        port(--     Inner Side
                    clk4x           : in std_logic;
                    adc_clk         : in std_logic;
                    adc_data        : out std_logic_vector(7 downto 0);
                    dac_data        : in std_logic_vector(7 downto 0);
                    comp_sync       : out std_logic;
                    filter_sync     : out std_logic;
                    -- A/D D/A Side
                    adc_data_p      : in std_logic_vector(9 downto 2);
                    dac_data_p      : out std_logic_vector(9 downto 2);
                    comp_sync_p     : in std_logic);
end component;

signal filter_sync          :       std_logic;
signal adc_data,dac_data    :       std_logic_vector(7 downto 0);
signal gcr_en,gcr_wr,gcr_rst :      std_logic;
signal comp_sync,line_pulse :       std_logic;
signal line_no              :       std_logic_vector(9 downto 0);
signal ver_sync,oddeven     :       std_logic;
signal lost_sync            :       std_logic;

component DDLL
        port(       clk             : in std_logic;
                    comp_sync       : in std_logic;
                    line_pulse      : out std_logic;
                    line_no         : out std_logic_vector(9 downto 0);
                    oddeven         : out std_logic;
                    ver_sync        : out std_logic;
                    lost_sync       : out std_logic
                    );
end component;
```

```
	component CAPTUREGCR
		port(	reset		: in std_logic;
			clk		: in std_logic;
			-- Sync
			line_pulse	: in std_logic;
			line_no		: in std_logic_vector(9 downto 0);
			-- GCR
			gcr_en		: in std_logic;
			gcr_rst		: out std_logic;
			gcr_wr		: out std_logic);
	end component;

	component TESTLED
		port(
			reset	:	in std_logic;
			clk	:	in std_logic;
			clkdv	:	in std_logic;
			clk2x	:	in std_logic;
			clk4x	:	in std_logic;
			locked	:	in std_logic;
			LED	:	out std_logic_vector(3 downto 0));
	end component;

begin
-- concurrent statements
	SHOW7SEG_1 : SHOW7SEG port map(	clk	=>	clk,
					HEX	=>	HEX,
					DOT	=>	DOT,
					LED	=>	LED,
					SW	=>	SW,
					sw2	=>	sw2,
					SEG_EN	=>	SEG_EN,
					n_SEG	=>	n_SEG,
					n_LED	=>	n_LED,
					n_SW	=>	n_SW,
					n_sw2	=>	n_sw2);

	DIV2_MUL4_1	: DIV2_MUL4 port map(	CLKIN	=>	clk_p,
					RESET	=>	SW(3),
					CLK	=>	clk,
					CLKDV	=>	clkdv_p,
					CLK2X	=>	clk2x,
					CLK4X	=>	clk4x,
					LOCKED1		=>	locked1,
					LOCKED2		=>	locked2,
					adc_clk_p	=>	adc_clk_p,
					adc_clk	=>	adc_clk);
	locked	<=	locked1and locked2;
	fifo_rst	<=	reset or gcr_rst;

	FIFO_1 : FIFO1024X8DC port map(	sinit		=>	fifo_rst,
					clk		=>	adc_clk,
					rd_en		=>	fifo_rd,
					wr_en		=>	gcr_wr,
					din		=>	adc_data,
					dout		=>	fifo_dout,
					full		=>	fifo_full,
					empty		=>	fifo_empty,
					data_count	=>	fifo_n);
	IFDSP_1 : IFDSP	port map(-- Innet XilinX Side
				clk		=>	clk,
				locked		=>	locked,
				reset		=>	reset,
```

```
                        dsp_bypass      =>      dsp_bypass,
                        -- DDLL
                        ver_sync        =>      ver_sync,
                        oddeven         =>      oddeven,
                        line_no         =>      line_no,
                        -- FIFO   I/F
                        fifo_rst        =>      fifo_rst,
                        fifo_rd         =>      fifo_rd,
                        fifo_dout       =>      fifo_dout,
                        fifo_empty      =>      fifo_empty,
                        fifo_full       =>      fifo_full,
                        -- Filter Side
                        wi_wr           =>      wi_wr,
                        wi_in           =>      wi_in,
                        select_wi       =>      dsp_sel_wi,
                        wi_out          =>      wi_out,
                        -- A/D D/A Side
                        gcr_en          =>      gcr_en,
                        -- DSP Side
                        n_ce            =>      n_ce,
                        n_oe            =>      n_oe,
                        n_rd            =>      n_rd,
                        n_wr            =>      n_wr,
                        addr_bus        =>      addr_bus,
                        data_bus        =>      data_bus);

DDLL_1 : DDLL port map( clk             =>      clk,
                        comp_sync       =>      comp_sync,
                        line_pulse      =>      line_pulse,
                        line_no         =>      line_no,
                        oddeven         =>      oddeven,
                        ver_sync        =>      ver_sync,
                        lost_sync       =>      lost_sync);

debug(0) <=     lost_sync when sw(1)='1' else line_pulse;
debug(1) <=     ver_sync;
debug(2) <=     oddeven;
debug(3) <=     wi_wr when sw(1)='1' else dsp_bypass;
debug(4) <=     gcr_wr;
debug(5) <=     fifo_rd     when sw(1)='1' else gcr_en;

CAPTUREGCR_1 : CAPTUREGCR port map(     reset           =>      reset,
                                        clk             =>      clk,
                                        line_pulse      =>      line_pulse,
                                        line_no         =>      line_no,
                                        gcr_en          =>      gcr_en,
                                        gcr_rst         =>      gcr_rst,
                                        gcr_wr          =>      gcr_wr);

IFADDA_1 : IFADDA port map(     -- Inner Side
                                clk4x           =>      clk4x,
                                adc_clk         =>      adc_clk,
                                adc_data        =>      adc_data,
                                dac_data        =>      dac_data,
                                comp_sync       =>      comp_sync,
                                filter_sync     =>      filter_sync,
                                -- A/D D/A Side
                                adc_data_p      =>      adc_data_p,
                                dac_data_p      =>      dac_data_p,
                                comp_sync_p     =>      comp_sync_p);

dac_data  <= xi_delay_16 when (SW(2)='1' or dsp_bypass='1') else filter_sum;
fir_reset  <= reset and filter_sync;
```

```
        Process(clk)
        begin
                if rising_edge(clk)  then
                        if SW(0)='1' then
                                select_wi <=        sw2;
                        else
                                select_wi <=        dsp_sel_wi;
                        end if;
                end if;
        end process;

        FIR_1 :   FIR port map(       reset             =>      fir_reset,
                                      clk4x             =>      clk4x,
                                      -- Wi
                                      wi_wr             =>      wi_wr,
                                      wi_in             =>      wi_in,
                                      select_wi         =>      select_wi,
                                      wi_out            =>      wi_out,
                                      -- Xi
                                      xi_in             =>      adc_data,
                                      xi_out            =>      xi_out,
                                      xi_delay_16       =>      xi_delay_16,
                                      sum               =>      filter_sum);

        TESTLED_1 : TESTLED port map(     reset     =>      SW(3),
                                          clk       =>      clk,
                                          clkdv     =>      adc_clk,
                                          clk2x     =>      clk2x,
                                          clk4x     =>      clk4x,
                                          locked    =>      locked,
                                          LED       =>      LED);

        HEX(0)  <=      wi_out(3 downto 0);
        HEX(1)  <=      wi_out(7 downto 4);
        HEX(2)  <=      select_wi(3 downto 0);
        HEX(3)  <=      select_wi(7 downto 4);

        DOT     <=      "0000";
end Structural;
```

ภาคผนวก ง
โปรแกรมกำจัดผี

c6211dsk.h

```
/************************************************************************
* FILENAME
*   c6211dsk.h
*
* DESCRIPTION
*   DSK Header File
*
************************************************************************/

/* Register definitions for C6211 chip on DSK */

/* Define EMIF Registers  */
#define EMIF_GCR        0x1800000       /* Address of EMIF global control          */
#define EMIF_CE0        0x1800008       /* Address of EMIF CE0 control             */
#define EMIF_CE1        0x1800004       /* Address of EMIF CE1 control             */
#define EMIF_CE2        0x1800010       /* Address of EMIF SDRAM extension         */
#define EMIF_CE3        0x1800014       /* Address of EMIF SDRAM extension         */
#define EMIF_SDCTRL     0x1800018       /* Address of EMIF SDRAM control           */
#define EMIF_SDRP       0x180001c       /* Address of EMIF SDRM refresh period     */
#define EMIF_SDEXT      0x1800020       /* Address of EMIF SDRAM extension         */

/* Define McBSP0 Registers */
#define McBSP0_DRR      0x18c0000       /* Address of data receive reg.        */
#define McBSP0_DXR      0x18c0004       /* Address of data transmit reg.       */
#define McBSP0_SPCR     0x18c0008       /* Address of serial port contl. reg.  */
#define McBSP0_RCR      0x18c000C       /* Address of receive control reg.     */
#define McBSP0_XCR      0x18c0010       /* Address of transmit control reg.    */
#define McBSP0_SRGR     0x18c0014       /* Address of sample rate generator    */
#define McBSP0_MCR      0x18c0018       /* Address of multichannel reg.        */
#define McBSP0_RCER     0x18c001C       /* Address of receive channel enable.  */
#define McBSP0_XCER     0x18c0020       /* Address of transmit channel enable. */
#define McBSP0_PCR      0x18c0024       /* Address of pin control reg.         */

/* Define McBSP1 Registers */
#define McBSP1_DRR      0x1900000       /* Address of data receive reg.        */
#define McBSP1_DXR      0x1900004       /* Address of data transmit reg.       */
#define McBSP1_SPCR     0x1900008       /* Address of serial port contl. reg.  */
#define McBSP1_RCR      0x190000C       /* Address of receive control reg.     */
#define McBSP1_XCR      0x1900010       /* Address of transmit control reg.    */
#define McBSP1_SRGR     0x1900014       /* Address of sample rate generator    */
#define McBSP1_MCR      0x1900018       /* Address of multichannel reg.        */
#define McBSP1_RCER     0x190001C       /* Address of receive channel enable.  */
#define McBSP1_XCER     0x1900020       /* Address of transmit channel enable. */
#define McBSP1_PCR      0x1900024       /* Address of pin control reg.         */

/* Define L2 Cache Registers */
#define L2CFG           0x1840000       /* Address of L2 config reg            */
#define MAR0            0x1848200       /* Address of mem attribute reg        */

/* Define Interrupt Registers */
#define IMH             0x19c0000       /* Address of Interrupt Multiplexer High*/
#define IML             0x19c0004       /* Address of Interrupt Multiplexer Low */

/* Define Timer0 Registers */
```

```
#define TIMER0_CTRL     0x1940000       /* Address of timer0 control reg.       */
#define TIMER0_PRD      0x1940004       /* Address of timer0 period reg.        */
#define TIMER0_COUNT    0x1940008       /* Address of timer0 counter reg.       */

/* Define Timer1 Registers */
#define TIMER1_CTRL     0x1980000       /* Address of timer1 control reg.       */
#define TIMER1_PRD      0x1980004       /* Address of timer1 period reg.        */
#define TIMER1_COUNT    0x1980008       /* Address of timer1 counter reg        */

/* Define EDMA Registers */
#define PQSR            0x01A0FFE0      /* Address of priority queue status     */
#define CIPR            0x01A0FFE4      /* Address of channel interrupt pending*/
#define CIER            0x01A0FFE8      /* Address of channel interrupt enable  */
#define CCER            0x01A0FFEC      /* Address of channel chain enable      */
#define ER              0x01A0FFF0      /* Address of event register            */
#define EER             0x01A0FFF4      /* Address of event enable register     */
#define ECR             0x01A0FFF8      /* Address of event clear register      */
#define ESR             0x01A0FFFC      /* Address of event set register        */

/* Define EDMA Transfer Parameter Entry Fields */
#define OPT             0*4             /* Options Parameter        */
#define SRC             1*4             /* SRC Address Parameter    */
#define CNT             2*4             /* Count Parameter          */
#define DST             3*4             /* DST Address Parameter    */
#define IDX             4*4             /* IDX Parameter            */
#define LNK             5*4             /* LNK Parameter            */

/* Define EDMA Parameter RAM Addresses */
#define EVENT0_PARAMS 0x01A00000
#define EVENT1_PARAMS EVENT0_PARAMS + 0x18
#define EVENT2_PARAMS EVENT1_PARAMS + 0x18
#define EVENT3_PARAMS EVENT2_PARAMS + 0x18
#define EVENT4_PARAMS EVENT3_PARAMS + 0x18
#define EVENT5_PARAMS EVENT4_PARAMS + 0x18
#define EVENT6_PARAMS EVENT5_PARAMS + 0x18
#define EVENT7_PARAMS EVENT6_PARAMS + 0x18
#define EVENT8_PARAMS EVENT7_PARAMS + 0x18
#define EVENT9_PARAMS EVENT8_PARAMS + 0x18
#define EVENTA_PARAMS EVENT9_PARAMS + 0x18
#define EVENTB_PARAMS EVENTA_PARAMS + 0x18
#define EVENTC_PARAMS EVENTB_PARAMS + 0x18
#define EVENTD_PARAMS EVENTC_PARAMS + 0x18
#define EVENTE_PARAMS EVENTD_PARAMS + 0x18
#define EVENTF_PARAMS EVENTE_PARAMS + 0x18
#define EVENTN_PARAMS EVENTF_PARAMS + 0x18
#define EVENTO_PARAMS EVENTN_PARAMS + 0x18

/* Define QDMA Memory Mapped Registers */
#define QDMA_OPT        0x02000000      /* Address of QDMA options register     */
#define QDMA_SRC        0x02000004      /* Address of QDMA SRC address register */
#define QDMA_CNT        0x02000008      /* Address of QDMA counts register      */
#define QDMA_DST        0x0200000C      /* Address of QDMA DST address register */
#define QDMA_IDX        0x02000010      /* Address of QDMA index register       */

/* Define QDMA Pseudo Registers */
#define QDMA_S_OPT      0x02000020      /* Address of QDMA options register     */
#define QDMA_S_SRC      0x02000024      /* Address of QDMA SRC address register */
#define QDMA_S_CNT      0x02000028      /* Address of QDMA counts register      */
#define QDMA_S_DST      0x0200002C      /* Address of QDMA DST address register */
#define QDMA_S_IDX      0x02000030      /* Address of QDMA index register       */

/* Definitions for the DSK Board and SW */
#define PI                      3.1415926
```

```
#define IO_PORT                 0x90080000      /* I/O port Address,top byte valid data  */
#define CE2_PORT                0xA0000000      /* CE2 Port at address A0000000         */
#define INTERNAL_MEM_SIZE       (0x4000)>>2
#define EXTERNAL_MEM_SIZE       (0x400000)>>2
#define FLASH_SIZE              0x20000
#define POST_SIZE               0x10000
#define FLASH_WRITE_SIZE        0x80
#define INTERNAL_MEM_START      0xc000
#define EXTERNAL_MEM_START      0x80000000
#define FLASH_START             0x90000000
#define POST_END                0x90010000
#define FLASH_ADR1              0x90005555
#define FLASH_ADR2              0x90002AAA
#define FLASH_KEY1              0xAA
#define FLASH_KEY2              0x55
#define FLASH_KEY3              0xA0
#define ALL_A                   0xaaaaaaaa
#define ALL_5                   0x55555555
#define CE1_8                   0xffffff03      /* reg to set CE1 as 8bit async         */
#define CE1_32                  0xffffff23      /* reg to set CE1 as 32bit async        */

/* GHOST */
#define GHOST_RESET             0xA0000000
#define GHOST_GCRDATA           0xA0000010
#define GHOST_GCRSTATUS         0xA0000014
#define GHOST_Wi_Wr             0xA0000020
#define GHOST_Wi_Rd             0xA0000024
#define GHOST_TEMP              0xA0000100
```

main.c

```
#include <stdio.h>
#include <c6x.h>
#include "c6211dsk.h"

/************************************************************************
*** My Declaration ******************************************************
************************************************************************/

#define n_Wi        256
#define nw_Wi       64
#define n_Wi2       128
#define nw_Wi2      32
#define n_fifo      640
#define nw_fifo     160
#define n_gcr       512
#define nw_gcr      128
#define MaxWi       32767
#define MinWi       -32768
#define WiGain      256
#define WiGain1     128
#define WiGainL     8
#define WiGainL1    7
#define beta        64

void delay_msec(short msec);
unsigned int timer0_read();
void timer0_start();
void timer0_init();

void ClearXi(void);
void InitXi(int);
void LMSTwoFrame(void);
void LMSOneSample(int data,int gcr,int N);
void DelayOneFrame(void);
int CheckGCRSign(void);
int CheckGCRExist(int idata);
void ProgWi(void);
void ClearWi(void);
void SetWi(void);
void PackGCRData(void);
void AvgGCRData(int);
void WaitGCRData(void);
int WaitGCRData2(void);
int CheckXilinX(void);
void InitDSP(void);
void delay_4nop(short nop);

/************************************************************************
*** My Variables ********************************************************
************************************************************************/

unsigned int data1[2][nw_fifo],data2[2][n_fifo];
signed int Wi[n_Wi],Ei[n_fifo],Yi[n_fifo],Xi[n_Wi];

signed int gcr_ref[2][n_fifo] = {
        {0x2E,0x2E,0x2F,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,
         0x2E,0x2F,0x2E,0x2E,0x2F,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2F,0x2E,0x2E,0x2E,0x2E,
         0x2E,0x2E,0x2E,0x2F,0x2E,0x2E,0x2F,0x2F,0x2F,0x2F,0x2F,0x2F,0x36,0x43,0x54,0x5E,0x60,
         0x60,0x60,0x60,0x60,0x61,0x60,0x60,0x60,0x60,0x60,0x60,0x60,0x60,0x60,0x61,0x61,
         0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,
```

```
0x61,0x61,0x62,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x60,0x60,0x60,0x60,0x60,
0x60,0x60,0x60,0x61,0x60,0x61,0x61,0x60,0x60,0x61,0x60,0x61,0x60,0x60,0x60,0x61,
0x62,0x6E,0x83,0x95,0x9B,0x9B,0x9B,0x9B,0x9C,0x9C,0x9B,0x9B,0x9C,0x9D,0x9C,0x9C,
0x9C,0x9D,0x9D,0x9C,0x9C,0x9D,0x9C,0x9C,0x9D,0x9D,0x9C,0x9C,0x9C,0x9D,0x9D,0x9C,
0x9D,0x9D,0x9D,0x9D,0x9D,0x9E,0x9E,0x9E,0x9E,0x9F,0x9F,0xA0,0xA0,0xA1,0xA1,0xA2,
0xA3,0xA4,0xA6,0xA7,0xA9,0xAB,0xAD,0xB0,0xB2,0xB6,0xB9,0xBB,0xBE,0xC1,0xC5,0xC8,
0xCB,0xCD,0xCE,0xCE,0xCD,0xCA,0xC5,0xBE,0xB5,0xAA,0x9E,0x92,0x85,0x79,0x6E,0x68,
0x66,0x6B,0x70,0x7D,0x8E,0xA2,0xB6,0xC5,0xCE,0xD0,0xC9,0xB8,0xA2,0x88,0x74,0x68,
0x67,0x75,0x8D,0xAA,0xC3,0xD1,0xCE,0xBB,0x9D,0x7E,0x69,0x67,0x7B,0x9C,0xBE,0xD2,
0xCC,0xB0,0x8A,0x6D,0x68,0x7E,0xA6,0xC8,0xD1,0xB9,0x8E,0x6C,0x6B,0x8A,0xB5,0xD0,
0xC4,0x9A,0x70,0x6A,0x8D,0xBB,0xD0,0xB6,0x86,0x69,0x7B,0xAE,0xCE,0xBC,0x8A,0x69,
0x7F,0xB5,0xCF,0xAF,0x79,0x6C,0x98,0xC9,0xC2,0x8B,0x69,0x89,0xC3,0xC8,0x8F,0x68,
0x8B,0xC5,0xC3,0x85,0x6A,0x9B,0xCD,0xB2,0x72,0x77,0xB9,0xC9,0x8C,0x6A,0xA1,0xCE,
0xA2,0x6A,0x8F,0xCB,0xAD,0x6D,0x88,0xC9,0xAE,0x6E,0x8B,0xCB,0xA7,0x6C,0x97,0xCC,
0x96,0x6D,0xAC,0xC5,0x7E,0x7C,0xC3,0xAC,0x6E,0x9F,0xC7,0x83,0x7C,0xC3,0xA6,0x6E,
0xAD,0xBE,0x74,0x94,0xC8,0x84,0x80,0xC6,0x95,0x76,0xC1,0xA1,0x71,0xBB,0xA9,0x6F,
0xB8,0xAC,0x6F,0xB8,0xAA,0x6F,0xBC,0xA4,0x72,0xC1,0x99,0x7A,0xC6,0x89,0x87,0xC7,
0x7A,0x9B,0xBD,0x6E,0xB6,0xA3,0x75,0xCA,0x7F,0x97,0xBE,0x6E,0xBE,0x93,0x87,0xC1,
0x75,0xB3,0x9B,0x86,0xBB,0x7E,0xAD,0x9B,0x8D,0xB2,0x85,0xAA,0x98,0x94,0xAA,0x8B,
0xA8,0x95,0x9A,0xA2,0x92,0xA5,0x95,0x9E,0x9D,0x96,0xA2,0x96,0x9E,0x9C,0x9A,0x9E,
0x99,0x9E,0x9A,0x9D,0x9C,0x9A,0x9E,0x99,0x9D,0x9B,0x9A,0x9D,0x9B,0x9B,0x9C,0x9B,
0x9C,0x9C,0x9B,0x9C,0x9B,0x9B,0x9C,0x9B,0x9C,0x9C,0x9B,0x9C,0x9B,0x9C,0x9B,0x9B,
0x9C,0x9B,0x9B,0x9B,0x9B,0x9C,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9A,0x9B,
0x9B,0x9B,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,0x9A,0x9B,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9A,0x9B,0x9B,0x9B,0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9A,0x9B,0x9B,0x9A,0x9B,0x9A,
0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A  },

{0x2E,0x2E,0x2F,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,
 0x2E,0x2F,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,0x2E,
 0x2E,0x2F,0x2F,0x2F,0x2E,0x2E,0x2E,0x2F,0x2F,0x2F,0x2F,0x36,0x43,0x54,0x5E,0x60,
 0x60,0x60,0x60,0x61,0x61,0x60,0x60,0x60,0x60,0x60,0x60,0x60,0x60,0x61,0x61,0x61,
 0x61,0x61,0x62,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,
 0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x61,0x60,0x60,0x60,0x60,0x60,
 0x60,0x60,0x60,0x61,0x61,0x61,0x61,0x60,0x60,0x61,0x60,0x61,0x60,0x60,0x60,0x61,
 0x62,0x6D,0x83,0x95,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9C,0x9C,0x9C,0x9B,
 0x9C,0x9C,0x9C,0x9C,0x9B,0x9C,0x9C,0x9C,0x9C,0x9C,0x9C,0x9C,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
 0x9B,0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x99,0x99,0x99,0x99,0x98,0x97,0x97,0x96,0x95,
 0x94,0x93,0x92,0x90,0x8E,0x8D,0x8B,0x88,0x86,0x83,0x7F,0x7C,0x79,0x76,0x73,0x70,
 0x6C,0x6B,0x69,0x69,0x6B,0x6D,0x72,0x79,0x82,0x8C,0x98,0xA5,0xB1,0xBE,0xC9,0xD0,
 0xD3,0xD0,0xC7,0xBA,0xAA,0x96,0x82,0x72,0x68,0x67,0x6F,0x81,0x97,0xAE,0xC4,0xD0,
 0xD0,0xC3,0xAC,0x90,0x75,0x67,0x69,0x7C,0x9B,0xBA,0xCF,0xD0,0xBD,0x9C,0x7B,0x67,
 0x6B,0x87,0xAD,0xCB,0xD1,0xBA,0x92,0x6F,0x68,0x7E,0xA9,0xCB,0xCE,0xB0,0x83,0x68,
 0x74,0x9E,0xC6,0xCE,0xAD,0x7D,0x67,0x80,0xB3,0xCF,0xBD,0x8B,0x69,0x7A,0xAE,0xCE,
 0xBA,0x84,0x69,0x87,0xBE,0xCD,0xA1,0x6F,0x74,0xAC,0xD0,0xAF,0x76,0x70,0xA6,0xCF,
 0xAE,0x73,0x74,0xB1,0xCE,0x9E,0x6B,0x85,0xC4,0xC1,0x80,0x6E,0xAB,0xCD,0x99,0x69,
 0x95,0xCD,0xAA,0x6D,0x88,0xCA,0xB0,0x6E,0x87,0xCA,0xAD,0x6E,0x8E,0xCC,0xA1,0x6C,
 0xA0,0xCB,0x8B,0x71,0xB8,0xBC,0x74,0x8B,0xC9,0x99,0x6F,0xB2,0xBD,0x74,0x91,0xC8,
 0x8C,0x78,0xC2,0xA5,0x6F,0xB2,0xB8,0x71,0xA1,0xC2,0x77,0x94,0xC7,0x7D,0x8B,0xC8,
 0x81,0x89,0xC8,0x80,0x8B,0xC8,0x7C,0x92,0xC6,0x76,0x9D,0xBE,0x71,0xAC,0xB2,0x70,
 0xBC,0x9E,0x78,0xC8,0x82,0x92,0xC3,0x6D,0xB6,0xA3,0x78,0xC9,0x7A,0xA2,0xB2,0x75,
 0xC2,0x85,0x99,0xB2,0x7B,0xB9,0x8B,0x9B,0xAB,0x84,0xB2,0x8D,0x9E,0xA4,0x8C,0xAC,
 0x8F,0xA1,0x9D,0x95,0xA5,0x93,0xA3,0x99,0x99,0xA1,0x95,0xA1,0x99,0x9C,0x9D,0x99,
 0x9E,0x9A,0x9E,0x9A,0x9C,0x9D,0x99,0x9D,0x9A,0x9B,0x9D,0x9B,0x9C,0x9B,0x9B,0x9C,
 0x9C,0x9B,0x9C,0x9B,0x9C,0x9C,0x9B,0x9D,0x9B,0x9B,0x9C,0x9B,0x9C,0x9B,0x9C,0x9B,
```

```
0x9B,0x9C,0x9B,0x9B,0x9B,0x9B,0x9C,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9A,0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9B,0x9B,0x9A,0x9B,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9A,0x9B,
0x9A,0x9A,0x9B,0x9B,0x9B,0x9A,0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,0x9B,
0x9A,0x9A,0x9A,0x9A,0x9B,0x9B,0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9B,0x9A,0x9B,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9B,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9B,0x9B,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,
0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A,0x9A};

/*********************************************************************
*** My Main ***********************************************************
*********************************************************************/

void main(void)
{
        int i,sw,imain,init;
        /*_DP = 0x0980;*/
        InitDSP();
        if(CheckXilinX())
        {
                AvgGCRData(0x2);
                if(CheckGCRSign())
                {
                        DelayOneFrame();
                }
                init=0;
                ClearWi();

                imain=0;
                for(i=0;i<256;i++)
                {
                        sw = *(unsigned volatile int *)IO_PORT;
                        sw = sw & 0x07000000;

                        switch(sw)
                        {
                                case 0x07000000:
                                        if(init)
                                        {
                                        /*AvgGCRData(0x10);*/
                                                if(CheckGCRSign())
                                                {
                                                        DelayOneFrame();
                                                }
                                                init=0;
                                        }
                                        LMSTwoFrame();
                                        ProgWi();
                                        break;

                                default:
                                *(unsigned volatile int *)IO_PORT = 0x08000000; /* Turn on all LED */
                                delay_msec(400);
                                *(unsigned volatile int *)IO_PORT = 0x07000000; /* Turn off all LED */
                                *(unsigned volatile int *)GHOST_Wi_Rd = imain;
                                delay_msec(400);
                                imain++;
                                if(imain>=n_Wi)
```

```
                                        {
                                                imain=0;
                                        }
                                        init=1;
                                        break;
                                }
                        }
                }

                /*ClearWi();*/
                ProgWi();
                while(1);
}

/*****************************************************************************
*** My Functions ************************************************************
*****************************************************************************/

/* Variable Changed : Xi, i, temp */
void ReadFifo(int idata)
{
                int i,j,k;
                unsigned int temp;

                for(j=n_Wi; j>=n_Wi2; j--)
                {
                        Xi[j] = 0;
                }

                /**(unsigned volatile int *)GHOST_GCRSTATUS = 1;*/
                k = 0;
                for(i=0;i<nw_Wi2;i++)
                {
                        if(WaitGCRData2())
                        {
                                temp = *(unsigned volatile int *)GHOST_GCRDATA;
                                data1[idata][i] = temp;
                                data2[idata][k] = temp & 0xff;
                                k++;
                                Xi[j] = temp & 0xff;
                                j--;
                                temp = temp >> 8;
                                data2[idata][k] = temp & 0xff;
                                k++;
                                Xi[j] = temp & 0xff;
                                j--;
                                temp = temp >> 8;
                                data2[idata][k] = temp & 0xff;
                                k++;
                                Xi[j] = temp & 0xff;
                                j--;
                                temp = temp >> 8;
                                data2[idata][k] = temp & 0xff;
                                k++;
                                Xi[j] = temp & 0xff;
                                j--;
                        }
                }
                /**(unsigned volatile int *)GHOST_GCRSTATUS = 0;*/

                for(i=nw_Wi2;i<nw_fifo;i++)
                {
                        if(WaitGCRData2())
```

```
                {
                        temp = *(unsigned volatile int *)GHOST_GCRDATA;
                        data1[idata][i] = temp;
                        data2[idata][k] = temp & 0xff;
                        k++;
                        temp = temp >> 8;
                        data2[idata][k] = temp & 0xff;
                        k++;
                        temp = temp >> 8;
                        data2[idata][k] = temp & 0xff;
                        k++;
                        temp = temp >> 8;
                        data2[idata][k] = temp & 0xff;
                        k++;
                }
        }

        for(i=0;i<128;i++)
        {
                Ei[i] = 0;
                Yi[i] = 0;
        }
}

/* Variable Changed : Xi, i, temp */
void InitXi(int idata)
{
        int i,j;
        unsigned int temp;

        for(j=n_Wi; j>=n_Wi2; j--)
        {
                Xi[j] = 0;
        }
        /*j = n_Wi2-1;*/
        for(i=0;i<nw_Wi2-1;i++)
        {
                temp = data1[idata][i];
                Xi[j] = temp & 0xff;
                j--;
                temp = temp >> 8;
                Xi[j] = temp & 0xff;
                j--;
                temp = temp >> 8;
                Xi[j] = temp & 0xff;
                j--;
                temp = temp >> 8;
                Xi[j] = temp & 0xff;
                j--;
        }

        /*for(i=0;i<128;i++)
        {
                Ei[i] = 0;
                Yi[i] = 0;
        }*/
}

int CheckGCRExist(int idata)
{
        return(1);
}
```

```
void LMSTwoFrame(void)
{
	int i,j;
	int tempdata1,tempdata2;
	/*unsigned temp;*/

	for(j=0;j<2;j++)
	{
		*(unsigned volatile int *)GHOST_RESET = 0;			/* Reset XilinX */
		*(unsigned volatile int *)GHOST_GCRSTATUS = 1;			/* Start Get GCR */

		WaitGCRData();
		*(unsigned volatile int *)GHOST_GCRSTATUS = 0;			/* End Get GCR */

		ReadFifo(j);
		CheckGCRExist(j);
		/**(unsigned volatile int *)GHOST_GCRSTATUS = 1;*/
		for(i=n_Wi2;i<n_fifo;i++)
		{
			tempdata1 = data2[j][i];
			tempdata2 = gcr_ref[j][i-16];
			LMSOneSample(tempdata1,tempdata2,i);

			/*if(i==256)
			{
				*(unsigned volatile int *)GHOST_GCRSTATUS = 0;
			}
			else if(i==384)
			{
				*(unsigned volatile int *)GHOST_GCRSTATUS = 1;
			}
			else if(i==512)
			{
				*(unsigned volatile int *)GHOST_GCRSTATUS = 0;
			}*/
		}
	}
}

/* Variable Changed : i, */
void LMSOneSample(int data,int gcr,int N)
{
	int Y,E,x;
	int BETA_E;

	Y = 0;

	for(x=0; x<n_Wi; x++)
	{
		Y = Y + ( (Wi[x]/WiGain) * Xi[x] );
	}
	if(Y > 32767)
	{
		Y = 32767;
	}
	else if(Y < 0)
	{
		Y = 0;
	}
	Yi[N] = Y/128;

	E = gcr - (Y/128);
	BETA_E = (beta*E);
```

```
		Ei[N] = E;

		for(x=n_Wi-1;x>0;x--)
		{
			Wi[x] = Wi[x] + ( (BETA_E*Xi[x]) / 65536 );
			Xi[x] = Xi[x-1];
		 }
		Wi[x] = Wi[x] + ( (BETA_E*Xi[x]) / 65536 );
		Xi[0] = data;
		for(x=0;x<n_Wi;x++)
		{
			if( Wi[x]>MaxWi )
			{
				Wi[x]=MaxWi;
			}
		}
		for(x=0;x<n_Wi;x++)
		{
			if( Wi[x]<MinWi )
			{
				Wi[x]=MinWi;
			}
		}
}

void DelayOneFrame(void)
{
		*(unsigned volatile int *)GHOST_RESET = 0;			/* Reset XilinX */
		*(unsigned volatile int *)GHOST_GCRSTATUS = 1;			/* Start Get GCR */

		WaitGCRData();
		*(unsigned volatile int *)GHOST_GCRSTATUS = 0;			/* End Get GCR */

		delay_4nop(2400);
		*(unsigned volatile int *)GHOST_RESET = 0;			/* Reset XilinX */
}

/* Sign=1 when GCR(-) */
int CheckGCRSign(void)
{
		int i,a1,a2,sign;

		a1=0;
		a2=0;
		for(i=0x80;i<0x98;i++)
		{
			a1 = a1 + data2[0][i];
			a2 = a2 + data2[0][i+0x18];
		}

		if(a1>a2)
		{
			sign=1;
		}
		else
		{
			sign=0;
		}

		return(sign);
}

/* Variable Changed : i, j */
```

```
void ProgWi(void)
{
	int i;

	for(i=0;i<n_Wi;i++)
	{
		*(unsigned volatile int *)GHOST_Wi_Wr = (Wi[(n_Wi-1)-i]/WiGain);		/* Write Wi */
		delay_4nop(7);
	}
}

/* Variable Changed : Wi, i */
void ClearWi(void)
{
	int i;

	for(i=0;i<n_Wi;i++)
	{
		Wi[i] = 0x0;
	}
	Wi[15] = 0x7f<<WiGainL;
}

/* Variable Changed : data1, i, j */
void PackGCRData(void)
{
	int i,j;

	for(j=0;j<2;j++)
	{
		for(i=0;i<nw_fifo;i++)
		{
			data1[j][i] = data2[j][(4*i)] | (data2[j][(4*i)+1]<<8) | (data2[j][(4*i)+2]<<16) |
			(data2[j][(4*i)+3]<<24);
		}
	}

	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0E000000; /* Turn on 1st LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0D000000; /* Turn on 2nd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0B000000; /* Turn on 3rd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0E000000; /* Turn on 1st LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0D000000; /* Turn on 2nd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0B000000; /* Turn on 3rd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0E000000; /* Turn on 1st LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0D000000; /* Turn on 2nd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0B000000; /* Turn on 3rd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0E000000; /* Turn on 1st LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0D000000; /* Turn on 2nd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0B000000; /* Turn on 3rd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0E000000; /* Turn on 1st LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0D000000; /* Turn on 2nd LED */
	delay_msec(200);
	*(unsigned volatile int *)IO_PORT = 0x0B000000; /* Turn on 3rd LED */
	delay_msec(200);
}
```

```
/* Variable Used : data2, i, j, k */
void AvgGCRData(int n_frame)
{
	int i,j,k;
	unsigned int temp;

	for(j=0;j<2;j++)
	{
		for(i=0;i<nw_fifo;i++)
		{
			data2[j][(4*i)+0] = 0x00;
			data2[j][(4*i)+1] = 0x00;
			data2[j][(4*i)+2] = 0x00;
			data2[j][(4*i)+3] = 0x00;
		}
	}

	for(k=0;k<n_frame;k++)
	{

		*(unsigned volatile int *)IO_PORT = ( ((k<<3)/n_frame) & 0x07) << 24;

		for(j=0;j<2;j++)
		{
			*(unsigned volatile int *)GHOST_RESET = 0;		/* Reset XilinX */
			*(unsigned volatile int *)GHOST_GCRSTATUS = 1;	/* Start Get GCR */

			WaitGCRData();
			*(unsigned volatile int *)GHOST_GCRSTATUS = 0;	/* End Get GCR */

			for(i=0;i<nw_fifo;i++)
			{
				if(WaitGCRData2())
				{
					temp = *(unsigned volatile int *)GHOST_GCRDATA;
					data2[j][(4*i)+0] = data2[j][(4*i)+0] + ( temp & 0xff );
					data2[j][(4*i)+1] = data2[j][(4*i)+1] + ( (temp >> 8) & 0xff );
					data2[j][(4*i)+2] = data2[j][(4*i)+2] + ( (temp >> 16) & 0xff );
					data2[j][(4*i)+3] = data2[j][(4*i)+3] + ( (temp >> 24) & 0xff );
				}
			}
		}
	}

	for(j=0;j<2;j++)
	{
		for(i=0;i<(4*nw_fifo);i++)
		{
			data2[j][i] = ( (data2[j][i]/n_frame) + ((data2[j][i]/(n_frame/2))%2) ) & 0xff;
		}
	}
}

void WaitGCRData(void)
{
	unsigned int xtmp;

	do
	{
		xtmp = *(unsigned volatile int *)GHOST_GCRSTATUS;
		xtmp = xtmp & 0x04;
	} while(xtmp!=0x04);
```

```
}

int WaitGCRData2(void)
{
	unsigned int xtmp,xi;

	xi=0;
	do
	{
		xtmp = *(unsigned volatile int *)GHOST_GCRSTATUS;
		xtmp = xtmp & 0x04;
		xi++;
	} while(xtmp!=0x04 && xi<100);
	if(xi<100)
	{
		return 1;
	}
	return 0;
}

int CheckXilinX(void)
{
	*(unsigned volatile int *)GHOST_RESET = 0;			/* Reset XilinX */
	*(unsigned volatile int *)GHOST_GCRSTATUS = 0;			/* End Get GCR */
	if(*(unsigned volatile int *)GHOST_RESET==0xf0c3a596)
	{
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = 0x07000000; /* Turn off all LED */
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x01) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x02) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x03) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x04) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x05) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = (~(0x06) & 0x0f) << 24;
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = 0x08000000; /* Turn on all LED */
		delay_msec(200);
		*(unsigned volatile int *)IO_PORT = 0x07000000; /* Turn off all LED */
		delay_msec(200);
		return 1;
	}
	return 0;
}

void InitDSP(void)
{
	*(unsigned volatile int *)EMIF_CE2 = 0x23D2CF21;
	/**(unsigned volatile int *)EMIF_CE2 = 0x8828E022;*/
	timer0_start();
}

/*****************************************************************************
*** Others Functions ********************************************************
*****************************************************************************/

/*****************************************************************************
* FUNCTION : timer0_read
```

```
*
* ARGUMENTS :
*   VOID
*
* DESCRIPTION :
*   Read and return the count in timer 0.
*
* OUTPUTS :
*   INT              <-- Returns the count in timer 0
*
*****************************************************************************/
unsigned int timer0_read()
{
 unsigned int z;

 z = *(unsigned volatile int *)TIMER0_COUNT;
 return z;
}

/*****************************************************************************
* FUNCTION : timer0_start
*
* ARGUMENTS :
*   VOID
*
* DESCRIPTION :
*   Start timer 0 after initializing it for 32-bit count and no interrupt.
*
* OUTPUTS :
*   VOID
*
*****************************************************************************/
void timer0_start()
{
 /* Hold the timer */
 *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;

 /* Use CPU CLK/4 */
 *(unsigned volatile int *)TIMER0_CTRL |= 0x200;

 /* Set for 32 bit counter */
 *(unsigned volatile int *)TIMER0_PRD  |= 0xffffffff;

 /* Start the timer */
 *(unsigned volatile int *)TIMER0_CTRL |= 0xC0;
}

/*****************************************************************************
*****************************************************************************/
void timer0_hold()
{
 /* Hold the timer */
 *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;
}

/*****************************************************************************
* FUNCTION : timer0_init
*
* ARGUMENTS :
*   VOID
*
* DESCRIPTION :
*   Start timer 0 after initializing it for a short period (~13.7 micro seconds)
```

```
*   with interrupt.
*
* OUTPUTS :
*   VOID
*
*******************************************************************************/
void timer0_init()
{
 /* Hold the timer */
 *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;

 /* Use CPU CLK/4 */
 *(unsigned volatile int *)TIMER0_CTRL |= 0x200;

 /* Set for a short period */
 *(unsigned volatile int *)TIMER0_PRD   = 0x200;

 /* Start the timer, enable timer0 int */
 *(unsigned volatile int *)TIMER0_CTRL |= 0x3C0;
}

/*******************************************************************************
* FUNCTION : delay_msec
*
* ARGUMENTS :
*   SHORT msec          <-- Period to delay in milliseconds
*
* DESCRIPTION :
*
*
* OUTPUTS :
*   VOID
*
*******************************************************************************/
void delay_msec(short msec)
{
 /* Assume 150 MHz CPU, timer peirod = 4/150 MHz */
 unsigned int timer_limit = (msec*9375)<<2;
 unsigned int time_start;

/*  timer0_start();*/
 time_start = timer0_read();
 while ((timer0_read()-time_start) < timer_limit);
/*  timer0_hold();*/
}

void delay_4nop(short nop)
{
 /* Assume 150 MHz CPU, timer peirod = 4/150 MHz */
 unsigned int time_start;

/*  timer0_start();*/
 time_start = timer0_read();
 while ((timer0_read()-time_start) < nop);
/*  timer0_hold();*/
}
```

เครื่องกำจัดผีในสัญญาณโทรทัศน์โดยใช้วงจรกรองเอฟไออาร์แบบเอฟพีจีเอ
A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS

พหล ศิริเหลืองทอง และวันเฉลิม โปรา
ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์
จุฬาลงกรณ์มหาวิทยาลัย กรุงเทพมหานคร 10330
โทรศัพท์: 218-6537 ต่อ 31, Email: jack@digital.ee.eng.chula.ac.th

บทคัดย่อ

บทความนี้กล่าวถึงการออกแบบเครื่องกำจัดผีในสัญญาณโทรทัศน์ระบบ PAL ซึ่งใช้สัญญาณอ้างอิงเพื่อกำจัดผี (GCR Signal) ตามมาตรฐาน ITU ส่วนประกอบสำคัญของเครื่องกำจัดผีคือ วงจรกรองเอฟไออาร์แบบปรับตัว (Adaptive FIR Filters) พัฒนาบนชิพเอฟพีจีเอของบริษัท Xilinx เบอร์ XCV300E-6 โดยนำเทคนิคการใช้ทรัพยากรร่วม (Resource Sharing) มาช่วยในประหยัดทรัพยากรในชิพเอฟพีจีเอได้เป็นอย่างมาก และชิพประมวลผลสัญญาณเชิงเลข (DSP Chip) เบอร์ TMS320C6211 ในการคำนวณหาสัมประสิทธิ์ของวงจรกรอง ด้วยอัลกอริทึมแบบแอลเอ็มเอส (LMS : Least Mean Square) สำหรับการทดสอบการทำงานของเครื่องกำจัดผี เราได้พัฒนาเครื่องจำลองการเกิดผีแบบง่ายขึ้นด้วยการใช้ดีเลย์ไลน์ (Delay Line) และผลการทดสอบพบว่าสามารถกำจัดผีที่มีความหน่วงอยู่ในช่วง –1 μs ถึง +20 μs ได้

Abstract

This paper presents a PAL TV ghost canceller that exploits a ghost cancellation reference signal (GCR Signal) of a ITU standard. The Ghost canceller is composed of an adaptive FIR filter, which developed on a Xilinx FPGA, XCV300E-6. The resource sharing technique is employed in order to reduce the size of the circuit. TMS320C6211, a DSP chip, optimizes the coefficients of the filtering accords to the LMS (Least Mean Square) algorithm. We also developed a ghost generator using delay line for testing the TV ghost canceller. From our experimentation, the ghost canceller can visually cancel the ghost that its delay range is from –1 μs to +20 μs.

Keyword: TV Ghost Canceller, Ghost Cancellation Reference (GCR) Signal, Adaptive FIR filter, Field-Programmable Gate Array (FPGA), Resource Sharing

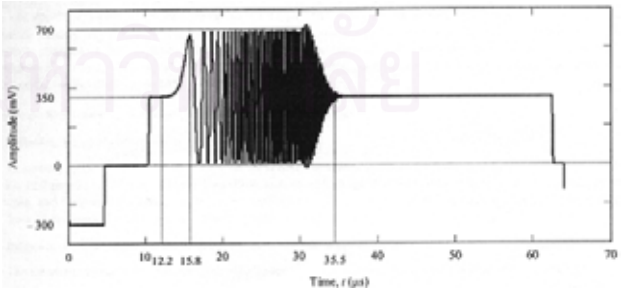
1. บทนำ

ในปัจจุบันนี้ มีการใช้งานโทรทัศน์กันอย่างแพร่หลายมาก อาจจะเรียกได้ว่ามีใช้กันแทบทุกครัวเรือน จึงก่อให้เกิดการพัฒนาเทคโนโลยีในระบบการรับ-ส่งสัญญาณโทรทัศน์ ทั้งทางด้านสถานีส่งและเครื่องรับโทรทัศน์ให้มีคุณภาพของทั้งภาพ และเสียงที่ดีขึ้น แต่ปัญหาหนึ่งที่ยังพบเห็นได้บ่อยครั้งคือ การเกิดภาพซ้อนหรือผีบนจอโทรทัศน์ อันเนื่องมาจากสถานีส่ง ส่งสัญญาณโทรทัศน์ออกมาแบบทุกทิศทาง ทำให้เครื่องรับได้รับสัญญาณโทรทัศน์มาจากหลายทิศทาง โดยในแต่ละทิศทางจะเดินทางมาถึงไม่พร้อมกัน และมีความแรงของสัญญาณไม่เท่ากัน ทำให้เห็นเป็นเงาภาพซ้อนกันบนจอภาพของเครื่องรับโทรทัศน์ [1]

ปัญหานี้เป็นปัญหาเฉพาะด้าน ซึ่งไม่สามารถแก้ไขด้วยกรรมวิธีทางด้านอิเล็กทรอนิกส์แบบดั้งเดิมเพียงอย่างเดียว แต่ต้องใช้กรรมวิธีในการประมวลผลสัญญาณเชิงเลขมาช่วยในการแก้ปัญหา ซึ่งในบทความนี้จะบรรยายถึง ส่วนประกอบต่างๆที่สำคัญ ของระบบกำจัดผีในสัญญาณโทรทัศน์ โดยจะเน้นไปที่เครื่องกำจัดผีในสัญญาณโทรทัศน์ที่ได้ทำการพัฒนาขึ้น

2. สัญญาณอ้างอิงเพื่อกำจัดผี

ในระบบกำจัดผีของสัญญาณโทรทัศน์ สัญญาณอ้างอิงเพื่อการกำจัดผีจะถูกแทรกเข้าไปในช่วงไร้ภาพทางแนวตั้ง (Vertical Blanking Interval) ของสัญญาณโทรทัศน์ [2] ที่สถานีส่ง และถูกส่งแพร่ภาพออกมาพร้อมสัญญาณโทรทัศน์ เพื่อให้เครื่องกำจัดผีที่ภาครับ ใช้สัญญาณอ้างอิงในการกำจัดผีก่อนปล่อยให้สัญญาณที่ปราศจากผีผ่านไปสู่เครื่องรับโทรทัศน์



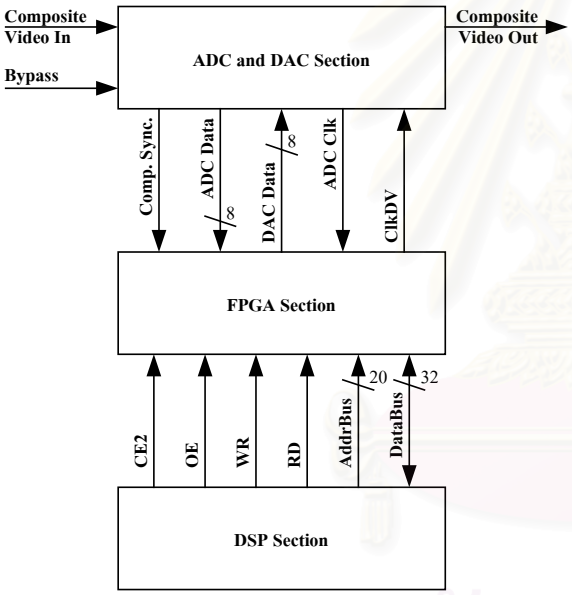
รูปที่ 1 สัญญาณอ้างอิงเพื่อกำจัดผี

ในโครงงานนี้ เราใช้สัญญาณอ้างอิงตามมาตรฐาน ITU [3] ดังรูปที่ 1 โดยใช้เครื่อง PM 5655 VITS Generator & Inserter ให้กำเนิด

สัญญาณอ้างอิงเพื่อใช้กำจัดผี และแทรกสัญญาณเข้าไปในสัญญาณวีดี-ทัศน์ ซึ่งสัญญาณอ้างอิงมีลักษณะเป็นสัญญาณรูปไซน์ (Sine Signal) ที่เปลี่ยนแปลงความถี่ตามเวลา โดยความถี่จะเปลี่ยนจากความถี่ต่ำไปความถี่สูง ในช่วงความถี่ของสัญญาณโทรทัศน์ มีขนาดของสัญญาณเท่ากันตลอดทุกความถี่ มีเฟสที่ต่อเนื่อง (Smooth Phase) ข้อดีที่สำคัญของสัญญาณในลักษณะนี้คือ มีผลกระทบน้อย ต่อการเปลี่ยนแปลงอัตราการสุ่มข้อมูล (Sampling Rate) และจำนวนบิตของข้อมูล

3. เครื่องกำจัดผีในสัญญาณโทรทัศน์

เครื่องกำจัดผีประกอบด้วยส่วนหลัก 3 ส่วนคือส่วนแปลงสัญญาณแอนาลอก-ดิจิตอล (ADC and DAC Section), ส่วนเอฟพีจีเอ (FPGA Section), ส่วนประมวลผลสัญญาณเชิงเลข (DSP Section) ดังรูปที่ 2 โดยในแต่ละส่วนจะมีโครงสร้างภายใน และหน้าที่การทำงานต่างๆ ดังจะได้กล่าวรายละเอียดในหัวข้อต่อไป



รูปที่ 2 แผนภาพบล็อกของเครื่องกำจัดผี

3.1. ส่วนแปลงสัญญาณแอนาลอก-ดิจิตอล

ส่วนนี้จะเป็นส่วนที่ติดต่อกับสัญญาณภายนอกเครื่องกำจัดผี กล่าวคือเป็นส่วนที่ทำหน้าที่รับสัญญาณวีดีทัศน์ที่มีสัญญาณผีเข้ามาแปลงให้อยู่ในรูปของสัญญาณเชิงเลข เพื่อส่งต่อให้ส่วนอื่นต่อไป และสัญญาณที่ปราศจากสัญญาณผีแล้วนั้น จะถูกส่งกลับมายังส่วนนี้ เพื่อแปลงให้กลับเป็นสัญญาณวีดีทัศน์แบบแอนะลอกเช่นเดิม ซึ่งในส่วนนี้จะประกอบด้วยไปด้วย 3 ส่วนหลักคือ

3.1.1. วงจรคงค่าระดับแรงดัน (Clamp Circuit) เป็นส่วนที่รับสัญญาณวีดีทัศน์เข้ามาแล้วทำการตรึงยอดล่างของสัญญาณซิงก์โครไนซ์ ให้อยู่ที่ระดับแรงดันคงที่ค่าหนึ่ง โดยใช้ชิพ LM1881 เป็นตัวคงค่าระดับแรงดันไว้ที่ 1.5 V อีกทั้งยังสร้างสัญญาณซิงก์โครไนซ์รวม (Composite Sync.) ให้อยู่ในรูปของสัญญาณเชิงเลข เพื่อส่งให้กับเอฟพีจีเอ ใช้ในการนับเส้นภาพเพื่อเก็บสัญญาณอ้างอิงด้วย

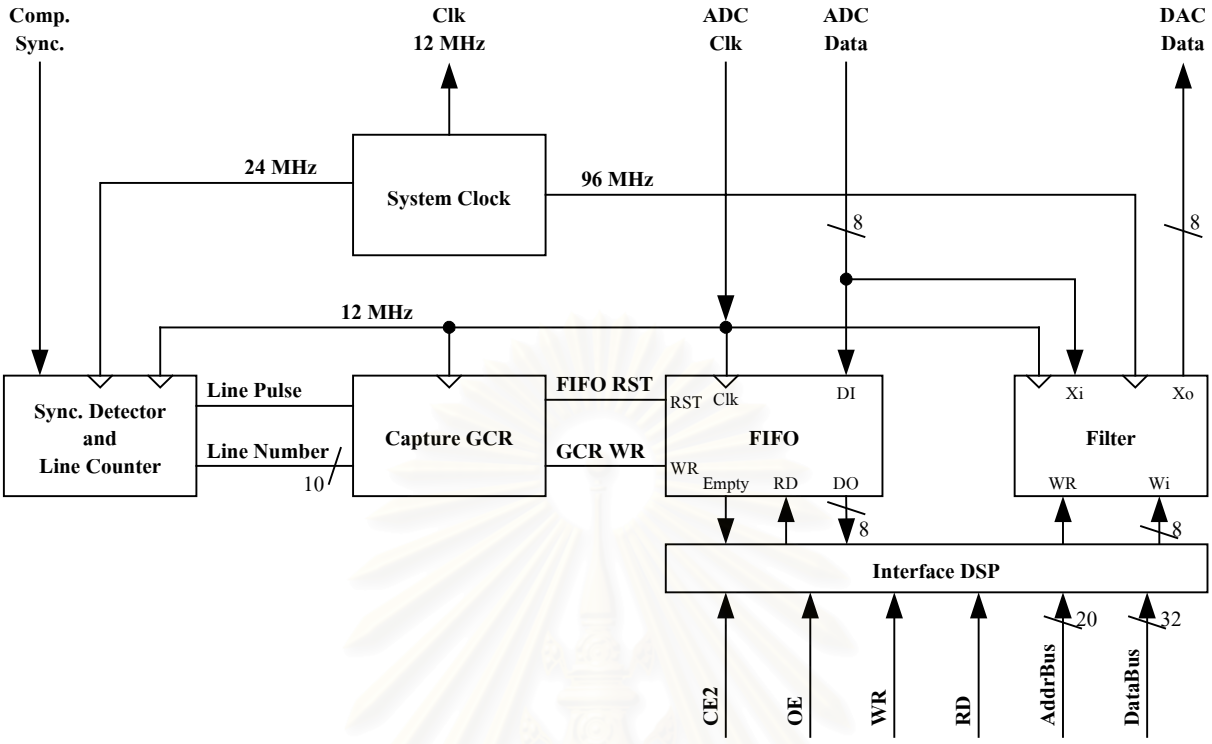
3.1.2. วงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณเชิงเลข (ADC Circuit) เป็นส่วนที่ทำหน้าที่แปลงสัญญาณวีดีทัศน์ที่ผ่านตัวคงค่าระดับแรงดันแล้ว ให้อยู่ในรูปของสัญญาณเชิงเลขขนาด 8 บิต ทำงานที่อัตราสุ่มข้อมูล 12 MS/s โดยใช้ชิพของบริษัท Texas Instruments เบอร์ THS1031IDW แล้วส่งสัญญาณเชิงเลขที่ได้ ให้กับส่วนเอฟพีจีเอต่อไป

3.1.3. วงจรแปลงสัญญาณเชิงเลขเป็นสัญญาณแอนาลอก (DAC Circuit) เป็นส่วนที่ทำหน้าที่แปลงสัญญาณภาพเชิงเลขที่ปราศจากผีซึ่งรับมาจากส่วนเอฟพีจีเอ ให้กลับมาอยู่ในรูปของสัญญาณวีดีทัศน์แบบแอนะลอก โดยใช้ชิพของบริษัท Texas Instruments เบอร์ THS5641AIDW แล้วส่งสัญญาณวีดีทัศน์แบบแอนาลอกที่ได้ ออกไปยังเครื่องรับโทรทัศน์ต่อไป

3.2. ส่วนเอฟพีจีเอ

การกำจัดผีในสัญญาณโทรทัศน์นั้น ทำได้โดยการผ่านข้อมูลสัญญาณภาพที่มีผี ไปยังวงจรกรองที่ถูกกำหนดค่าสัมประสิทธิ์ให้สามารถกำจัดผีได้ โดยค่าสัมประสิทธิ์ดังกล่าวจะถูกคำนวณโดยส่วนประมวลผลสัญญาณเชิงเลข ดังนั้นวงจรกรองจึงเป็นส่วนประกอบหนึ่งที่สำคัญมากในการกำจัดผีในสัญญาณโทรทัศน์ อีกทั้งเพื่อให้สามารถกำจัดผีที่มีความหน่วงมากๆได้ จึงจำเป็นต้องใช้วงจรกรองที่มีความยาวสูงตามไปด้วย ทำให้สิ้นเปลืองทรัพยากรภายในเอฟพีจีเอเป็นจำนวนมาก จึงได้นำเทคนิคการใช้ทรัพยากรร่วม มาช่วยในการประหยัดทรัพยากรภายในเอฟพีจีเอดังจะได้กล่าวในรายระเอียดต่อไป

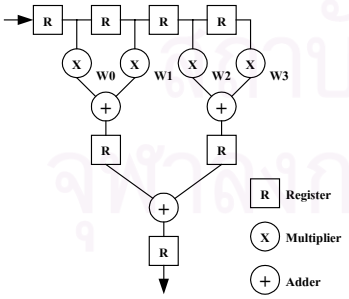
สำหรับส่วนประกอบอื่นภายในเอฟพีจีเอ ได้แก่ส่วนของวงจรตรวจจับสัญญาณซิงก์โครไนซ์ และนับจำนวนเส้นภาพ (Sync. Detector and Line Counter), ส่วนตรวจจับสัญญาณอ้างอิงเพื่อกำจัดผี (Capture GCR) และหน่วยความจำแบบเข้าก่อนออกก่อน (FIFO) ซึ่งส่วนต่างๆเหล่านี้ เป็นส่วนช่วยในการตรวจจับ และเตรียมข้อมูลของสัญญาณอ้างอิงเพื่อกำจัดผี ไว้ให้กับส่วนประมวลผลสัญญาณเชิงเลขนำไปคำนวณต่อไป ดังรูปที่ 3



รูปที่ 3 แผนภาพบล็อกของวงจรที่สังเคราะห์ลงในเอฟพีจีเอ

3.2.1. วงจรกรองเอฟไอออาร์โดยการใช้ทรัพยากรร่วม

ในกระบวนการกำจัดผีในสัญญาณโทรทัศน์ จำเป็นต้องใช้วงจรกรองเอฟไอออาร์ ที่มีความยาวสูงมากเพื่อให้สามารถกำจัดผีที่มีความหน่วงทางเวลามากๆได้ ดังนั้นวงจรกรองเอฟไอออาร์จึงเป็นส่วนที่ใช้ทรัพยากรภายในชิพเอฟพีจีเอมากที่สุด โดยโครงสร้างของวงจรกรองประกอบไปด้วยวงจรคูณ, วงจรบวก และรีจิสเตอร์เป็นจำนวนมากดังรูปที่ 4



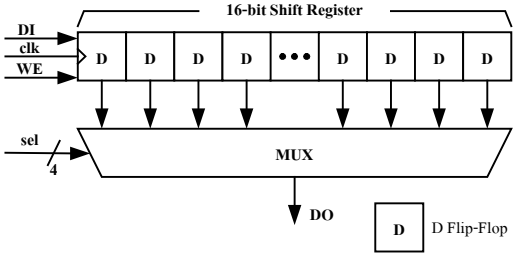
รูปที่ 4 โครงสร้างทางกายภาพของวงจรกรองเอฟไอออาร์

จากการทดสอบสังเคราะห์วงจรกรองเอฟไอออาร์นี้ลงบนเอฟพีจีเอเบอร์ XCV300E พบว่าวงจรกรองที่มีความยาวเพียง 32 แท็ป จะใช้ทรัพยากรในเอฟพีจีเอไปมากกว่าครึ่งของทรัพยากรทั้งหมด หรือมากกว่า 150,000 เกต แต่วงจรสามารถทำงานได้ที่ความถี่สูงมาก โดยเฉพาะเมื่อใช้วงจรคูณแบบไปป์ไลน์ [4] แล้ว วงจรกรองจะสามารถทำงานได้ที่ความถี่สูงกว่า 140 MHz ในขณะที่ใช้ทรัพยากรภายในเอฟพีจีเอเพิ่มขึ้นอีกเพียงเล็กน้อยเท่านั้น ดังตารางที่ 1

เทคนิค	ทรัพยากร	ความเร็ว
วงจรคูณแบบธรรมดา	54%	75.7 MHz
วงจรคูณแบบไปป์ไลน์	60%	144.4 MHz

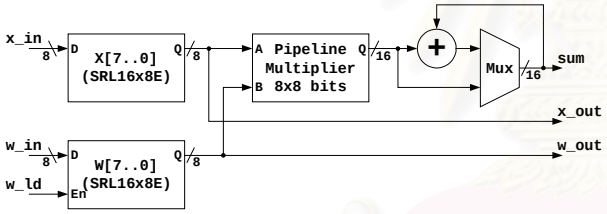
ตารางที่ 1 ประสิทธิภาพของวงจรกรองเอฟไอออาร์ขนาด 32 แท็ป

จากผลที่ได้ดังตารางที่ 1 พบว่าวงจรกรองที่สังเคราะห์ลงบนเอฟพีจีเอนั้น มีความเร็วสูงกว่าอัตราสุ่มข้อมูลที่ต้องการ (12 MS/s) อยู่มาก อีกทั้งโครงสร้างภายในเอฟพีจีเอที่ใช้ ซึ่งประกอบไปด้วย วงจร 2 ส่วนหลักคือ วงจรแอลยูที (LUT : Look-Up Table) ขนาด 4 อินพุต และรีจีสเตอร์ โดยที่วงจรส่วนแอลยูทีสามารถออกแบบให้เป็นวงจรที่มีลักษณะเป็น รีจีสเตอร์เลื่อนข้อมูล (Shift Register) กับตัวมัลติเพลกซ์ (Multiplexer) ดังรูปที่ 5 ซึ่งใช้ชื่อย่อว่า SRL16x1E สำหรับ 1 แอลยูที ใช้เก็บข้อมูลขนาด 1 บิต และเมื่อต้องการข้อมูลขนาด 8 บิต ทำได้โดยการนำ SRL16x1E ดังกล่าวมาต่อขนานกัน 8 ตัว โดยใช้ชื่อย่อว่า SRL16x8E



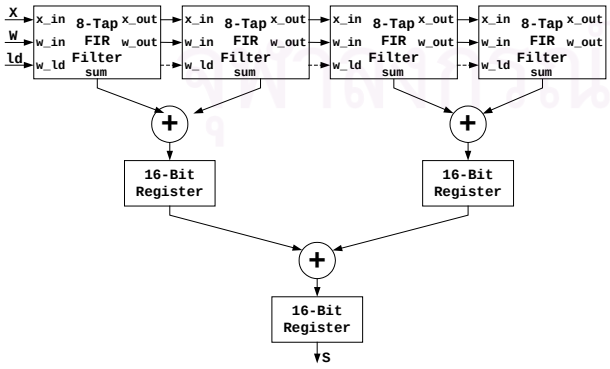
รูปที่ 5 วงจรเลื่อนข้อมูลกับตัวมัลติเพลกซ์ (SRL16x1E)

จากโครงสร้างของเอฟพีจีเอดังกล่าว เราจึงใช้ประโยชน์จากจุดนี้ กับการนำเทคนิคการใช้ทรัพยากรร่วม (Resource Sharing) มาช่วยในการออกแบบวงจรกรอง ดังรูปที่ 6 แสดงโครงสร้างของวงจรกรองขนาด 8 แท็ป ที่ใช้เทคนิคการใช้ทรัพยากรร่วม โดยการใช้ SRL16x8E เป็นตัวเก็บข้อมูล และสัมประสิทธิ์ของวงจรกรองเพียง 2 ตัว แล้วนำผลลัพธ์สลับกันผ่านเข้าวงจรคูณแบบไปป์ไลน์ขนาด 8x8 บิต ทีละชุด แล้วนำผลคูณในแต่ละชุดผ่านวงจรบวกเพื่อหาผลรวมของทั้ง 8 ชุด ซึ่งจะเห็นได้ว่าด้วยเทคนิคการใช้ทรัพยากรร่วมนี้ จะทำให้เราสามารถประหยัดทรัพยากรลงเหลือเพียง 1 วงจรคูณกับ 1 วงจรบวกเท่านั้น สำหรับวงจรกรองขนาด 8 แท็ป



รูปที่ 6 วงจรกรองเอฟไอออาร์ขนาด 8 แท็ป โดยใช้ทรัพยากรร่วม

และเมื่อต้องการวงจรกรองที่มีความยาวเพิ่มขึ้นก็สามารถนำวงจรกรองขนาด 8 แท็ปที่ทำการออกแบบไว้นี้ มาต่อกันดังรูปที่ 7 โดยผลลัพธ์ของวงจรกรองแต่ละชุด จะต้องนำมาผ่านวงจรบวกเพื่อหาผลรวมของวงจรกรองทั้งหมดอีกครั้ง



รูปที่ 7 วงจรกรอง FIR ที่มีความยาวมากกว่า 8 แท็ป

การนำเทคนิคการใช้ทรัพยากรร่วมมาใช้กับวงจรกรองเอฟไอออาร์นั้น ช่วยให้สามารถประหยัดทรัพยากรภายในเอฟพีจีเอไปได้เป็นอย่างมากดังตารางที่ 2 จะเห็นว่าเราสามารถสร้างวงจรกรองเอฟไอออาร์ขนาด 256 แท็ปภายในเอฟพีจีเอตัวเดิมนี้ได้ โดยใช้ทรัพยากรไปเพียง 73% (ประมาณ 219,000 เกต) เท่านั้น ซึ่งวงจรยังคงมีความเร็วสูงถึง 141.9 MHz แต่ด้วยการใช้ทรัพยากรร่วม 8 ครั้ง ทำให้วงจรกรองนี้ทำงานที่อัตราเร็วของข้อมูลได้ที่ 17.7 MHz ซึ่งยังคงมากกว่าอัตราสุ่มข้อมูลที่ต้องการที่ 12 MHz อยู่

เทคนิค	ความยาว (แท็ป)	ทรัพยากร	ความเร็ว (MHz)	
			วงจร	ข้อมูล
วงจรคูณแบบไปป์ไลน์	32	60%	144.4	144.4
วงจรคูณแบบไปป์ไลน์ กับการใช้ทรัพยากรร่วม 8 ครั้ง	32	9%	147.1	18.3
วงจรคูณแบบไปป์ไลน์ กับการใช้ทรัพยากรร่วม 8 ครั้ง	256	73%	141.9	17.7

ตารางที่ 2 ตารางเปรียบเทียบประสิทธิภาพของวงจรกรองเอฟไอออาร์แบบเดิม กับแบบใช้ทรัพยากรร่วม 8 ครั้ง

3.2.2. วงจรตรวจจับสัญญาณซิงก์โครไนซ์ และนับจำนวนเส้นภาพ

เป็นส่วนที่รับสัญญาณซิงก์โครไนซ์รวมเข้ามา แล้วทำการตรวจหาจุดเริ่มต้นของฟิลด์ ซึ่งเป็นส่วนหนึ่งในช่วงไร้ภาพทางแนวราบ แล้วทำการนับเส้นภาพ ส่งต่อให้กับส่วนตรวจจับสัญญาณอ้างอิงเพื่อกำจัดผีต่อไป

3.2.3. ส่วนตรวจจับสัญญาณอ้างอิงเพื่อกำจัดผี

เป็นส่วนที่ควบคุมการเขียนข้อมูลสัญญาณอ้างอิงเพื่อกำจัดผีลงในหน่วยความจำแบบเข้าก่อนออกก่อน กล่าวคือเมื่อรับจำนวนเส้นสัญญาณภาพมาแล้ว จะทำการเปรียบเทียบ เมื่อถึงเส้นภาพที่มีสัญญาณอ้างอิงเพื่อกำจัดผีอยู่ ก็จะทำการหน่วงเวลาจากขอบของสัญญาณซิงก์โครไนซ์ไปยังตำแหน่งเริ่มต้นของสัญญาณอ้างอิงเพื่อกำจัดผี จึงส่งสัญญาณเขียนข้อมูลไปยังหน่วยความจำแบบเข้าก่อนออกก่อน

3.2.4. หน่วยความจำแบบเข้าก่อนออกก่อน

ทำหน้าที่เก็บข้อมูลสัญญาณอ้างอิงเพื่อกำจัดผี โดยเมื่อมีข้อมูลอยู่ในหน่วยความจำ จะส่งสัญญาณบอกสถานะไปยังส่วนประมวลผลสัญญาณเชิงเลข แล้วรอให้ส่วนประมวลผลสัญญาณเชิงเลขมาอ่านข้อมูล

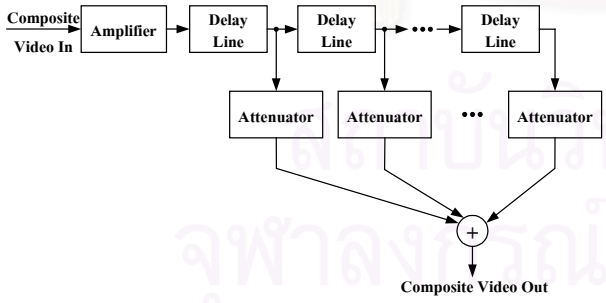
สัญญาณอ้างอิงไป เพื่อใช้ในการคำนวณหาสัมประสิทธิ์ของการกำจัดผีที่เหมาะสมต่อไป

3.3. ส่วนประมวลผลสัญญาณเชิงเลข

ส่วนประมวลผลสัญญาณเชิงเลข เป็นอีกส่วนหนึ่งที่มีความสำคัญเป็นอย่างมาก ซึ่งเป็นส่วนที่ทำหน้าที่คำนวณหาค่าสัมประสิทธิ์ของวงจรกรองเอฟไออาร์เพื่อใช้ในการกำจัดผี โดยส่วนประมวลผลสัญญาณเชิงเลขนี้ จะรับข้อมูลของสัญญาณอ้างอิงเพื่อกำจัดผีเข้ามา จากนั้นทำการคำนวณ โดยใช้อัลกอริทึมแบบแอลเอ็มเอส (LMS : Least Mean Square) [5] จากนั้นเมื่อได้ค่าสัมประสิทธิ์ของวงจรกรองเอฟไออาร์เพื่อใช้ในการกำจัดผีแล้ว ก็ส่งค่าที่ได้ให้กับวงจรกรองเอฟไออาร์ในส่วนของเอฟพีจีเอต่อไป และเพื่อให้ระบบมีเสถียรภาพที่ดีขึ้น ส่วนประมวลผลสัญญาณเชิงเลขนี้ จะคอยตรวจสอบผลรวมของค่าความผิดพลาด เมื่อผลรวมของค่าความผิดพลาดมีค่าสูงเกินค่าที่ตั้งไว้ ส่วนประมวลผลสัญญาณเชิงเลขนี้จะทำการคำนวณหาสัมประสิทธิ์ของวงจรกรองใหม่อีกครั้งหนึ่ง

4. การทดสอบการทำงานของเครื่องกำจัดผี

เนื่องจากสัญญาณผีที่เกิดขึ้นเองตามธรรมชาติ จะมีคุณสมบัติไม่แน่นอนขึ้นกับสภาพแวดล้อมต่างๆ ทำให้ลำบากในการทดสอบเครื่องกำจัดผีในสัญญาณโทรทัศน์ เราจึงได้ทำการพัฒนาเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์อย่างง่ายขึ้น ดังรูปที่ 8 เพื่อใช้ในการทดสอบการทำงานของเครื่องกำจัดผีในสัญญาณโทรทัศน์ โดยการนำสัญญาณภาพผ่านวงจรหน่วงเวลา (Delay Line) เพื่อให้ได้สัญญาณภาพที่มีความหน่วงทางเวลาหลายๆ ค่า จากนั้นนำสัญญาณภาพที่ความหน่วงต่างๆ ผ่านวงจรลดทอนสัญญาณ (Attenuator) ที่มีอัตราการลดทอนไม่เท่ากัน แล้วนำสัญญาณที่ได้มารวมกัน ก็จะได้สัญญาณภาพที่มีผีเกิดขึ้น



รูปที่ 8 โครงสร้างของเครื่องจำลองการเกิดผีในสัญญาณโทรทัศน์

รูปที่ 9 และ รูปที่ 10 เป็นตัวอย่างของสัญญาณภาพโทรทัศน์ก่อนและหลังการกำจัดผี โดยสัญญาณผี 2 สัญญาณ ซึ่งมีความหน่วง เป็น 1.08 μs กับ 4.05 μs และขนาดเป็น –12 dB กับ –20 dB ของสัญญาณหลัก ซึ่งจะเห็นได้ว่าสัญญาณผี หรือภาพซ้อนที่เกิดขึ้นในรูปที่ 9 จะถูกกำจัดออก หลังจากผ่านเครื่องกำจัดผีแล้ว โดยในการทดสอบนี้ได้ทำการเปลี่ยนสัญญาณภาพ, ความหน่วง และขนาดของสัญญาณผี หลายๆแบบ พบว่าขนาดของสัญญาณผีที่เครื่องกำจัดผียังสามารถกำจัดได้ จะต้องไม่มากกว่า –6 dB

รูปที่ 9 สัญญาณภาพโทรทัศน์ก่อนกำจัดผี

รูปที่ 10 สัญญาณภาพโทรทัศน์หลังกำจัดผี

5. สรุป

เราได้พัฒนาเครื่องกำจัดผีในสัญญาณโทรทัศน์ระบบ PAL ขึ้น ด้วยการการสังเคราะห์วงจรกรองเอฟไออาร์ลงบนเอฟพีจีเอ โดยนำเทคนิคการใช้ทรัพยากรร่วมมาช่วย ทำให้ประหยัดทรัพยากรในส่วนของวงจรกรองไปได้มากกว่า 6 เท่า และผลการทดสอบทำให้เห็นว่า เครื่องกำจัดผีที่พัฒนาขึ้น สามารถกำจัดผีที่อยู่ในช่วง –1 μs ถึง +20 μs ได้

ในโครงงานนี้ ได้ทำการทดสอบแต่สัญญาณโทรทัศน์ภายในห้องทดลอง ซึ่งเป็นสัญญาณที่มีคุณภาพดี มีสัญญาณรบกวนน้อย ดั้งนั้นในขั้นต่อไปจะได้ทำการทดสอบกับสัญญาณที่มีสัญญาณรบกวนด้วย เพื่อทดสอบความทดทานต่อสัญญาณรบกวน

6. กิตติกรรมประกาศ

ขอขอบคุณ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์ และคอมพิวเตอร์แห่งชาติ (NECTEC) และโครงการเงินก้นถุงของจุฬาลงกรณ์มหาวิทยาลัย ที่ได้ให้ทุนวิจัยสนับสนุนโครงการนี้เป็นอย่างดี

เอกสารอ้างอิง

[1] Mary Louise Bucher. “Simulation of multipath fading/ghosting for analog and digital television transmission in broadcast channels.” IEEE Transactions on broadcasting Vol. 38 (December 1992) : 256-262

[2] Sims, H.V. Principles of PAL Colour Television and Related Systems. 2nd ed. London : lliffe books, 1969

[3] Recommendation ITU-R BT.1142-2. “Reference signals for ghost cancelling in analogue television system”

[4] Xilinx Inc. “Multiply Generator v3.1” May 11, 2001

[5] Simon Haykin. Adaptive Filter Theory. 3rd ed. New Jersey : Prentice-Hall, 1996

พหล ศิริเหลืองทอง สำเร็จการศึกษาระดับวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2541 ปัจจุบันกำลัง ศึกษาต่อในระดับปริญญามหาบัณฑิต สาขาวิศวกรรมไฟฟ้า แขนงวิชาวิศวกรรมไฟฟ้าอิเล็กทรอนิกส์ ที่จุฬาลงกรณ์มหาวิทยาลัย ได้รับรางวัลชนะเลิศการแข่งขันการประกวดการออกแบบวงจรรวมแห่งชาติครั้งที่ 2 (The National IC Design Contest 2001: NIC2001) ซึ่งจัดขึ้นโดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

วันเฉลิม โปรา สำเร็จการศึกษาระดับวิศวกรรมศาสตร์บัณฑิต และระดับปริญญามหาบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2535 และ 2538 ตามลำดับ จากนั้นได้รับราชการที่ภาควิชาวิศวกรรมไฟฟ้า และได้ทุนจากทบวงมหาวิทยาลัยเพื่อศึกษาต่อ และสำเร็จการศึกษาขั้นปริญญาเอกที่ Imperial College, University of London ในปี 2543 งานวิจัยที่อยู่ในความสนใจได้แก่ การออกแบบ และประยุกต์วงจรเชิงเลข การประมวลผลสัญญาณเชิงเลข เป็นต้น

A TV GHOST CANCELLER USING FPGA-BASED FIR FILTERS

Wanchalerm Pora and Pahol Siriluangtong

Electrical Engineering Department
Chulalongkorn University
Bangkok, Thailand 10330
Phone: +66-2218-6488
Email: Wanchalerm.P@Chula.ac.th

Electrical Engineering Department
Chulalongkorn University
Bangkok, Thailand 10330
Phone: +66-2218-6537
E-mail: jack@digital.ee.eng.chula.ac.th

ABSTRACT

This paper presents a PAL TV ghost canceller that employs a ghost cancellation reference signal (GCR signal) recommended in an ITU standard. The LMS algorithm is chosen to find the channel inverse, which is estimated by a 256-tap FIR filter. The emphasis is upon resource sharing technique, which increases utilization of multipliers by eight times. To test the canceller, a TV ghost emulator is constructed. From subjective tests, the canceller can remove the ghost whose power is lower than -6 dB compared to that of the main signal, providing its delay is lower than 10 μs.

1. INTRODUCTION

Even though the home theater concept has driven TV technologies a long way off, the most popular source of TV applications is still via analog broadcasting, which is subject to multipath propagation [1]. The signal from each path reaches TV receivers with different delays and powers; hence the duplicate horizontally-shifted pictures on the TV screens. This phenomenon is well-known as "ghost."

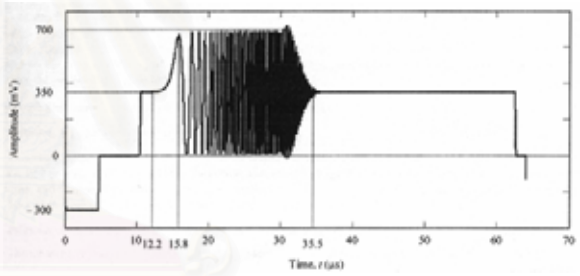
The ghost cannot be removed from the main picture using only conventional electronics. Signal processing must be incorporated, channel equalization in particular. Note that there were many attempts to implement ghost cancellers in the past twenty years [2]. However, they were very costly due to the cost of high performance signal processors. Ghost cancellation systems have, therefore, never been popular.

Advanced technologies have greatly increased the performance-cost of digital ICs. It is now worth reexamining the structure of a ghost canceller. The purpose of this paper is to propose a ghost cancellation system, which may be included within mid-range to high-end TV receivers. Its main components are a DSP processor and an FPGA that functions as a high-speed, long filter. Since DSP chip and FPGA are both reprogrammable, this combination provide other features, when equalization is not needed, such as noise cancellation, DTS decoder, etc.

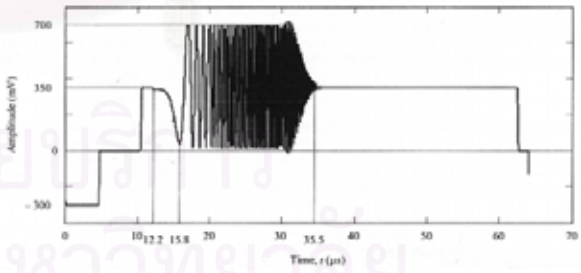
2. GHOST CANCELLATION REFERENCE (GCR) SIGNAL

In a ghost cancellation system, a GCR signal must be inserted onto a TV signal during its vertical blanking interval [3] at a TV broadcaster. The ghost cancellers at the TV receivers exploit the knowledge of how the known GCR signal responds to the effect of the channel. They can then find an inverse of such effect that essentially removes the ghost from TV signals.

There were also many attempts to find "good" GCR patterns. Eventually, ITU gathered a few good GCR patterns in the Recommend ITU-R BT.1124 [4].



(a) Positive polarity



(b) Negative polarity

Figure 1 Patterns of the GCR-C Signal

In this research, we adopt an ITU standard GCR-C signal whose patterns are shown in Figure 1. The positive and negative polarity signals are alternately inserted onto

line 318 of the composite video signal (CVS). Its figure looks alike that of a sinusoidal signal whose frequency increases temporally. This GCR signal has flat frequency response and smooth phase characteristic within a TV signal bandwidth. Moreover, its spectral characteristic is not sensitive to selections of sampling frequency and word length of A/D converters [5].

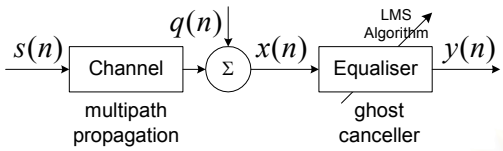


Figure 2 Signal propagates through channel and equalizer

3. EQUALIATION SCHEME

It is well known that a received signal, $x(n)$ traveled through a multipath propagation channel, as shown in Figure 2, can be estimated in discrete-time domain as

$$x(n) = \sum_{i=0}^{N} c_i s(n-i) + q(n) \qquad (1)$$

where c_i, $i = 0,...,N$ are parameters of the channel, $s(n)$ is the transmitted signal and $q(n)$ is the additive noise. The channel length (No. of taps) is denoted by N.

Without equalization, the received signal, $x(n)$ will cause ghost image on TV screens, i.e., $x(n)$ is a ghosted signal. Intuitively, the desired response of the equalizer (or ghost canceller in this case) shall reverse the channel effect without amplifying noise.

If an FIR filter is used to estimate the inverse of the channel then, the equalized (deghosted) signal, $y(n)$ is

$$y(n) = \sum_{j=0}^{M} w_j x(n-j) \qquad (2)$$

where w_j, $j = 0,...,M$ are parameters of the filter and M denotes the filter length.

The LMS algorithm is chosen to find such parameters due to its computational complexity and robustness [6]. The LMS adjust the parameters in a way that the equalizer output approaches the reference signal (only) when $x(n)$ is the result of the channel on the GCR signal. In other words, the filter cancels the channel effect.

4. PROPOSED TV GHOST CANCELLER

A TV ghost canceller, as shown in Figure 3, consists of a sync separator, an A/D and a D/A converters, a digital signal processor and an FPGA which functions as an FIR filter and a GCR separator. Each part will be described as follows:

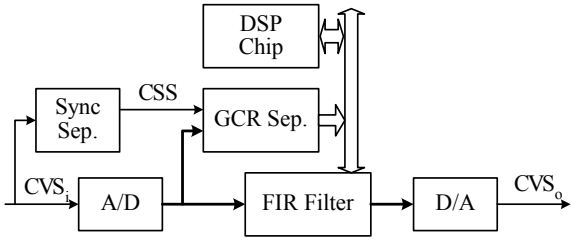


Figure 3 Structure of the proposed ghost canceller

4.1 Sync Separator (LM1881)

LM1881 separates composite sync. signal (CSS) from CVS input (CVS_i). Moreover, its internal clamp circuit always keeps DC voltage of the sync. tip level, the lowest voltage level, of the CVS_i at 1.5 V.

4.2 A/D converter (THS1031)

Typically, a CVS signal is $1V_{p\text{-}p,}$ so the clamped CVS_i may vary within a range of 1.5-2.5V. The converter is, therefore, set to convert an analog input signal to 8-bit unsigned data with middle reference voltage at 2V. To maximize the length of the filter in continuous-time domain, whilst avoiding the alias effect, the sampling rate is chosen to be 12 Msamples/s.

4.3 GCR data separator

Implemented in a Xilinx FPGA [7], GCR data separator prepares GCR sequence for the DSP chip. It synchronizes the horizontal and vertical time bases with the CSS signal [2]. It then keeps the effect of the channel on the GCR by storing some parts of line 317 and 319 and the whole line of 318 in its FIFO. Recall that the transmitter inserts the GCR signal onto line 318 of the CVS signal.

4.4 Digital signal processor (TMS320C6211)

The DSP Chip performs the LMS algorithm. It gets the channel effect on GCR sequence by reading FIFO inside the GCR separator, searches the optimized parameters, and then readjusts the FIR parameters.

To improve picture stability, DSP chip will stop searching if summation of squared error [6] is greater than a threshold, and will start searching again when such value is higher than another higher threshold.

4.5 D/A converter (THS5641)

After equalization or ghost cancellation, deghosted digital data will be converted back to analog domain. THS5641 with an I-to-V amplifier converts 8-bit digital data to $1V_{p\text{-}p}$ analog CVS_o signal.

4.6 FIR Filter with Resource Sharing Technique

It is desirable to have an FIR filter whose length is as long as possible in order to cancel the ghost that has long

delay. In fact, to estimate the channel inverse with an FIR filter very well, the length of that FIR should be doubly longer than that of the channel [6], i.e., M (in Eq. (2)) should be greater than $2N$ (in Eq. (1)).

Since the filter operation is in a form of sum of products (see (2)), so each filter tap invokes multiplication and accumulation (MAC). If one wants to implement a long filter running at high speed only by DSP processors one would require lots of them. For example to implement a 256-tap filter running at 12 Msamples/s (3-billion MACs/s), one would need at least 20 TMS320C6211s, hi-end DSP processors. This is obviously not feasible.

Dedicated hardware such as FPGA seems to be a better choice. From its implementation report shown in Table 1, Xilinx XCV300E spends 54% of its hardware resource (about 160,000 gates) to construct a 32-tap FIR filter. This filter can process data up to 75.7 Msamples/s. However, if pipeline multipliers [8] are employed, it will spend a bit more resource but it can work almost doubly faster than the conventional one.

Table 1 Resource- performance of 32-tap FIR filter, implemented on XCV300E

Technique	Resource	Speed
Conventional Multiplier	54%	75.7 MHz
Pipeline Multiplier	60%	144.4 MHz

Since the speed of the filter implemented on the FPGA is much faster than 12 Msamples/s required. It is therefore possible to redesign the filter such that every 12 MHz multiple filter taps access the same multiplier and accumulator at different time, i.e., time multiplex. This is a concept of resource sharing. It will lengthen the filter significantly with small additional resource.

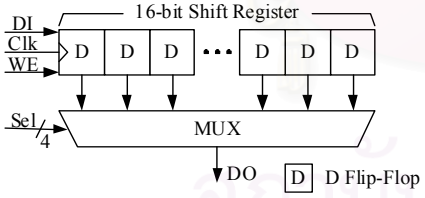


Figure 4 Structure of SRL16x1E, a primitive component of XCV300E

4.6.1 8-bit 8-to-1 Data Multiplexer and Shifter

We decide to reuse the MAC unit eight times over 12MHz cycle, so there must be a circuit that select 1-of-8 pair k, $k = 8,...,1$ in order to prepare a pair of sampled data, $x(n-8j+k)$ and a filter parameter w_{8j-k}, $j = 1,2,...$ and $8j-k \in [0,M]$ for the MAC unit number j.

A primitive component, SRL16x1E, of the selected FPGA perfectly satisfies this requirement. Illustrated in Figure 4, it is composed of a 16-bit shift register and a multiplexer. However, only lower eight bits are used in this design. A group of eight SRL16x1E, SRL16x8E, can then represent 8-bit of eight $x(n-8j+k)$ or w_{8j-k}.

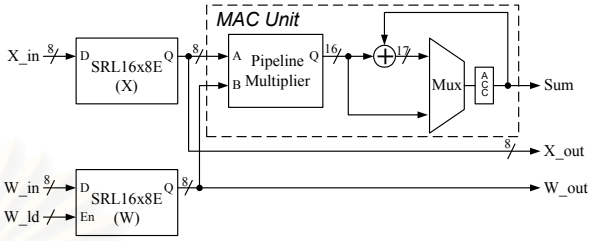


Figure 5 8-tap FIR filter using resource sharing

4.6.2 8-tap FIR Filters

An 8-tap filter j shown in Figure 5 is composed of two SRL16x8E and a MAC unit. One SRL16x8E is to keep $x(n-8j+k)$ and another is to keep w_{8j-k}. Intuitively, after alternating k eight times the accumulator output shall have the value of $\sum_{k=8}^{1} w_{8j-k}x(n-8j+k)$. Hence, by this configuration an 8-tap of an FIR filter can be achieved using only one MAC unit.

After changing k seven times, upon the last time when n increases $(n = n+1)$, the eight registers inside SRL16x8E will be shifted once (every 12MHz), so that the new $x(n-8j+8)$ comes in from the (j+1)th 8-tap filter and its $x(n-8j+1)$ will become $x(n-8j+8)$ of (j-1)th 8-bit filter.

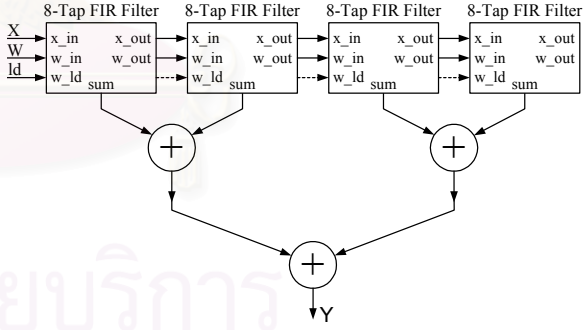


Figure 6 32-tap FIR filter

Value of $w_j, j = 0...M$ can be updated by shifting too. When DSP chip wants to adjust filter parameters, it will activate W_ld signal and send w_M first and w_0 last, at each rising edge clock, w_j will propagate via shift register and seat in its designated position eventually after $M+1$ clocks.

To complete the whole filter, each 8-tap FIR filter is connected sequentially. Shown in Figure 6 is an example

of a 32-tap FIR filter

A design of filters using resource sharing technique can save lots of resource of FPGA. As shown in Table 2, a 256-tap filter consumes only 73% of resource in XCV300E or about 219,000 gates and the speed of the filter can run up to 141.9 MHz. Since the resource is reused eight times, the data rate can be up to 17.7 Msamples/s.

Table 2 Resource-performance comparison between conventional FIR filter and FIR filter using resource sharing implemented on XCV300E

Technique	Length (taps)	Resource	Speed (MHz)	
			Circuit	Data
Conventional	32	54%	75.7	75.7
Pipeline Multiplier and Resource sharing	32	9%	147.1	18.3
Pipeline Multiplier and Resource sharing	256	73%	141.9	17.7

Finally, after synthesized and implemented together with the GCR separator, Xilinx Foundation 3.3i software reports that these circuits use up 99% of slices (a slice is a group of primitive components of Xilinx FPGA.)

5. PERFORMANCES

The effect of ghost can vary in many patterns depending upon the terrain of TV broadcasters and that of TV receivers. Hence we developed a ghost generator which consists of 64 270ns-delay-lines, amplifiers/attenuators and a combiner in order to emulate such effect thoroughly. However, the longest delay this emulator can generate is 17.3 μs. In unusual terrain, the ghost can have longer delay than this.

Figure 7 shows a deghosted TV screen, which demonstrates a visual test. In this example, we used two ghosts, whose delays are 1.08 μs and 4.05 μs, and powers are -12dB and -20dB, compared to the main signal respectively.

Thorough subjective tests have been conducted. The results are categorized according to the range of ghost delays as follows:

Delays between -1.3μs to 16μs: The canceller can visually remove the ghost completely within a few seconds when the total power of all delays is less than -12dB (1/4), compared to that of the main signal.

Delay between 0-10μs: Limitation of the range helps the filter to estimate the channel inverse better. The total ghost power can be up to -3dB. However, if the ghost is stronger than -6dB, it may distort sync. signal too much such that sometimes the GCR separator cannot synchronize with the CVS signal, so it may take a few more seconds to find the optimal parameters

If the ghost is stronger or its delays are longer than stated above, the canceller may not remove the ghost completely; it may be able to remove strong ghost but induces ghost at another position. If the ghost is too much strong, TV receivers will lose sync.

Figure 7 TV screen after ghost has been removed

6. CONCLUSIONS

We developed A PAL TV ghost canceller using a DSP chip and an FPGA. Resource sharing technique was employed. From our experiments, the ghost canceller can visually cancel the ghost within a few seconds if the total ghost power is not more than -6 dB compared to the main signal and the delays are within a specific range.

7. FUTURE WORKS

We are developing DSP processors on FPGA so that the new system will consist only one or two FPGA and some technique that may lengthen the filter responses.

REFERENCES

[1] Mary Louise Bucher, “Simulation of multipath fading/ghosting for analog and digital television transmission in broadcast channels,” IEEE Transactions on broadcasting, Vol. 38 (December 1992), pp. 256-262

[2] K. Kim, J. Oh et al, “ New Ghost Cancellation System,” Consumer Electronics IEEE Inter. Conf. on 1994, pp. 288-289 1994

[3] H.V. Sims, “Principles of PAL Colour Television and Related Systems,” 2nd ed. , Hiffe books, 1969

[4] Recommendation ITU-R BT.1124-3. “Reference signals for ghost cancelling in analogue television system,” ITU 2001

[5] Report ITU-R BT.2018, “Study of the system C ghost canceling reference signal for the evaluation and correction of linear distortion in the television chain.” ITU 1998

[6] Simon Haykin, “Adaptive Filter Theory,” 3rd ed. New Jersey : Prentice-Hall, 1996

[7] Xilinx Incorporation. “The Programmable Logic Data Book 2000,” 2000.

[8] Xilinx Inc. “Multiply Generator v3.1,” May 2001

ประวัติผู้เขียนวิทยานิพนธ์

นายพหล ศิริเหลืองทอง เกิดวันที่ 9 กุมภาพันธ์ พ.ศ.2520 ที่จังหวัดกาญจนบุรี สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยในปีการศึกษา 2540 และสมัครเข้าเป็นผู้ช่วยวิจัยของห้องปฏิบัติการวิจัยระบบเชิงเลข คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เป็นเวลา 1 ปี จึงได้เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิตสาขาวิศวกรรมไฟฟ้า แขนงวิชาวิศวกรรมไฟฟ้าอิเล็กทรอนิกส์เชิงเลข ที่คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2542 ในระหว่างการศึกษาได้รับรางวัลชนะเลิศการแข่งขันการประกวดการออกแบบวงจรรวมแห่งชาติครั้งที่ 2 (The National IC Design Contest 2001 : NIC2001) ประเภทดิจิตอล ซึ่งจัดขึ้นโดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)