ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

SPECTRUM ANALYZER RANGE 0-1 MH_z BY VHDL AND FPGAs DEVICE

นักศึกษา

1. นายผลิศักดิ์ ตันเจริญ รหัสประจำตัว 39031319
2. นายศักดิศร วัชราคม รหัสประจำตัว 39031328
3. นายสนธยา วันชัย รหัสประจำตัว 39031329
4. นายณัฐสมทัย บุญทองโท รหัสประจำตัว 39031331

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1. อาจารย์โกศล ตราชู
2. อาจารย์อำพล ทองระอา
3. อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

คณะกรรมการสอบสวนปริญญานิพนธ์		ลายมือชื่อ
1. อาจารย์กิติพงศ์	มะโน	.....
2. อาจารย์โกศล	ตราชู	.....
3. อาจารย์อำพล	ทองระอา	.....
4. อาจารย์ไพบูลย์	พวงวงศ์ตระกูล	.....
5. อาจารย์ปิยะ	จิตธรรมมาภิรมย์	.....

วันเดือนปีที่สอบ วันที่ 14 เดือน ธันวาคม พ.ศ. 2540 เวลา 23.00 น. ถึง 24.00 น.

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

.....

(ผู้ช่วยศาสตราจารย์ เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่.....เดือน.....พ.ศ.....

ปริญญานิพนธ์

เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

SPECTRUM ANALYZER RANGE 0-1 MH_z BY VHDL AND FPGAs DEVICE

นายผลิศักดิ์ ตันเจริญ

นายศักดิศร วัชราคม

นายสนธยา วันชัย

นายณัฐสมทัย บุญทองโท

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์

เรื่อง เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

SPECTRUM ANALYZER RANGE 0-1 MH_z BY VHDL AND FPGAs DEVICE

ผู้จัดทำ

1. นายผลิศักดิ์ ตันเจริญ
2. นายศักดิศร วัชราคม
3. นายสนธยา วันชัย
4. นายณัฐสมทัย บุญทองโท

อาจารย์ที่ปรึกษา

ลงนาม.....

(อาจารย์โกศล ตราชู)

ลงนาม.....

(อาจารย์อำพล ทองระอา)

ลงนาม.....

(อาจารย์ไพบูลย์ พวงวงศ์ตระกูล)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม.....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

SPECTRUM ANALYZER RANGE 0-1 MH_z BY VHDL AND FPGAs DEVICE

วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาโปรแกรมภาษา VHDL
2. เพื่อศึกษาหลักการของเครื่องวิเคราะห์แถบความถี่
3. เพื่อศึกษาการทำงานของอุปกรณ์ FPGAs และวิธีการโปรแกรมอุปกรณ์ FPGAs
4. เพื่อออกแบบเครื่องวิเคราะห์แถบความถี่
5. เพื่อสร้างเครื่องวิเคราะห์แถบความถี่โดยใช้ภาษา VHDL
6. เพื่อนำเครื่องวิเคราะห์แถบความถี่ไปใช้งานในความถี่ย่าน 0-1 MH_z

ประโยชน์ที่คาดว่าจะได้รับ

1. มีความรู้ในเรื่องการเขียนโปรแกรมภาษา VHDL
2. มีความรู้เรื่องหลักการของเครื่องวิเคราะห์แถบความถี่
3. มีความรู้และเข้าใจหลักการทำงานของอุปกรณ์ FPGAs
4. มีความรู้ความสามารถในการโปรแกรมอุปกรณ์ FPGAs
5. ได้วงจรเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z
6. ได้เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z ซึ่งเป็นเครื่องต้นแบบ
7. สามารถนำเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z ไปใช้งานได้จริง

เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

นายผลิศักดิ์ ตันเจริญ
นายศักดิศร วัชราคม
นายสนธยา วันชัย
นายณัฐสมทัย บุญทองโท

อาจารย์ที่ปรึกษา
อาจารย์โกศล ตราชู
อาจารย์อำพล ทองระอา
อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

ปีการศึกษา 2540

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอหลักการออกแบบเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs ซึ่งสามารถวิเคราะห์แถบความถี่ย่าน 0-1 MH_z และแสดงผลเป็นแถบความถี่และตัวเลขออกทางจอภาพของโทรทัศน์ขาว-ดำ ซึ่งในการวิเคราะห์หาแถบความถี่จะใช้อัลกอริทึมของ ฟัซท ฟูเรียร์ ทรานฟอร์ม (FFT) ผลการทดสอบเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z ค่าแถบความถี่ที่วัดได้มีความเที่ยงตรงเป็นที่น่าพอใจ

SPECTRUM ANALYZER RANGE 0-1 MH_z BY VHDL AND FPGAs DEVICE

MR.PLISAK TANCHALERN

MR.SAKDISOND WATCHARAKOM

MR.SONTAYA WANCHAI

MR.NUTTHASOMTAI BOONTONGTO

ADVISORS

MR.KOSON TRACHU

MR.AMPHON THONGRA-AR

MR.PAIBOON PONGWONGTRAGULL

1997

ABSTRACT

This thesis presents the design of spectrum analyzer rang 0-1 MHz using VHDL and FPGAs devices. It can analyze signal from 0-1 MHz and show the spectrum and data on television. The spectrum analyzer 0-1 MHz uses Fast Fourier Transform (FFT) algorithms to analyze signal.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์นี้สามารถสำเร็จลุล่วงด้วยดี จากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน นอกจากนี้ยังได้รับความอนุเคราะห์จากอาจารย์ที่ปรึกษาปริญญานิพนธ์และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมศาสตร์ทุกท่านในการให้คำปรึกษาแนะนำและความช่วยเหลือต่าง ๆ ตลอดจนให้โอกาสในการทำปริญญานิพนธ์อย่างเต็มที่ ทั้งด้านเวลาสถานที่ เครื่องมือและอุปกรณ์ต่าง ๆ และขอขอบคุณบุพการีผู้ให้กำเนิดที่ให้โอกาสในทางการศึกษา เพื่อน ๆ ตลอดจนผู้ที่เกี่ยวข้องทุกท่านที่ให้คำแนะนำต่าง ๆ และเป็นกำลังใจในการทำปริญญานิพนธ์ จนสำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

เครื่องวิเคราะห์แถบความถี่ (Spectrum Analyzer) ในปัจจุบันมีความละเอียดในการวัดสูง สามารถใช้วัดแถบความถี่ได้กว้าง ความถี่สูงสุดได้อาจมีค่าหลายจิกะเฮิรตซ์ เพราะว่าอัลกอริทึมที่ใช้ในการวิเคราะห์สัญญาณแบบ Fast Fourier Transform (FFT) มีขนาดเล็กลงและทำงานได้ที่ความเร็วสูง

อุปกรณ์ FPGAs (Field Programmable Gate Arrays) ซึ่งเป็นอุปกรณ์ลอจิกประเภทโปรแกรมได้ที่มีความจุสูง และสามารถรองรับการโปรแกรมด้วยภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) ซึ่งเป็นภาษาที่ใช้สำหรับบรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลข (Digital Circuit) ของอุปกรณ์ทางด้านลอจิก อีกทั้งยังมีความเร็วในการทำงานที่สูงมาก

การออกแบบวงจรเชิงเลขนั้นแบ่งออกเป็น 2 กลุ่ม คือ ฟลูลี่คัสตอม (Full Custom) และแบบเซมิคัสตอม (Semi Custom) โดยการออกแบบแบบฟลูลี่คัสตอมนั้นเมื่อทำการออกแบบวงจรเชิงเลข โดยใช้กฎการออกแบบวงจรรวมของโรงงานผู้ผลิตไอซี และทำการสังเคราะห์จนได้วงจรระดับเกท (Gate Level Schematic) แล้วต้องนำไปทำเป็นวงจรในระดับทรานซิสเตอร์ (Transistor) ในขั้นตอนสุดท้ายจะต้องนำไปให้โรงงานผลิตออกมาเป็นไอซีสำเร็จรูปให้ แต่แบบเซมิคัสตอมนั้นเราสามารถดำเนินการเองได้จนสิ้นสุดกระบวนการออกแบบ ซึ่งทางกลุ่มก็ได้เลือกใช้แบบ เซมิคัสตอมในการออกแบบ โดยทำการออกแบบวงจรเชิงเลขด้วยภาษา VHDL (Very High Speed Itegrated Circuit Hardware Description Language) และสังเคราะห์จนได้วงจรในระดับเกทแล้ว จึงนำวงจรที่ได้มาโปรแกรมลงบนอุปกรณ์ประเภทที่สามารถโปรแกรมได้ (Programmable Logic Devices) เช่นอุปกรณ์ FPGAs (Field Programmable Gate Arrays) ซึ่งทางคณะผู้จัดทำได้เลือกใช้ในโครงงานนี้

ภาษา VHDL เป็นภาษาที่ใช้สำหรับการบรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลข (Digital Circuit) โดยตัวภาษานั้นสามารถบรรยายการทำงานของวงจรเชิงเลขได้หลาย ๆ ลักษณะ เราสามารถบรรยายถึงความสัมพันธ์ระหว่างสัญญาณอินพุทและเอาท์พุท หรือที่เราเรียกว่าการบรรยายแบบ Behavioral Description หรือบรรยายถึงลักษณะความสัมพันธ์ของสัญญาณต่าง ๆ ที่วิ่งอยู่ในวงจรเชิงเลขนั้น ซึ่งเรียกว่าการบรรยายแบบ Dataflow Discription หรือแบบสุดท้ายเป็นการบรรยายในระดับต่ำ นั่นคือการบรรยายว่าในวงจรมีอุปกรณ์ใดต่ออยู่บ้าง ซึ่งเรียกว่าการบรรยายแบบ Structure Decription การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL นั้น ผู้ออกแบบไม่จำเป็นต้องทราบถึงรายละเอียดของวงจรว่าประกอบด้วยอะไรบ้างเพียงแต่รู้ถึงการทำงานของวงจร ซึ่งในการออกแบบวงจรนั้น เราต้องทำการแบ่งวงจรออกเป็นส่วนย่อย ๆ แล้วเขียนโปรแกรมบรรยายการทำงานของแต่ละส่วนขึ้นมาด้วยภาษา VHDL จากนั้นทำการแปลภาษา (Compile) และจำลองการทำงาน (Simulate) เพื่อตรวจสอบดูว่าผลการทำงานตรงตามจุดประสงค์ที่ต้องการหรือไม่ ถ้าต้องการแก้ไขก็เพียงแต่เข้าไปแก้ไขที่โปรแกรมภาษา VHDL เท่านั้น แล้วทำการแปลภาษาและจำลองการทำงานใหม่ จนกว่าจะได้ผลการทำงานตามที่ต้องการจึงนำไปสังเคราะห์ (Synthesis) เพื่อให้ได้เป็นวงจรแสดงการเชื่อมต่อซึ่งประกอบด้วยเกท (Gate) ฟลิป-ฟลอป (Flip-Flop) และอุปกรณ์พื้นฐานต่าง ๆ หลังจากนั้นจึงนำไปโปรแกรมลงบนอุปกรณ์ FPGAs

อุปกรณ์ FPGAs จะรวมกลุ่มของเกทและอุปกรณ์โปรแกรมได้ของ PLDs (Programmable Logic Devices) เข้าด้วยกัน โดยภายในตัว FPGAs จะมีส่วนสำคัญ คือ เส้นทางการเชื่อมต่อระหว่างบล็อค ซึ่งเส้นทางเหล่านี้เราสามารถโปรแกรมได้จำนวนเกทที่เราสามารถนำมาใช้ได้มีอยู่ระหว่าง 600 ถึง 25,000 เกท ซึ่งสามารถนำมาใช้ประโยชน์ได้อย่างกว้างขวาง เพราะสามารถโปรแกรมให้ทำงานเป็นฟังก์ชั่นต่าง ๆ และยังสามารถโปรแกรมใหม่ได้

โครงงานนี้นำเสนอการประยุกต์ใช้อุปกรณ์ FPGAs เพื่อสร้างเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MHz ขึ้นมาใช้งานแทนเครื่องวิเคราะห์แถบความถี่ย่านความถี่ต่ำซึ่งหาได้ยากตามท้องตลาด โดยมีการแสดงผลบนจอโทรทัศน์ขาวดำและวิธีการคำนวณหาแถบความถี่โดยใช้อัลกอริทึม FFT ขนาด 1024 จุด

1.2 ขอบเขตของปริญญานิพนธ์

1. ใช้วิเคราะห์แถบความถี่ย่าน 0-1 MHz ได้
2. ใช้หลักการคำนวณหาแถบความถี่แบบ Fast Fourier Transform (FFT)
3. แสดงผลบนจอภาพโทรทัศน์ ขาว ดำ

1.3 เนื้อหาโดยสังเขป

บทที่ 1 บทนำ จะมีเนื้อหาเกี่ยวกับ ความเป็นมาและความสำคัญของปริญญานิพนธ์ ทฤษฎีและหลักการอย่างคร่าว ๆ เพื่อให้ผู้อ่านเข้าใจหลักการในการออกแบบ

บทที่ 2 ทฤษฎี และหลักการ กล่าวถึงภาษา VHDL ซึ่งได้แก่ ข้อกำหนดของภาษา VHDL องค์ประกอบพื้นฐานในภาษา VHDLในส่วนต่อมาจะกล่าวถึงอุปกรณ์ FPGAs ซึ่งได้แก่ ส่วนที่เป็นองค์ประกอบของลอจิก, ส่วนอินพุทและเอาท์พุทของอุปกรณ์ FPGAs, รายละเอียดการใช้งานของอุปกรณ์ FPGAs, ข้อควรระวังในการใช้อุปกรณ์ FPGAs, ในหัวข้อต่อมาจะกล่าวถึงทฤษฎีของโทรทัศน์เบื้องต้น ซึ่งเกี่ยวกับส่วนประกอบของภาพ, วิธีการสแกนและการหักเหของลำอิเล็กตรอน, เครื่องส่งและเครื่องรับโทรทัศน์, สัญญาณต่าง ๆ ที่ส่ง, การหาความจุของภาพดิจิตอล, ในหัวข้อต่อมาจะกล่าวถึงทฤษฎีการแปลงสัญญาณ ซึ่งมีเนื้อหาเกี่ยวกับ วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล และวงจรแปลงสัญญาณดิจิตอลเป็นแอนาลอก, ทฤษฎีของการสุ่ม

บทที่ 3 การออกแบบ การสร้าง และการทำงาน ในบทนี้จะกล่าวถึง ขั้นตอนการออกแบบ การทำงานของวงจรต่าง ๆ ที่เกี่ยวข้อง

บทที่ 4 การทดลอง และผลการทดลอง ในบทนี้จะกล่าวถึงวิธีการทดลองวงจรต่าง ๆ ที่เกี่ยวข้อง และผลการทดลองที่ได้

บทที่ 5 บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา จะกล่าวถึง การสรุปของการทำปริญญานิพนธ์, ปัญหาที่เกิดขึ้น, วิธีการแก้ไขปัญหาที่เกิดขึ้น, ข้อเสนอแนะและแนวทางในการนำปริญญานิพนธ์ไปพัฒนาต่อไป

บทที่ 2
ทฤษฎีและหลักการ

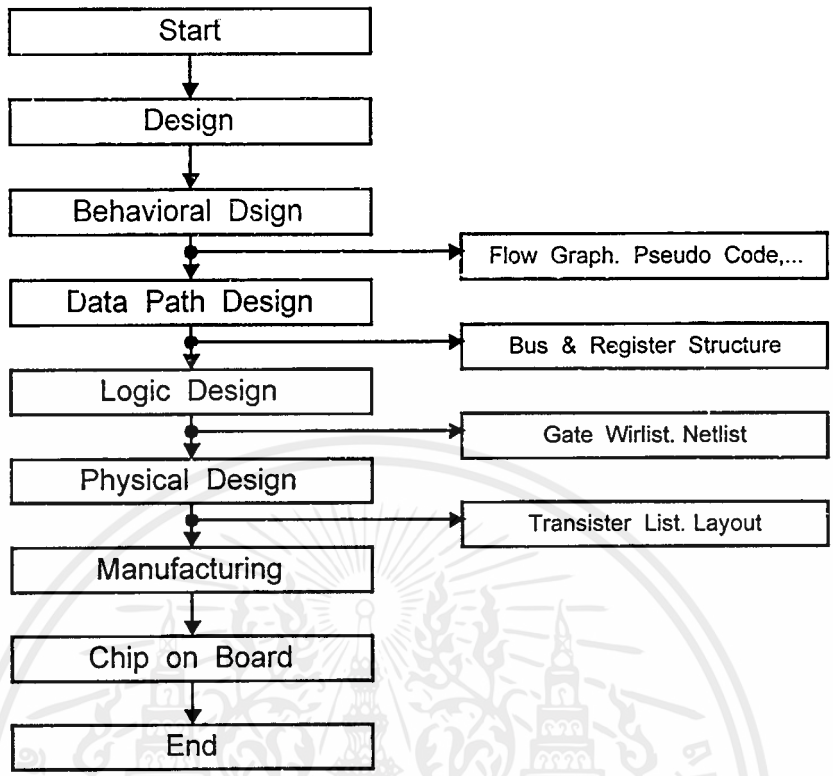
2.1 กล่าวนำ

ในขณะที่ขนาดและความซับซ้อนของระบบเชิงเลขได้เพิ่มมากขึ้น คอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD ย่อมาจาก Computer Aided Design) ถูกนำเข้ามาใช้ในกระบวนการออกแบบมากขึ้น อุปกรณ์และวิธีการออกแบบใหม่ ๆ ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบ และภาษาสำหรับบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL ย่อมาจาก Hardware Description Language) ก็เป็นเครื่องมืออีกอย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่อง เพื่อช่วยในการปรับปรุงกระบวนการออกแบบระบบเชิงเลข

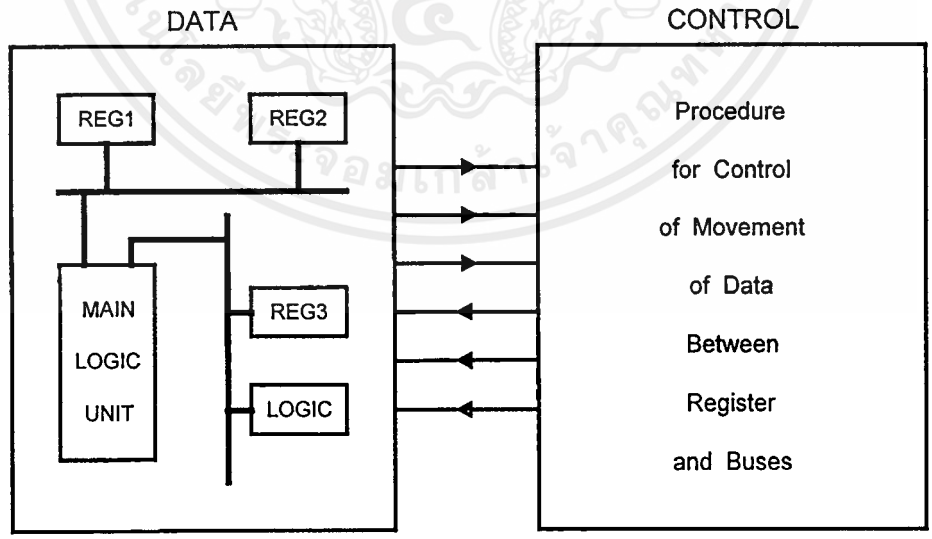
ในการออกแบบระบบเชิงเลข จะเริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้ จะต้องผ่านขั้นตอนต่าง ๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและก็เข้าสู่กระบวนการออกแบบในขั้นต่อไป

รูปที่ 2.1 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบเชิงเลขทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบ และทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นภายในขั้นตอนนี้จึงจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo Code) ก็ได้

ขั้นต่อไปจะเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์ (Register) และวงจรตรรก (Logic) ส่วนที่จำเป็น เพื่อจะนำมาประกอบกันเป็นระบบที่สมบูรณ์แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) แต่กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจร ตรรกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.2



รูปที่ 2.1 ขั้นตอนการออกแบบระบบเชิงเลข



รูปที่ 2.2 การออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกจะเป็นขั้นตอนต่อไป ซึ่งการออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิป-ฟลอปเป็นส่วนประกอบของอุปกรณ์ย่อยต่าง ๆ ได้แก่ รีจิสเตอร์เก็บข้อมูลบัสวงจรตรรกและส่วนควบคุมฮาร์ดแวร์ และในขั้นสุดท้ายจะออกมาเป็นเครือข่ายของการเชื่อมโยงระหว่างเกทและฟลิป-ฟลอปนั่นเอง ต่อมาเป็นขั้นตอนการเปลี่ยนเครือข่ายการเชื่อมต่อในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์ลีสและเลย์เอาท์ (Transistor List and Layout) ในขั้นตอนนี้เกี่ยวข้องกับการจัดวางเกทและฟลิป-ฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซลขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด การออกแบบตามวิธีที่กล่าวมานี้เรียกว่าการออกแบบแบบฟูลลี่คัสตอม

2.2 แนะนำภาษา VHDL

ในปี ค.ศ. 1981 สถาบันเพื่อการวิเคราะห์และป้องกัน (The Institute for Defense Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่ง เพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนก ๆ หนึ่งของสถาบันที่ทำงานเกี่ยวข้องกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Circuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐาน และมีการยอมรับกันอย่างกว้างขวางใน วงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านการกำหนดพฤติกรรมการทำงานของวงจร ให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งของวงจรเชิงเลขทั่วไปและในระบบที่แตกต่างออกไป เช่น พฤติกรรมการทำงานของระบบเรดาห์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้สามารถจะใช้ได้ตลอดในทุกระดับขั้นของการออกแบบ นั่นคือในกระบวนการออกแบบ ตั้งแต่ระดับสูง (System Level) จนถึงในระดับที่ต่ำกว่า (Lower Hardware Level) สามารถใช้ภาษาเดียว

กันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานรวมกันได้เป็นอย่างดี

2.2.1 ข้อกำหนดของภาษา VHDL

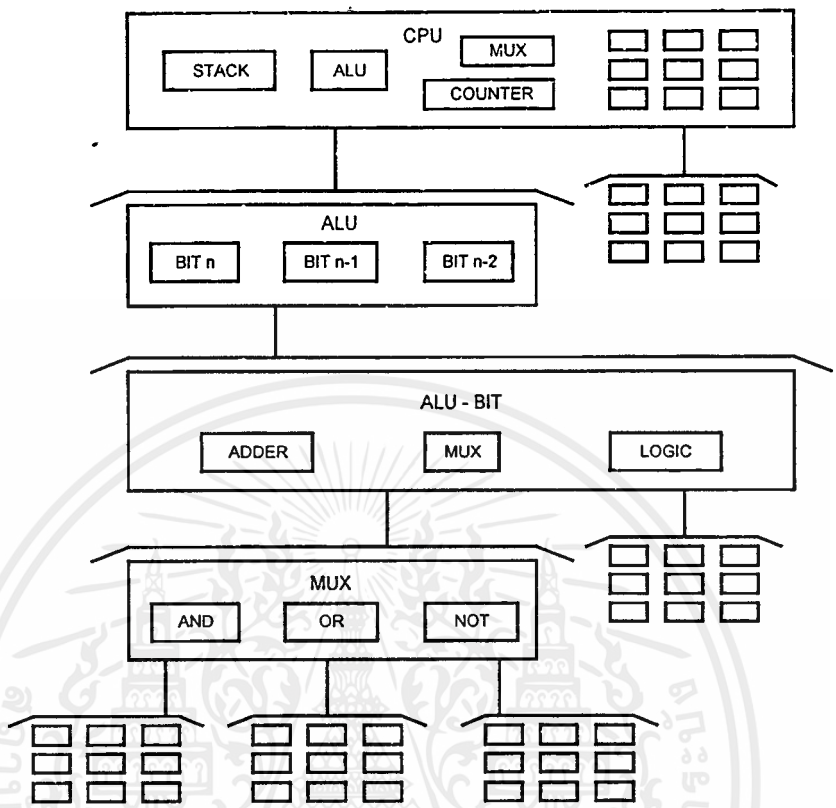
ในเอกสารของ DoD (The Unitad State Department of Defense) ซึ่งได้ออกมาในเดือนมกราคม ปี ค.ศ. 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

ลักษณะทั่วไป

กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึง ความสามารถในการอธิบายและออกแบบในระดับสูงความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้นภาษา VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ ตั้งแต่ระดับของระบบจนถึงระดับเกท เนื่องจากในการทำงานของระบบเชิงเลขจริงทุกองค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อม ๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งในภาษา VHDL ด้วย สำหรับภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้วความพร้อมเพรียงจะหมายถึงทุก ๆ คำสั่งองค์ประกอบเกท หรือวงจรตรรกจะนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อม ๆ กัน

การสนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่ง สำหรับการออกแบบที่มีหลายระดับ ในการออกแบบจะประกอบด้วย ส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบสามารถกำหนดได้ด้วยตัวเองหรือถูกกำหนดโดยโครงสร้างประกอบด้วยองค์ประกอบย่อย ๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานของมันด้วยตัวมันเอง และไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ดังรูป 2.3



รูปที่ 2.3 ตัวอย่างการออกแบบแบบลำดับขั้น

ไลบรารี

ภาษา VHDL ได้สนับสนุนการให้มีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารี หลังจากที่ได้ผ่านการแปลเรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่น ๆ สามารถนำไปใช้ได้ด้วย

ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกัน จะเป็นคุณสมบัติที่สำคัญของภาษา VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งเองเอาไว้เมื่อผู้ออกแบบได้กำหนดหน้าที่ และองค์ประกอบการทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงาน

ซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบไปด้วยโครงสร้างแบบ case,if-then-else และ loop การบรรยายแบบลำดับคำสั่ง ทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์สามารถกระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของภาษา VHDL ยังคงเป็นการทำงานแบบพร้อมเพรียงกันอยู่

การกำหนดคุณสมบัติ

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่น ๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์เช่นกัน ทั้งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ๆ ภาษาสำหรับการออกแบบที่ดีควรที่จะช่วยให้ผู้ออกแบบสามารถกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาดทางกายภาพ เวลา โหลดและเงื่อนไขทางสภาพแวดล้อมอื่น ๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL

ชนิดของข้อมูล

ภาษา VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิดบิท และบูลีนเท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับของการนับ (Enumerate type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเอง

โปรแกรมย่อย

ความสามารถในการใช้ฟังก์ชันและโพรซีเจอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งในภาษา VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิกและการกำหนดตัวกระทำต่าง ๆ ทั้งเก่าและใหม่ได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

การควบคุมเวลา

ภาษา VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาการส่งผ่านข้อมูลหรือสัญญาณ ได้ การตรวจสอบการออกแบบเกทหรือการหน่วงเวลาสามารถกระทำได้โดยการ กำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ และนอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้

การกำหนดแบบโครงสร้าง

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกระดับของการออกแบบการกำหนดโครงสร้างองค์ประกอบรวมที่เกิดจากองค์ประกอบย่อย ๆ ที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่ง

2.2.2 องค์ประกอบพื้นฐานในภาษา VHDL

รูปแบบพื้นฐานในการบรรยายถึงองค์ประกอบในภาษา VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำว่า ENTITY แล้วตามด้วยชื่อขององค์ประกอบและคำว่า IS ภายในบรรยายถึงพอร์ตการติดต่ออินพุทเอาท์พุทพอร์ตขององค์ประกอบส่วนคุณลักษณะและภายนอกอื่น ๆ เช่น เวลาและอุณหภูมิ ในส่วนของการกำหนดลักษณะในเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ ซึ่งหน้าที่ของการทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท และเอาท์พุท และพารามิเตอร์อื่น ๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อ ดังรูปที่ 2.4

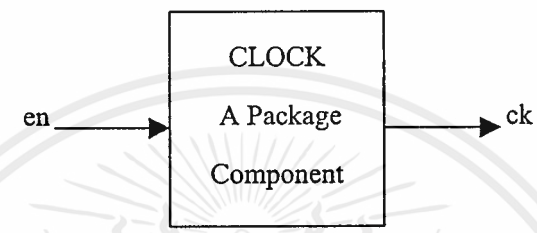
```
ENTITY component_name IS
        input and output port.
        physical and other parameters.
END component_name ;
```

```
ARCHITECTURE identifier OF conponent_name IS
        declarations.
BEGIN
        specification of the functionality of the component
        in terms of its input line and as influenced
        by physical and other parameters.
END identifier ;
```

รูปที่ 2.4 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับขั้นบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอก ดังตัวอย่างในรูปที่ 2.5 แสดงผังการทำงาน และการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา




```
ENTITY clock_component IS
        PORT(en:IN  BIT;  clk:OUT  BIT) ;
END clock_component ;
```

รูปที่ 2.5 ผังการทำงานและการบรรยายการเชื่อมต่อของ clock_component

การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายลงในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในจำนวนของอินพุท หรือที่อยู่ในรูปขององค์ประกอบแบบอื่น ๆ หรือองค์ประกอบทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.6 ซึ่งเป็นการบรรยายในเชิงพฤติกรรม en เป็นอินพุท และ clk เป็นเอาท์พุท PROCESS เป็นค่าเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรมภายในโพรเซส กำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น '0' ถ้าสัญญาณ en มีค่าเป็น '1' ค่าของ periodic จะถูกคอมพลีเมนต์ (Complement) และจะส่งค่าไปให้กับ clk ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเวลาเป็น 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS
    BEGIN
        PROCESS
            VARIABLE periodic : BIT := '0';
            BEGIN
                IF en = '1' THEN
                    periodic := NOT periodic ;
                END IF ;
                clk <= periodic ;
                WAIT FOR 1 ns ;
        END PROCESS ;
END behavior ;
```

รูปที่ 2.6 การบรรยายเชิงพฤติกรรมของ clock_component

โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ในภาษา VHDL สามารถเปรียบเทียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมชั้นสูงทั่ว ๆ ไป ค่าที่ถูกส่งกลับมาหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรง เช่น ถ้าเราใช้ฟังก์ชันแทนการกระทำในสมการบูลีน จะมีผลต่อวงจรตรรกจริง ๆ ในขณะที่เราใช้โปรแกรมในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 2.7 แสดงการใช้ฟังก์ชัน โดยกำหนดให้ x เป็นตัวแปรชนิดบิท แทนการกระทำในสมการบูลีน รูปที่ 2.8 แสดงโพรซีเจอร์ที่เป็นตัวกลับสัญญาณ

โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL ก็มีตัวกระทำทางคณิตศาสตร์และลอจิกเช่นเดียวกับภาษาซอฟแวร์ทั่วไปดังรูปที่ 2.9

```
FUNCTION hav (a,b,c : BIT) RETURN BIT IS
  VARIABLE x : BIT
    BEGIN
        x := ((NOT a) AND (NOT b) AND c) ;
         RETURN x ;
END hav ;
```

รูปที่ 2.7 ตัวอย่างฟังก์ชัน

```
LIBRARY mgc_portable ;
USE mgc_portable.qsim_logic.ALL ;
ENTITY inert_example IS
    PORT(in_a:IN  qsim_state;  out_b:OUT  qsim_state ;)
END invert_excample
ARCHITECTRUE alg  OF  invert_excample  IS
    PROCEDURE qsim_invecter(SIGNAL  bit_in:IN  qsim_state ;
            SIGNAL bit_out:OUT  qsim_state) IS
        BEGIN
          IF bit_in='1'THEN
              bit_out<='0';
          ELSEIF  bit_in='0' THEN
              bit_out<='1';
          ELSE
              bit_out<='X';
          END IF ;
END qsim_state ;
        BEGIN
            qsim_invecter(in_a,out_b) ;
END alg ;
```

รูปที่ 2.8 ตัวอย่างโพรซีเจอร์

PREDEFINED OPERATORS
LOGICCAL OPERATORS : NOT AND OR NAND NOR XOR OPERAND TYPE : BIT BOONLEAN RESULTI TYPE : BIT BOOLEAN
RELATIONAL OPERATORS : = /= < <= > >= OPERAND TYPE : any type RESULT TYPE : Boonlean
ARITHMETIC OPERATORS : + - * / MOD REM ABS OPERAND TYPE : INTEGER REAL Physical RESULT TYPE : INTEGER REAL Physical
CONCANTENATION OPERATORS : & OPERAND TYPE : array of any type RESULT TYPE : array of any type

รูปที่ 2.9 ตัวกระทำในภาษา VHDL

2.3 โครงสร้างภายในของอุปกรณ์ FPGAs

FPGAs จัดเป็นวงจรรวมเฉพาะกิจชนิดหนึ่งโปรแกรมเป็นวงจรเชิงเลขใด ๆ ก็ได้ เช่นเดียวกับ EPLD ต่างกันที่ EPLD โปรแกรมลงบน EPROM ภายใน และสามารถโปรแกรมใหม่ได้หลังจากนำไปลบด้วยแสง UV แต่ FPGAs โปรแกรมลงบนสแตติกแรมภายในด้วยข้อมูลที่อยู่ภายนอก และสามารถโปรแกรมใหม่ได้โดยการรีเซตด้วยสัญญาณไฟฟ้านอกจากนั้น FPGAs ยังประหยัดไฟและความความจุวงจรสูง (จำนวนเกทมาก) ได้อีกด้วย

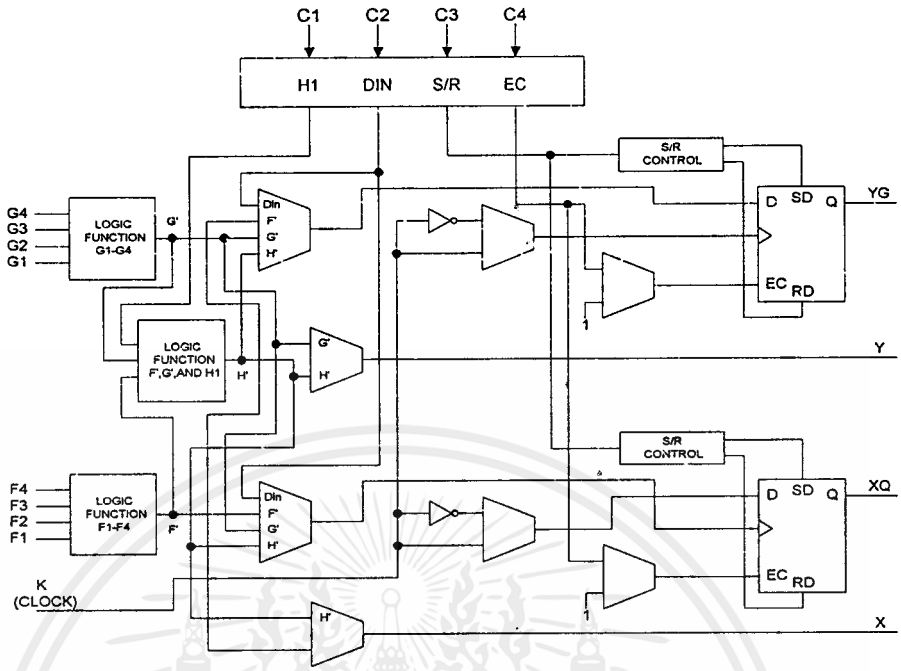
วงจรรวมชนิดที่ใช้ในโครงงานนี้ผลิตโดยบริษัทไซลิงค์ (Xilinx) ซึ่งเป็นบริษัทที่ทำการค้นคว้าร่วมกับบริษัทเอ็มเอ็มไอ (MMI) สร้างเป็นกลุ่มของเกทจำนวน 600-25,000 เกท

ตารางที่ 2.1 ตารางประมาณการนับเกทของเกทพื้นฐาน

Gate	Equipvalent gate count	Gate	Equipvalent gate count
INV	1	RS Latch	3
NAIINR2	1	D Latch	4
NAIINR3	2	D Latch with CLR	5
NAIINR4	2	D Latch with PRE	5
NAIINR6	5	D Latch with PRE/CLR	6
NAIINR8	6	D F/F	6
NAIINR9	7	D F/F with CLR	7
NAIINR12	8	D F/F with PRE	7
NAIINR16	11	D F/F with PRE/CL	8
BUFF	2	JK F/F with CLR	9
ANIIOR2	2	JK F/F with PRE	12
ANIIOR3	2	JK F/F with PRE/CL	13
ANIIOR4	3	T F/F with CLR	8
XOR2	3	T F/F with PRE	8
XNOR2	3	T F/F with PRE/CL	9

NAIINR2 หมายถึงเกท NAND2 หรือ NOR2

FPGAsมีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกทอาเรย์ (GAL,Gate Array Logic) มากสามารถโปรแกรมและลบคอนฟิกกูเรชัน (Configuration) ภายในสแตติกแรม (Static RAM) ได้โดยใช้กระแสไฟฟ้า ซึ่งทำการโปรแกรมได้โดยดึงข้อมูลฐานสิบหกมาจากภายนอก เช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD, PAL ที่มี EPROM อยู่ในตัว ภายใน FPGAs จะจัดเรียงเป็นลอจิกเซลล้อมรอบภายนอกด้วยอินพุทเอาท์พุทเซล FPGAs ตัวแรกที่ผลิตโดยบริษัทไซลิงค์คือเบอร์ XC2064 (2000Family) ประกอบด้วยเซลเรียงกันเป็นเมตริกซ์ (Matrix) เป็นจำนวน 64 เซล หลังจากนั้นผลิต FPGAs ตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างซับซ้อนขึ้นสามารถเพิ่มจำนวนเกทได้มากขึ้นและดีขึ้น แต่ละเซลเรียกว่า CLB (Configurable Logic Block)



รูปที่ 2.10 แผนผัง CLB ของตระกูล 4000

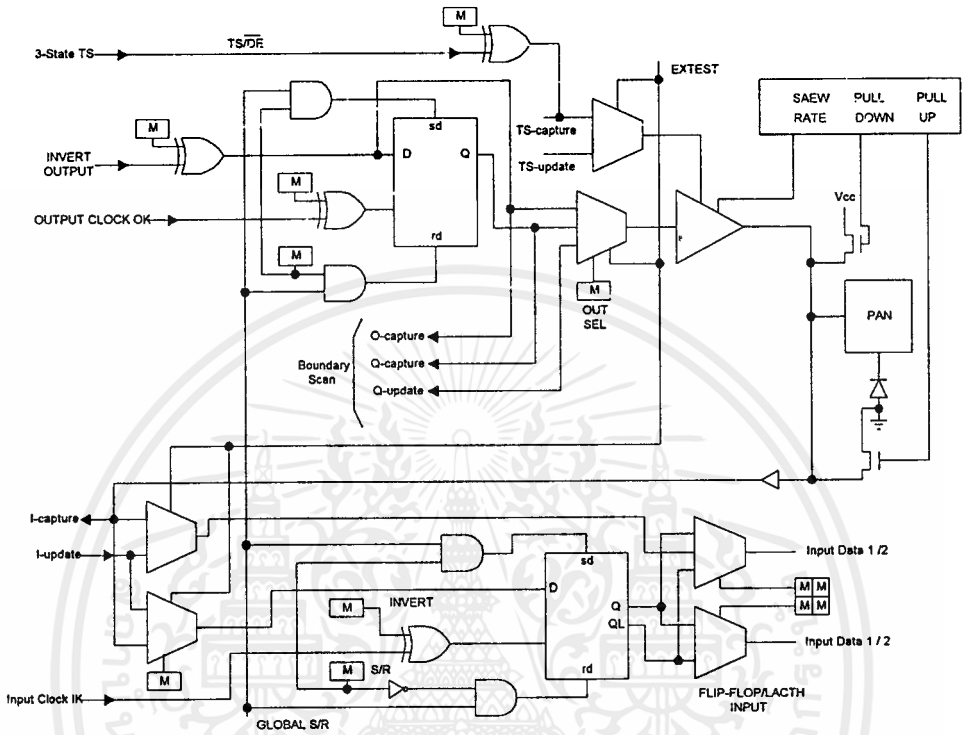
2.3.1 ส่วนที่เป็นองค์ประกอบของลอจิก (Configurable Logic Block)

CLB จะจัดเรียงกันเป็นแบบเมตริกซ์แบบอาเรย์ขนาด M*N การออกแบบนั้นสามารถทำได้โดยการจัดการวาง CLB และต่อเชื่อมขาของ CLB ให้ต่อถึงกัน เราสามารถจัด CLB ให้เชื่อมต่อถึงกันได้โดยการทำด้วยมือหรือให้โปรแกรมที่สนับสนุน FPGAs ทำให้โดยอัตโนมัติ หรือโดยวิธีของมันเองสำหรับไฟล์ที่ได้จากโปรแกรมเหล่านี้ เราจะเรียกว่าไฟล์ที่กำหนดการวางของอุปกรณ์ (Configuration) ซึ่งจะบรรจุโครงร่างภายในของ CLB ตามความเหมาะสม อีกด้านหนึ่งไฟล์กำหนดการวางอุปกรณ์นั้นจะเป็นไฟล์กระแสข้อมูล (Bit Stream) ซึ่งสามารถใช้โปรแกรมหน่วยความจำภายในของ FPGAs ได้ สำหรับรูปที่ 2.10 แสดง CLB ของ FPGAs ตระกูล 4000

2.3.2 ส่วนอินพุทและเอาท์พุทของอุปกรณ์ FPGAs

รอบนอกของ FPGAs จะประกอบไปด้วย IOBs ประมาณ 64 ถึง 144 ตัว ซึ่งจะขึ้นอยู่กับตระกูลของ FPGAs ซึ่ง IOBs จะเป็นตัวเชื่อมต่อระหว่างภายในกับภายนอกของวงจรลอจิกของ FPGAs ลักษณะของ IOBs จะมีลักษณะ 2 ทิศทาง สามารถ

โปรแกรมให้เป็นอินพุทหรือเอาท์พุทก็ได้ สำหรับรูปที่ 2.11 แสดง IOBs ของ FPGAs ตระกูล 4000



รูปที่ 2.11 แผนผัง IOBs ของตระกูล 4000

2.3.3 รายละเอียดการใช้งานของอุปกรณ์ FPGAs

FPGAs สามารถทำงานได้หลายลักษณะ โดยเราสามารถกำหนดได้ที่ขาสัญญาณ M0 M1 M2 ดังแสดงในตารางที่ 2.2 ในลักษณะมาสเตอร์พาราเรล (Master Parallel Mode) รับโปรแกรมคอนฟิกทีละ 1 ไบท์ (Byte) จากหน่วยความจำภายนอกที่เป็นแบบขนานโดยสามารถรับโปรแกรมคอนฟิก (Config) จากแอดเดรส (Address) ต่ำหรือสูงก่อนก็ได้การต่อในลักษณะเพอริเฟอรัล (Peripheral) จะรับโปรแกรมคอนฟิกทีละ 1 ไบท์จากไมโครโปรเซสเซอร์ โดยสามารถโต้ตอบกันได้ว่าพร้อมหรือไม่ที่จะรับข้อมูลต่อไป การต่อลักษณะสเลฟซีเรียล (slave Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิท (Bit) จากไมโครโปรเซสเซอร์ตามสัญญาณอินพุท CCLK ส่วนการต่อลักษณะมาสเตอร์

ซีเรียล (Master Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิทจากหน่วยความจำภายนอกที่เป็นแบบอนุกรม

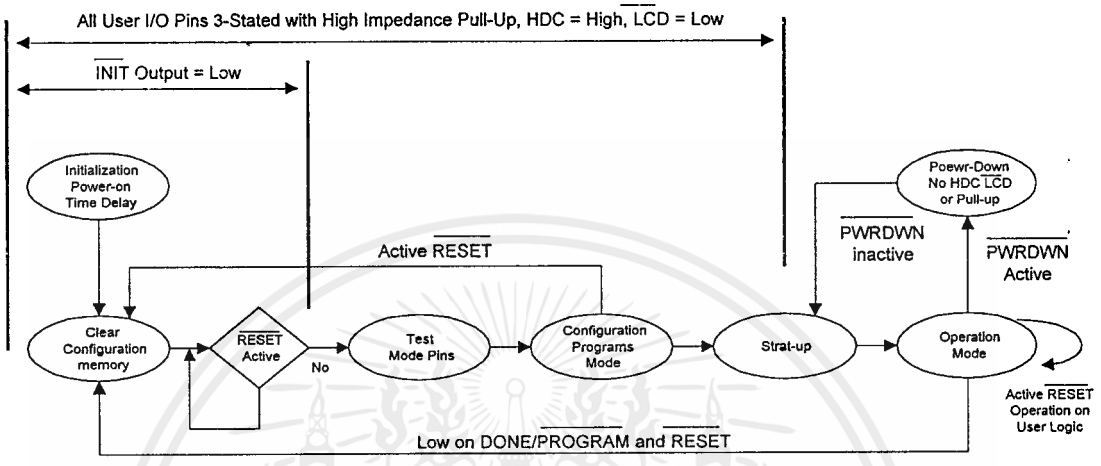
ตารางที่ 2.2 โหมดต่าง ๆ ของการคอนฟิกกูเรชัน

Mode	M2	M1	M0	CCLK	Data
Master Serial	0	0	0	output	Bit-Serial
Slave Serial	1	1	1	input	Bit-Serial
Master parallel up	1	0	0	output	Byte-Wide,00000 up
Master parallel down	1	1	0	output	Byte-Wide,3FFFF down
Peripheral Synchr	0	1	1	input	Byte-Wide
Peripheral Asynchr	1	0	1	output	Byte-Wide
Reserved	0	1	0	---	---
Reserved	0	0	1	---	---

จากความต้องการสร้างให้ใช้กระแสไฟฟ้าต่ำจากลักษณะของการต่อใช้งานทั้ง 5 แบบจึงมีเพียง 2 แบบเท่านั้นที่เหมาะสม คือ แบบมาสเตอร์ซีเรียลและแบบสเลฟซีเรียล ส่วนแบบมาสเตอร์พาราแรลต้องใช้ EPROM 27CXXX ซึ่งกินกระแสมากกว่า PROM XC17XXX เหมาะในการทดสอบต้นแบบก่อน เมื่อวงจรต้นแบบทำงานได้ถูกต้องแล้วจึงทำการอัดโปรแกรมลง PROM อีกทีหนึ่งเพราะว่าในแบบพาราแรลนั้น EPROM สามารถโปรแกรมได้ใหม่ต่างกับ PROM ที่โปรแกรมได้เพียงครั้งเดียว

การใช้งาน FPGAs ในการต่อลักษณะสเลฟซีเรียลและมาสเตอร์ซีเรียล เมื่อเริ่มจ่ายไฟเข้าตัว FPGAs จะทำการลบข้อมูลหน่วยความจำที่ใช้ในคอนฟิก (Configuration Memory) ตรวจสอบลักษณะการคอนฟิกว่าเป็นลักษณะใดในตารางที่ 2.3 ว่าเป็นแบบอนุกรมหรือแบบขนาน แล้วหลังจากนั้นก็จะเริ่มทำการโปรแกรมคอนฟิกสัญญาณ DONE/PROGRAM เป็น "0" ซึ่งอยู่ในระหว่างโปรแกรม และเมื่อข้อมูลในคอนฟิกที่รับมาจากภายนอกเต็มหน่วยความจำที่ใช้ในการคอนฟิก และความยาวของข้อมูลตรงกับที่

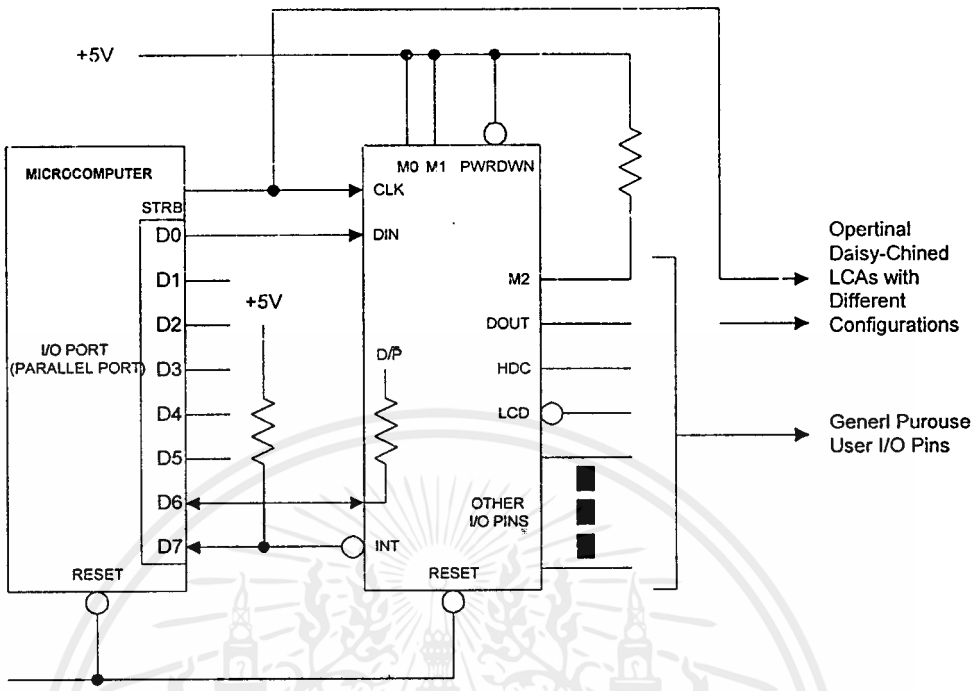
ส่วนหัวของข้อมูลคอนฟิก สัญญาณ DONE/PROGRAM จะเป็น “1” ซึ่งหมายถึง โปรแกรมทำการคอนฟิกเสร็จสิ้น ดูรูป 2.12 ประกอบ



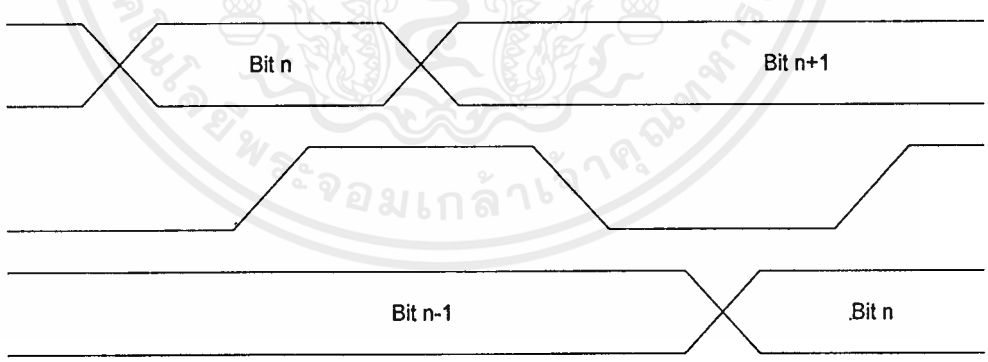
รูปที่ 2.12 ลำดับไดอะแกรมในการคอนฟิกเมื่อเริ่มป้อนแหล่งจ่ายไฟเข้าไอซี และการโปรแกรมใหม่

1. การใช้งานในลักษณะสเลฟซีเรียล

การต่อใช้งานในลักษณะนี้ เหมาะสมกับวงจรที่ถูกออกแบบมาเพื่อทำงานร่วมกับไมโครคอมพิวเตอร์อยู่แล้ว ทั้งนี้เพราะว่า FPGAs ได้ใช้ความสามารถของไมโครคอมพิวเตอร์ในการเก็บและส่งข้อมูลคอนฟิกให้ เพียงแต่ต้องเขียนโปรแกรมเพื่อส่งโปรแกรมคอนฟิกให้เพิ่มลักษณะของการต่อในลักษณะนี้เป็นดังรูปที่ 2.13 ซึ่งไมโครคอมพิวเตอร์จะสร้างสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs การป้อนโปรแกรมคอนฟิกให้ FPGAs ทำได้โดยการต่อสัญญาณ Strobe เข้ากับขา CCLK และพอร์ต Do เข้ากับขา DIN สร้างสัญญาณคล็อกป้อนที่ขา CCLK และป้อนโปรแกรมคอนฟิกแบบอนุกรมเข้าที่ขา DIN ดังแผนภูมิในรูปที่ 2.14



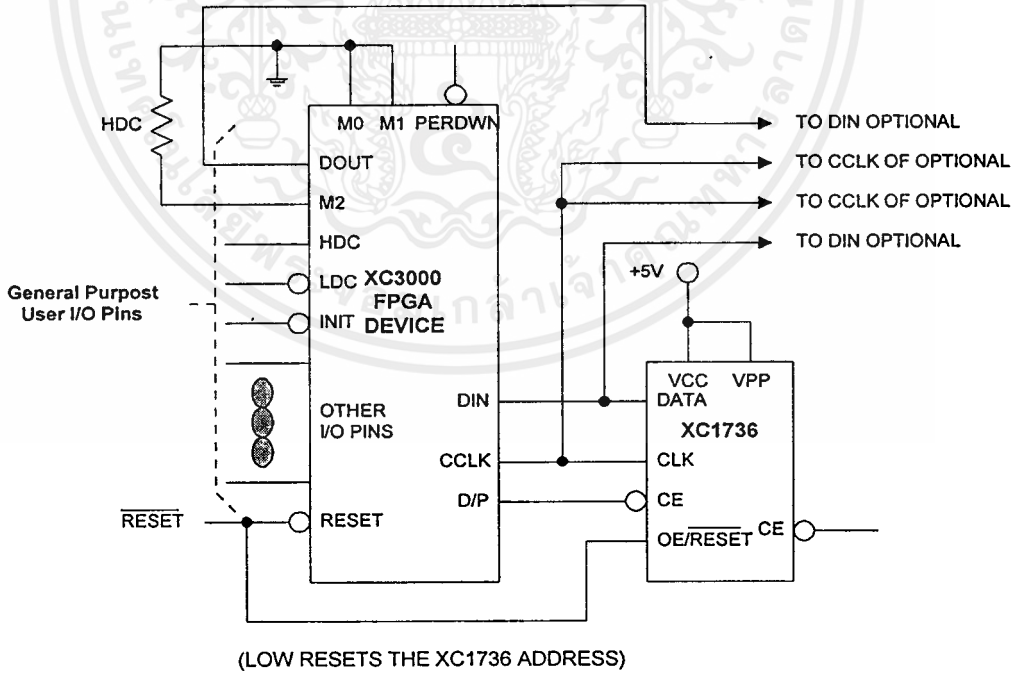
รูปที่ 2.13 การต่อใช้งานในลักษณะสเลฟซีเรียล



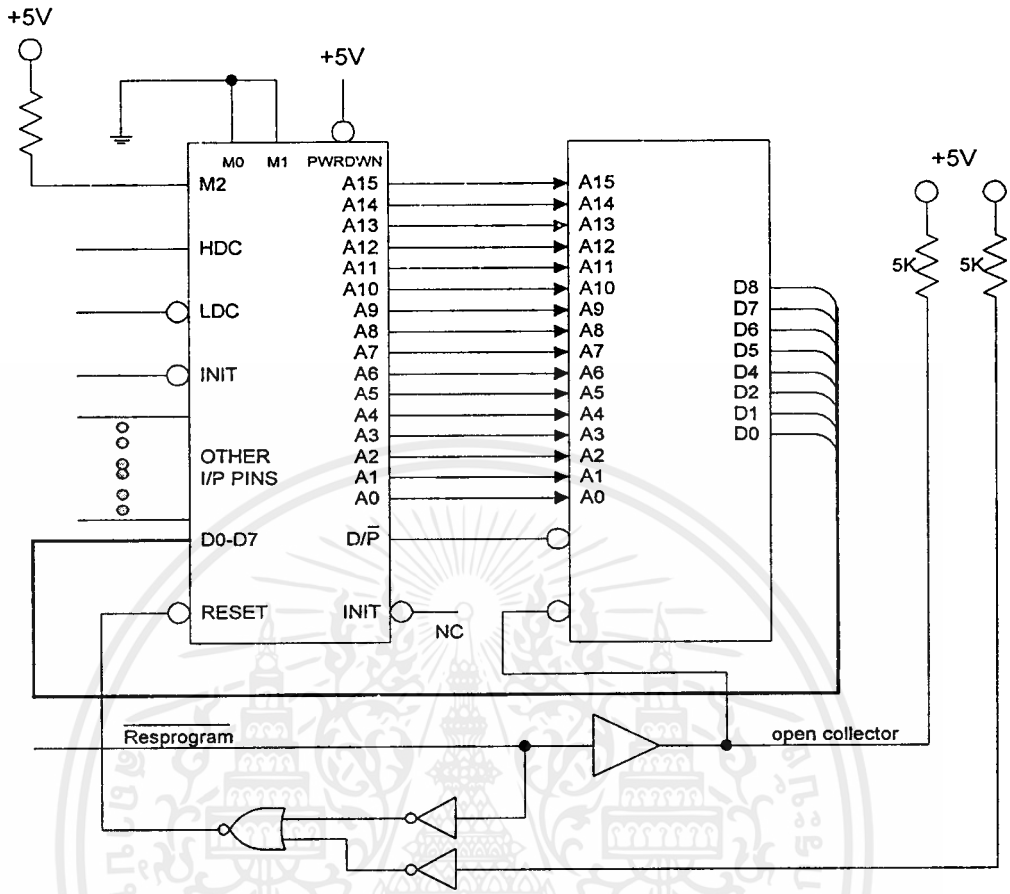
รูปที่ 2.14 แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในลักษณะสเลฟซีเรียล

2. การใช้งานในลักษณะมาสเตอร์ซีเรียล

การต่อใช้งานในลักษณะนี้ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากการต่อลักษณะแรกคือใช้ PROM เบอร์ XC17XXX เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง PROM จะทำตามขั้นตอนดังนี้คือ เมคบิท (MakeBits) สร้างไฟล์ .BIT จากวงจรที่ออกแบบ และใช้โปรแกรม MakePROM สร้าง Hex ไฟล์แล้วทำการอัดโปรแกรมลงใน PROM ด้วยอุปกรณ์อัด PROM ที่มาพร้อมกับตัวโปรแกรมของไซลิงค์ PROM XC17XXX จะส่งสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ ดังแสดงในรูปที่ 2.15 D0-D7 เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบขนาน A0-A15 เป็นแอดเดรสที่ FPGAs สร้างให้กับ EPROM เพื่ออ่านข้อมูลจากหน่วยความจำมาเก็บไว้ในสแตติกแรม (StaticRAM แอดเดรสทั้ง 16 เส้นไม่จำเป็นต้องต่อให้ครบก็ได้ ขึ้นอยู่กับขนาดหน่วยความจำ EPROM ที่ใช้และสามารถกำหนดให้นับขึ้นหรือลงได้



รูปที่ 2.15 การต่อใช้งานในลักษณะมาสเตอร์ซีเรียล



รูปที่ 2.16 การต่อใช้งานในลักษณะมาสเตอร์พาราแรล

2.3.4 ข้อควรระวังในการใช้อุปกรณ์ FPGAs

สิ่งที่สำคัญ คืออุปกรณ์ FPGAs ไวต่อความร้อนมาก การบัดกรีโดยหัวแร้งกำลังสูงหรือบัดกรีโดยจี้หัวแร้งที่ขาไอซีเป็นเวลานานจะทำให้ไอซีเสียหายได้ ระยะเวลาในการบัดกรีหนึ่งจุดไม่ควร เกิน 5-10 วินาที ควรใช้ซ็อกเก็ต (Socket) ไอซีในการประกอบวงจรลงแผ่นวงจรพิมพ์

การป้องกันไอซีจากแรงดันไอซีจากแรงดันไฟฟ้าไม่ควรต่อสลับขั้วบวกกับขั้วลบจะทำให้ไอซีเสียได้นอกจากนั้นแรงดันของแหล่งไฟต้องอยู่ในช่วงที่โรงงานกำหนดมา สำหรับ FPGAs ค่าแรงดันที่ใช้งานอยู่ในช่วง Vcc = 4.75 - 5.25 V และแรงดันที่ทนได้อยู่ในช่วง -0.5 - 7 V ดังนั้นก่อนป้อนแรงดันควรตรวจเช็คให้แน่ใจก่อน

2.4 ทฤษฎีของโทรทัศน์เบื้องต้น

2.4.1 ส่วนประกอบของภาพ

หากเราพิจารณาดูภาพหรือรูปที่ปรากฏในหน้าหนังสือพิมพ์ หรือวารสารต่าง ๆ แล้ว จะเห็นว่าภาพเหล่านี้ประกอบขึ้นด้วยจุดดำเล็ก ๆ เป็นจำนวนมากซึ่งมีทั้งส่วนที่ดำสนิทและส่วนที่ดำจาง

ขนาดของจุดดำในส่วนของภาพที่มืดสนิท เราจะมองเห็นใหญ่กว่าขนาดของจุดดำในส่วนของภาพที่จาง จำนวนจุดดำที่มีมากหรือน้อยนี้มีผลทำให้ภาพมองดูละเอียดหรือหยาบแตกต่างกัน ระยะทางที่มองดูภาพมีส่วนสำคัญอยู่ไม่น้อย ภาพที่หยาบ ถ้าหากเรามองดูในระยะทางซึ่งไกลกว่าระยะที่ใช้มองดูภาพละเอียดทำให้รู้สึกว่าพอดูได้เหมือนกัน

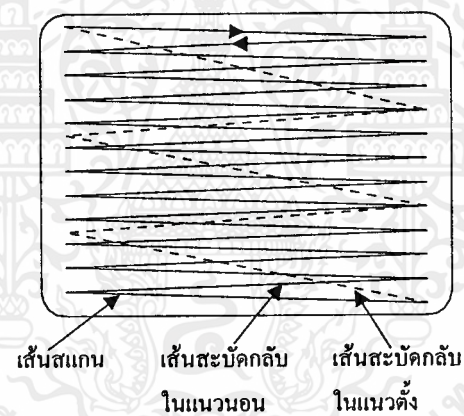
ในทำนองเดียวกัน ภาพที่ปรากฏบนจอหลอดภาพของเครื่องรับโทรทัศน์นั้นประกอบไปด้วยเส้นขวางเล็ก ๆ ในแนวนอนเป็นจำนวนมาก ซึ่งแต่ละเส้นนี้มีทั้งส่วนที่ดำสนิท ส่วนที่ดำจาง และส่วนที่สว่างมากรวมกันอยู่เส้นขวางเล็ก ๆ ตามแนวนอนเหล่านี้เรียกว่า เส้นสแกน (scan) ซึ่งประกอบไปด้วยส่วนหรือจุดเล็กๆที่มีทั้งมืดและสว่างปะปนกัน

ภาพที่ปรากฏบนจอหลอดภาพจึงประกอบขึ้นจากจุดเล็ก ๆ ที่มีระดับของความสว่างที่แตกต่างกันเป็นจำนวนมากมายจุดเล็ก ๆ เหล่านี้เรียกว่าส่วนประกอบของภาพ (picture elements) ซึ่งมีความสำพันธ์กับความละเอียดของภาพมากหากจำนวนจุดเล็ก ๆ หรือจำนวนของเส้นสแกนในแนวนอนมีมากยิ่งขึ้นเพียงไร ภาพที่เห็นบนจอหลอดภาพจะละเอียดมากขึ้นเท่านั้น ดังนั้น โทรทัศน์ในระบบยุโรปที่มีจำนวนเส้นสแกน 625 เส้น จึงให้ภาพที่ละเอียดกว่าโทรทัศน์ในระบบอเมริกา ที่มีจำนวนเส้นสแกนเพียง 525 เส้นเท่านั้น แต่อย่างไรก็ตามภาพที่ปรากฏเห็นบนจอหลอดภาพจะมองดูละเอียด,หยาบ หรือน่าดูอย่างไรนั้นยังขึ้นอยู่กับส่วนประกอบอีกหลายอย่างเช่น ความสว่างของภาพ และระยะทางที่มองดูภาพเป็นต้น สำหรับโทรทัศน์ระบบอเมริกาแม้จะมีจำนวนเส้นสแกนน้อยกว่าจำนวนเส้นของโทรทัศน์ระบบยุโรป อาจทำให้เห็นภาพหยาบไปบ้างก็ตาม แต่ถ้าหากมองในระยะทางห่างประมาณ 4 ถึง 8 เท่าของความสูงภาพแล้ว จะเห็นว่าเป็น

ภาพที่พอใช้ได้เหมือนกัน นอกจากนี้ มาตรฐานเพื่อการมองของสายตาของคน จะกำหนดให้ภาพมีขนาดอัตราส่วนความกว้างต่อความสูงของภาพเป็น 4 : 3 อีกด้วย

2.4.2 วิธีการสแกน และการหักเหของลำอิเล็กตรอน

ภายในหลอดภาพของเครื่องรับโทรทัศน์ ส่วนของอิเล็กตรอนที่หลุดออกมาจากขั้วแคโถด (cathode) และถูกดึงดูดให้วิ่งเป็นลำไปกระทบกับขั้วแอโนด (anode) หรือจอหลอดภาพที่ฉาบวัสดุเรืองแสงบางชนิดเอาไว้ ทำให้มองเห็นเป็นจุดสว่างขึ้นที่จอหลอดภาพ คือ ทำให้จุดสว่างเคลื่อนที่ในจังหวะที่ถูกต้อง ทั้งในแนวนอน และแนวตั้งของจอหลอดภาพ โดยอาศัยความเข้มของสนามแม่เหล็กไฟฟ้า (electromagnetic) เข้าช่วยเหลือ ดังแสดงในรูปที่ 2.17



รูปที่ 2.17 การเคลื่อนที่หักเหของลำอิเล็กตรอนในจังหวะที่ถูกต้อง

ในขณะที่ไม่มีสนามแม่เหล็กไฟฟ้า ลำอิเล็กตรอนจะวิ่งไปกระทบจอหลอดภาพตรงกลางโดยไม่ถูกหักเหเลย ถ้าหากต้องการเบนลำอิเล็กตรอนไปทางซ้ายมือในแนวนอน ก็จำเป็นต้องใช้สนามแม่เหล็กไฟฟ้าที่มีขั้วเหนือ-ขั้วใต้อยู่ในแนวตั้ง หากกลับขั้วแม่เหล็กลำของอิเล็กตรอนก็จะถูกเบนไปทางขวามือในแนวนอนของจอหลอดภาพ การที่ลำอิเล็กตรอนถูกเบี่ยงเบนไปทางขวามือหรือทางซ้ายมือของจอนี้จะทำให้เห็นเป็นจุดสว่างที่เคลื่อนที่ไปทางเดียวกันด้วย ในทำนองเดียวกัน หากมีขั้วแม่เหล็กในแนวนอน ลำอิเล็กตรอนหรือจุดสว่างก็จะถูกเบนไปในทางแนวตั้งของจอหลอดภาพ เพื่อช่วยใน

การหักเหลำอิเล็กตรอนในทิศทางที่ต้องการ จึงใช้สนามแม่เหล็กไฟฟ้าในแนวนอนและในแนวตั้งร่วมกัน สนามแม่เหล็กไฟฟ้านี้เกิดขึ้นจากการปล่อยกระแสไฟฟ้าผ่านขดลวดที่พันอยู่รอบ ๆ ของคอหลอดภาพขดลวดนี้เรียกว่าขดลวดของการหักเหทางแนวนอนและขดลวดของการหักเหทางแนวตั้งตามลำดับ ส่วนรูปร่างของกระแสไฟฟ้าที่ไหลผ่านขดลวดทั้งสอง เพื่อที่จะทำให้เกิดการสแกนตามรูปที่ 2.17 นั้น มีความสำคัญมาก และนิยมใช้เป็นกระแสรูปฟันเลื่อย และความถี่ของกระแสที่ไหลผ่านขดลวดทั้งสองนี้จะไม่เท่ากัน สำหรับโทรทัศน์ระบบอเมริกา กระแสรูปฟันเลื่อยที่ไหลผ่านขดลวดของการหักเหทางแนวนอนจะมีความถี่ 15,750 Hz ส่วนกระแสรูปฟันเลื่อยที่ไหลผ่านขดลวดของการหักเหทางแนวตั้งจะมีความถี่เพียง 60 Hz เท่านั้น

โดยปกติการสแกนจะเริ่มต้นขึ้น โดยการทำให้จุดสว่างเคลื่อนที่จากทางซ้ายมือด้านบนของจอไปทางขวามือในแนวนอน ซึ่งเมื่อไปถึงตำแหน่งขวาสุด ก็จะถูกเบนต่ำลงเล็กน้อย แล้วจะกลับไปตั้งต้นใหม่ทางซ้ายมือเพื่อเคลื่อนที่มาทางขวามือในแนวนอนอีกและจะเป็นอยู่เช่นนี้เรื่อย ๆ จนกระทั่งจุดสว่างไปถึงตำแหน่งขวามือข้างล่างสุดของหลอดภาพ เป็นอันเสร็จสิ้นการสแกนภาพนิ่ง 1 ภาพ เรียกว่า 1 เฟรม (frame) หลังจากนั้น ลำอิเล็กตรอนจะกลับไปตั้งต้นใหม่ทางด้านซ้ายมือบนสุดของจอหลอดภาพอีก เพื่อการสแกนภาพนิ่งอันดับต่อไป อย่างไรก็ตาม ในการลดอาการกระพริบของภาพ การสแกนภาพนิ่งแต่ละภาพมักจะนิยมจัดทำสองครั้งในแบบของการสแกนไขว้กัน โดยกำหนดให้ภาพนิ่ง 1 เฟรม ประกอบด้วยภาพนิ่ง 2 ฟิลด์ (field) และเริ่มต้นด้วยการสแกนภาพนิ่งฟิลด์เส้นคี่ก่อน เมื่อเสร็จสิ้นถึงตำแหน่งขวามือล่างสุดของจอหลอดภาพ จึงกลับไปตั้งต้นใหม่ทางซ้ายมือบนสุดของจอหลอดภาพ เพื่อเริ่มต้นสแกนภาพนิ่งฟิลด์เส้นคู่ต่อไป จนถึงตำแหน่งขวามือล่างสุด หลังจากนั้นจะเริ่มต้นสแกนภาพนิ่งอันดับอื่นต่อไปใหม่

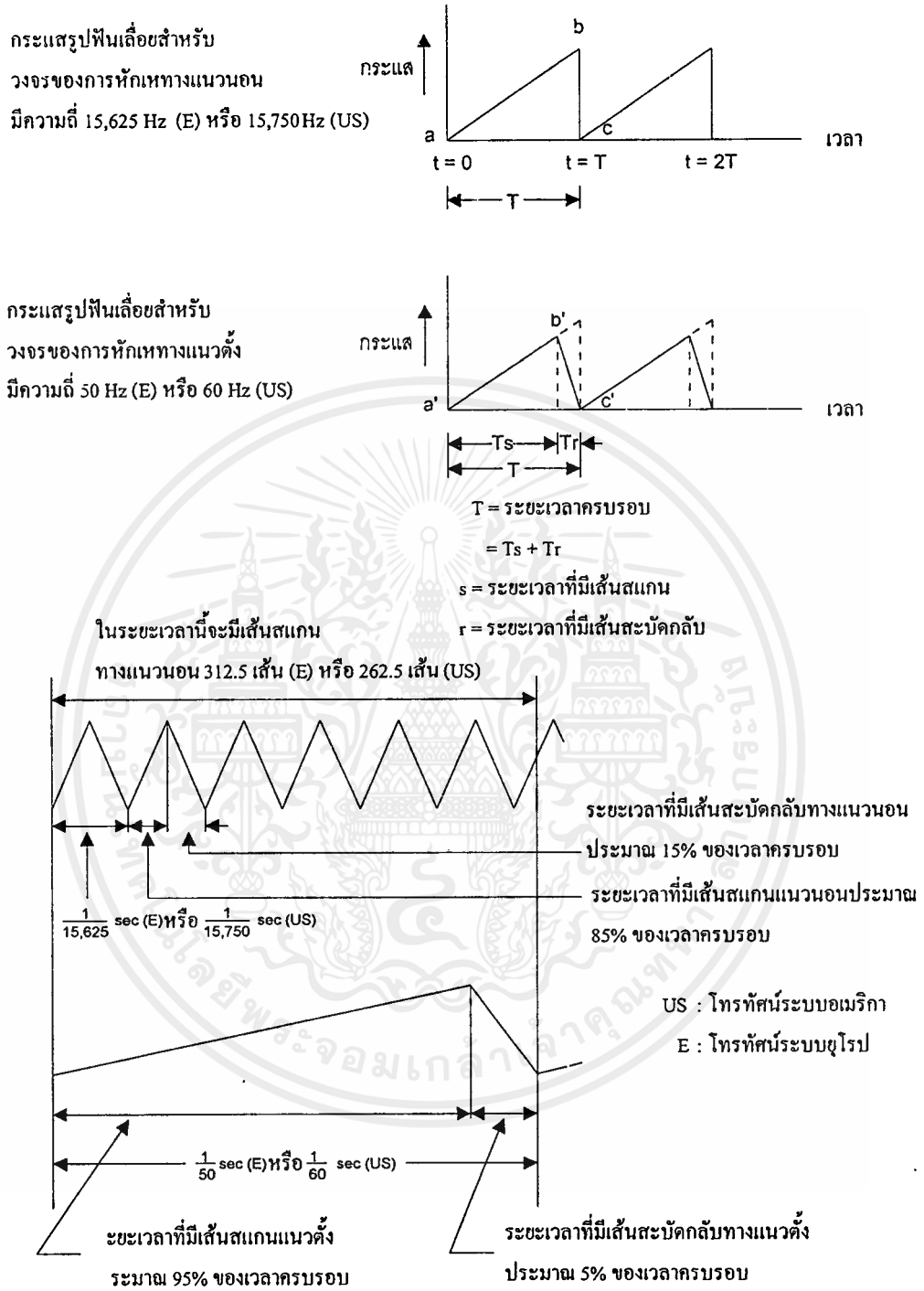
ดังนั้น การสแกนภาพนิ่งหนึ่งภาพหรือหนึ่งเฟรมจึงประกอบด้วยการสแกนภาพนิ่งด้วยฟิลด์เส้นคี่ และการสแกนภาพนิ่งด้วยฟิลด์เส้นคู่สำหรับโทรทัศน์ระบบยุโรปใช้เส้นสแกนแนวนอน 625 เส้นต่อภาพ และ 25 ภาพต่อวินาที ความถี่ของกระแสไฟฟ้าที่ทำให้เกิดการหักเหทางแนวนอน และการหักเหทางแนวตั้งจึงมีค่าเป็น (625) × (25) หรือ 15,625 Hz และ 50 Hz ตามลำดับ ความถี่ของกระแสสำหรับการหักเหทางแนว

นอน และการหักเหทางแนวตั้งทั้งสอง แสดงในรูปที่ 2.18 ในระยะเวลาครบรอบหนึ่ง ๆ ของกระแสรูปฟันเลื่อยประกอบด้วยส่วนที่เพิ่มขึ้นจากค่าต่ำสุดไปหาค่าสูงสุด ซึ่งตรงกับเวลาที่จุดสว่างใช้ไปในการสแกนจากซ้ายมือสุดไปจนถึงขวามือสุด และส่วนที่ลดลงจากค่าสูงสุดไปหาค่าต่ำสุดตรงกับระยะเวลาที่จะสว่างบนจอหลอดภาพใช้ในการสะบัดกลับ (fly-back) จากขวามือสุดไปตั้งต้นใหม่ทางซ้ายมือสุด โดยปกติระยะเวลาที่มีเส้นสะบัดกลับ จะเป็นช่วงเวลาที่น้อยมากเมื่อเทียบกับช่วงเวลาที่มีเส้นสแกน จุดสว่างที่มองเห็นสะบัดกลับไปในช่วงเวลาดังกล่าวแล้วนี้ ไม่ก่อให้เกิดประโยชน์อันใดเลย จึงหาวิธีทำให้เกิดสิ่งอื่นมาข่มจุดสว่างในช่วงเวลานี้ เพื่อมิให้สังเกตเห็นได้ทางจอหลอดภาพ สัญญาณที่ใช้ลบเส้นสะบัดกลับนี้เรียกว่า สัญญาณแบล้งค์กิ้ง (blanking signal)

เนื่องจากการสแกนภาพนิ่งตามที่กล่าวถึงแล้วนี้ กระทำติดต่อกันไปเรื่อย ๆ โดยมีจำนวนเส้นต่อภาพและจำนวนภาพต่อวินาที ตามแต่ละระบบโทรทัศน์ที่ใช้ ภาพที่มาปรากฏบนจอหลอดภาพเครื่องรับโทรทัศน์ จึงมีผลเหมือนกับการฉายภาพนิ่ง ซึ่งแต่ละภาพแตกต่างกันบ้างเล็กน้อยเป็นจำนวนหลาย ๆ ภาพต่อหนึ่งวินาที และด้วยคุณลักษณะพิเศษของสายตาจะทำให้มองเห็นภาพบนจอหลอดภาพเครื่องรับโทรทัศน์เป็นภาพที่เคลื่อนไหวติดต่อกันไปตลอดเวลา

2.4.3 เครื่องส่งและเครื่องรับโทรทัศน์

เครื่องส่งและเครื่องรับโทรทัศน์จะต้องมีการสแกนทั้งทางแนวนอน และการสแกนทางแนวตั้งพร้อมกัน จึงจะมีภาพเกิดขึ้นที่เครื่องรับโทรทัศน์ ภาพที่ปรากฏขึ้นบนจอหลอดภาพจำเป็นจะต้องอาศัยวงจรการหักเหทางแนวนอน (horizontal deflection circuit) และวงจรการหักเหทางแนวตั้ง (vertical deflection circuit) ซึ่งแต่ละวงจรจะมีกระแสรูปฟันเลื่อยไหลผ่าน ส่วนทางด้านของกล้องโทรทัศน์จำเป็นต้องอาศัยความถี่ทั้งสองช่วยทำให้เกิดสัญญาณทางไฟฟ้าเช่นเดียวกัน



รูปที่ 2.18 กระแสรูปฟันเลื่อย สำหรับใช้ในวงจรที่ทำให้เกิดการหักเหของอิเล็กตรอน ในแนวนอน และในแนวตั้ง

ดังนั้นความถี่ทางวงจรของการหักเหทางแนวนอนและวงจรของการหักเหทางแนวตั้งที่ใช้ในเครื่องส่งโทรทัศน์ และที่ใช้ในเครื่องรับโทรทัศน์นี้จะต้องเท่ากันตลอดเวลา จึงจะทำให้เกิดภาพขึ้นที่เครื่องรับโทรทัศน์ ด้วยเหตุนี้ ต้องมีวิธีทำให้ความถี่ของวงจรดังกล่าวทางเครื่องส่ง และทางเครื่องรับโทรทัศน์เท่ากันตลอดเวลา ดังแสดงตามรูปที่ 2.19 โดยที่สถานีโทรทัศน์จะต้องส่งสัญญาณชนิดหนึ่งที่เรียกว่า สัญญาณซิ้งค์ (synchronizing) ไปพร้อม ๆ กับสัญญาณภาพและสัญญาณเสียง สัญญาณซิงค์จะช่วยทำให้ความถี่ในวงจรของการหักเหทางแนวนอนและวงจรของการหักเหทางแนวตั้งในเครื่องส่ง และเครื่องรับโทรทัศน์เท่ากันเพื่อทำให้เกิดภาพที่จอหลอดภาพของเครื่องรับโทรทัศน์ตลอดเวลาได้

2.4.4 สัญญาณต่าง ๆ ที่ส่ง

เพื่อให้เกิดผลตามความมุ่งหมาย สถานีโทรทัศน์ที่ส่งภาพขาวดำจำเป็นจะต้องส่งสัญญาณหลายอย่าง คือ

-สัญญาณเสียง

-สัญญาณภาพ

-สัญญาณแบล้งค์กิ้ง

-สัญญาณซิ้งค์

-สัญญาณอีควอไลซิ่ง

สัญญาณเสียงมีคลื่นพาห์ (carrier wave) เป็นของตัวเองโดยเฉพาะ ส่วนสัญญาณภาพ และสัญญาณอื่น ๆ นั้นจะรวมกันเป็นรูปร่างอันเดียวกัน เรียกว่าสัญญาณภาพรวม (composite video signal) แล้วใช้คลื่นพาห์เป็นตัวพาออกอากาศรวมกับคลื่นพาห์ของเสียงส่งไปยังเครื่องรับโทรทัศน์ เหตุผลและความจำเป็นในการใช้สัญญาณต่าง ๆ มีดังนี้

ก) สัญญาณภาพ และสัญญาณเสียง เป็นสัญญาณที่ใช้เพื่อทำให้เกิดภาพ และเสียงทางเครื่องรับโทรทัศน์ตามความต้องการ

ข) สัญญาณแบล้งค์กิ้ง เป็นสัญญาณที่ใช้เพื่อลบเส้นสแกนสะบัดกลับทั้งในแนวนอนและในแนวตั้ง เพื่อมิให้สังเกตเห็นได้ชัดทางจอหลอดภาพ สำหรับโทรทัศน์ระบบอเมริกาวงจรของการหักเหทางแนวนอนมีความถี่ 15,750 H_z ดังนั้นในระยะเวลา 1/15,750 S หรือ 63.5 S จะต้องเกิดเส้นสแกนสะบัดกลับครั้งหนึ่ง จึงต้องใช้แบล้งค์กิ้ง

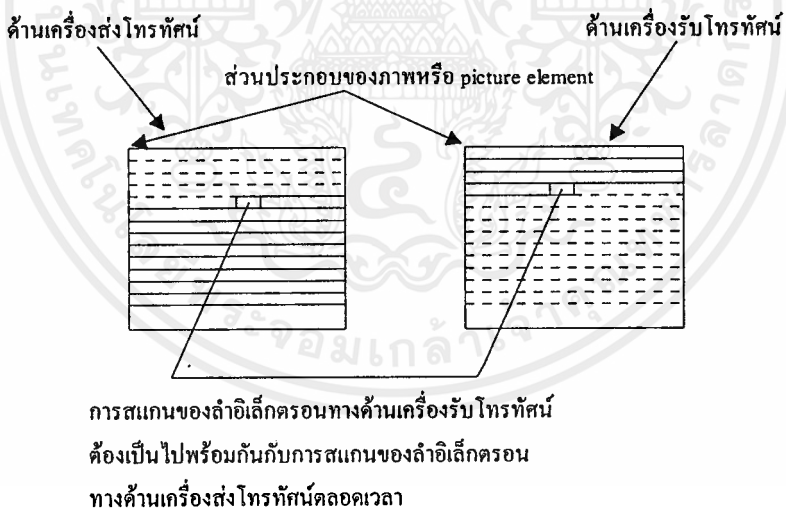
พัลซ์ทางแนวนอน (horizontal fly-back line) หนึ่งครั้ง โดยมีขนาดประมาณ 10 S ในทำนองเดียวกันทุก ๆ ช่วงเวลา 1/60 S หรือ 16.667 S ก็ต้องใช้แบล้งค์กิ้งพัลซ์ทางแนวตั้ง (vertical fly-back line) ครั้งหนึ่งโดยมีขนาดประมาณ 1,250 S

ค) สัญญาณซิ้งค์ เป็นสัญญาณที่ใช้เพื่อช่วยทำให้วงจรการหักเหทางแนวนอนและวงจรการหักเหทางแนวตั้งในเครื่องส่งกับเครื่องรับโทรทัศน์มีความถี่ตรงกันตลอดเวลาสัญญาณซิงค์ทางแนวนอนมีความถี่ 15,750 H_z เท่ากับความถี่ของวงจรการหักเหทางแนวนอนและสัญญาณซิ้งค์ทางแนวตั้งมีความถี่ประมาณ 60 H_z ซึ่งเท่ากับความถี่ของวงจรหักเหทางแนวตั้งเช่นกัน เนื่องจากความถี่ของสัญญาณซิ้งค์มีค่าเท่ากับความถี่ของสัญญาณแบล้งค์กิ้งพอดี จึงจำเป็นต้องป้องกันการรบกวนที่อาจเกิดขึ้น โดยจำเป็นจะต้องกำหนดขนาดของซิ้งค์พัลซ์ให้น้อยกว่าขนาดของแบล้งค์กิ้งพัลซ์ คือ ทำให้ซิ้งค์พัลซ์ทางด้านแนวนอนมีขนาดเพียง 5 S และซิ้งค์พัลซ์ทางแนวตั้งมีขนาดเพียง 190 S เท่านั้น นอกจากนี้ ยังใช้วิธีส่งซิ้งค์พัลซ์เหล่านี้ปนไปกับแบล้งค์กิ้งพัลซ์อีกด้วย โดยให้ฐานของซิ้งค์พัลซ์อยู่ทับขอบบนของแบล้งค์กิ้งพัลซ์อีกชั้นหนึ่ง ดังนั้น เมื่อจัดขอบเขตความต่างศักย์ให้ระดับสูงสุดของแบล้งค์กิ้งพัลซ์เป็นระดับดำมืดจนมองไม่เห็นแล้ว ระดับของซิ้งค์พัลซ์ที่อยู่บนยอดสูงสุดของแบล้งค์กิ้งพัลซ์ก็จะเป็นระดับดำมืดสนิท และจะไม่ทำให้เกิดการรบกวนภาพที่จอหลอดภาพแต่อย่างใด

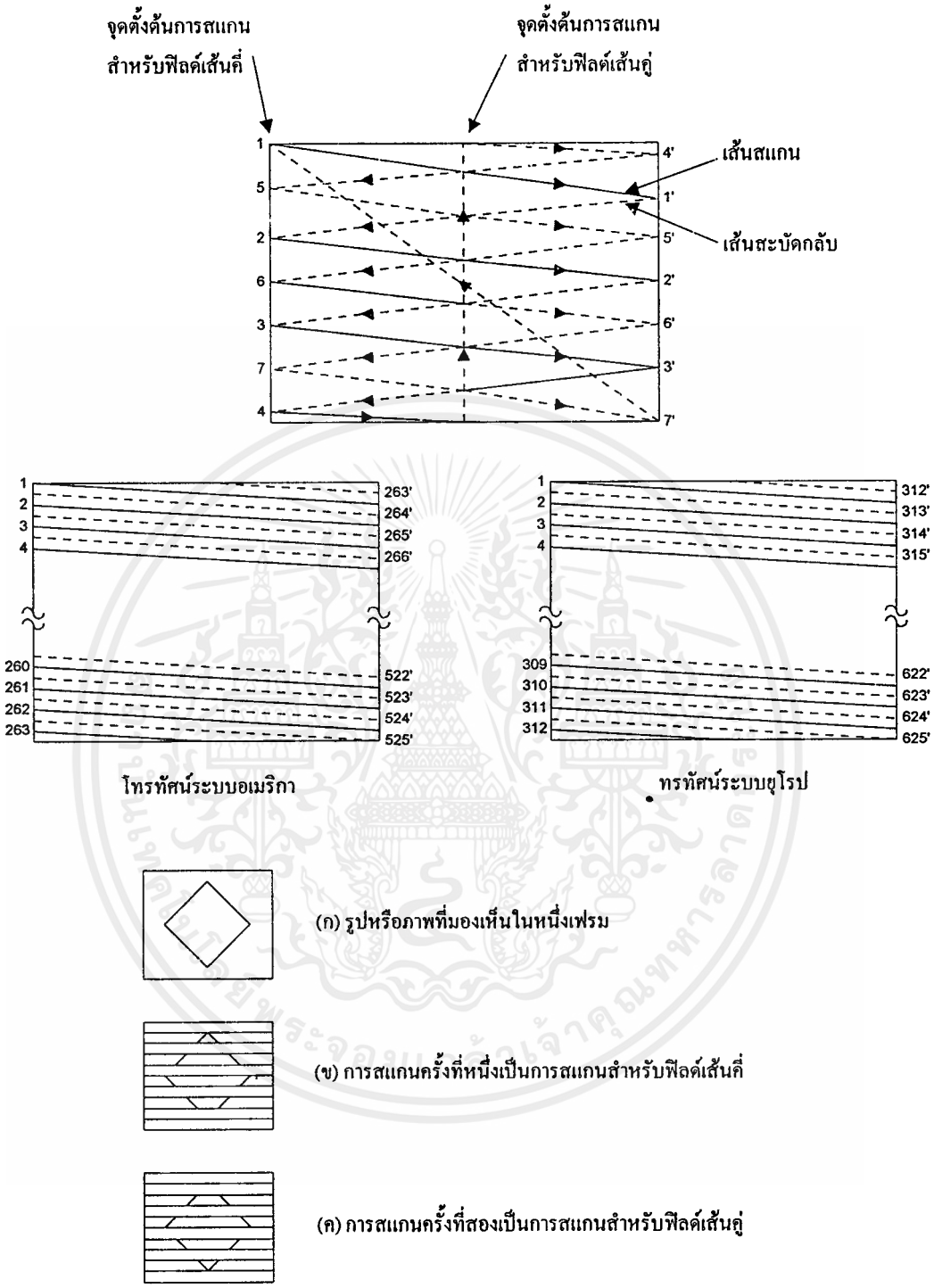
ง) สัญญาณอีควอไลซิ่ง เป็นสัญญาณที่ใช้เพื่อช่วยให้สัญญาณซิ้งค์ทางแนวตั้งยังคงมีรูปร่างดีเหมือนเดิม หลังจากแยกออกมาจากสัญญาณซิงค์ทางแนวนอนแล้ว นอกจากนี้ ยังช่วยทำให้การสแกนแบบไขว้กันเป็นไปด้วยความเรียบร้อยสม่ำเสมอ รวมทั้งสัญญาณซิ้งค์ทางแนวนอนก็ไม่ขาดหายไปในช่วงเวลาของสัญญาณซิ้งค์ทางแนวตั้งอีกด้วย ขนาดของพัลซ์ที่กล่าวถึงนี้ จะเท่ากับสัญญาณซิ้งค์ทางแนวตั้งหรือ 190 S หรือประมาณสามเท่าของขนาดสัญญาณซิ้งค์ในทางแนวนอน และยังนิยมแบ่งพัลซ์นี้ออกเป็น 6 พัลซ์เล็ก ๆ ด้วยกัน ดังรูปที่ 2.22 เพื่อทำให้เกิดสัญญาณซิ้งค์ทางแนวนอนครั้งหนึ่งในทุก ๆ สองครั้งที่มีพัลซ์เล็ก ๆ นอกจากนี้ ยังนิยมแบ่งสัญญาณซิ้งค์ทางแนวตั้งออกเป็นพัลซ์เล็ก ๆ เช่นเดียวกัน

รูปที่ 2.21 แสดงภาพขาวสลับดำ ซึ่งเริ่มจากสีขาว,สีดำจาง ๆ และสีดำสนิทเป็นแถบ ๆ กล้องโทรทัศน์จะเปลี่ยนภาพให้เป็นสัญญาณทางไฟฟ้าชนิดหนึ่ง เมื่อรวมกับ

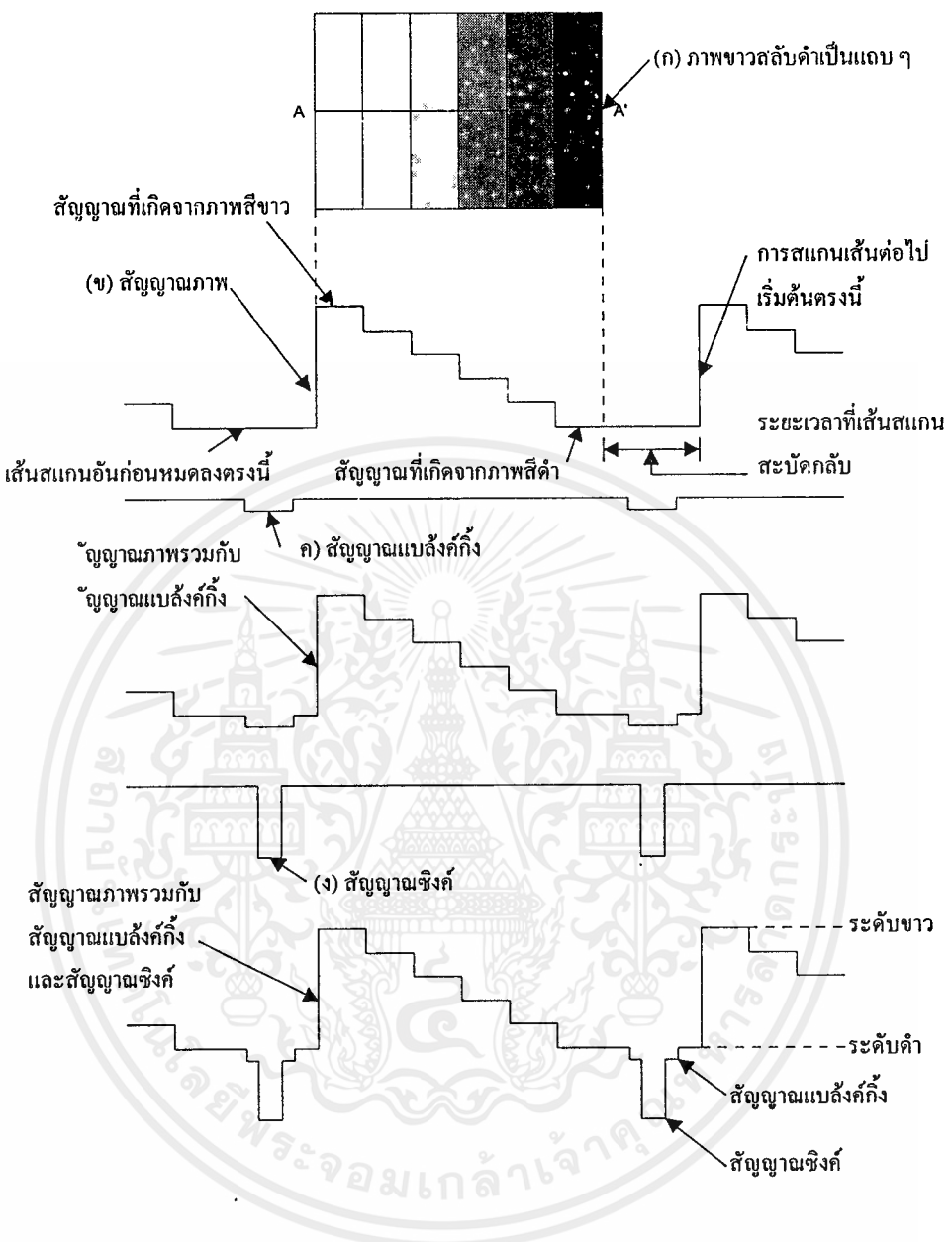
สัญญาณแบล้งค์กิ้ง และสัญญาณซิ้งค์แล้ว ก็จะได้สัญญาณภาพรวมตามที่แสดงไว้ในรูปที่ 2.22 ภาพแต่ละชนิดจะให้สัญญาณทางไฟฟ้าที่มีความถี่สูงต่ำแตกต่างกัน สำหรับโทรทัศน์ระบบอเมริกาความถี่สูงสุดของภาพไม่ควรเกิน 4 MH_z ส่วนโทรทัศน์ระบบยุโรป ความถี่สูงสุดของภาพนี้ไม่ควรเกิน 5 MH_z ในเรื่องนี้ ภาพที่เกิดจากสัญญาณโทรทัศน์ที่มีความถี่สูงย่อมละเอียดกว่า หรือมีจำนวนจุดดำอันเป็นส่วนประกอบของภาพมากกว่าภาพที่เกิดจากสัญญาณโทรทัศน์ที่มีความถี่ต่ำเมื่อเครื่องรับโทรทัศน์รับเอาสัญญาณโทรทัศน์มาแล้ว จะมีการแยกเอาสัญญาณต่าง ๆ ตามที่กล่าวนี้ไป ให้วงจรซึ่งทำหน้าที่ต่าง ๆ กัน เพื่อทำให้เกิดภาพและเสียงตามความต้องการ สัญญาณเสียงก็จะผ่านไปยังวงจรเสียง, สัญญาณภาพ และสัญญาณแบล้งค์กิ้งก็จะตรงไปยังขั้วแคโถดหรือกริด (grid) ของหลอดภาพ ส่วนสัญญาณซิ้งค์นั้นเมื่อแยกออกจากสัญญาณภาพรวมแล้วก็จะผ่านไปยังวงจรแยกซิ้งค์,วงจรของการหักเหทางแนวนอน และวงจรของการหักเหทางแนวตั้ง



รูปที่ 2.19 ความถี่ของกระแสรูปฟันเลื่อยในวงจรของการหักเหทางแนวนอนและวงจรของการหักเหทางแนวตั้งทางด้านเครื่องส่งและเครื่องรับโทรทัศน์

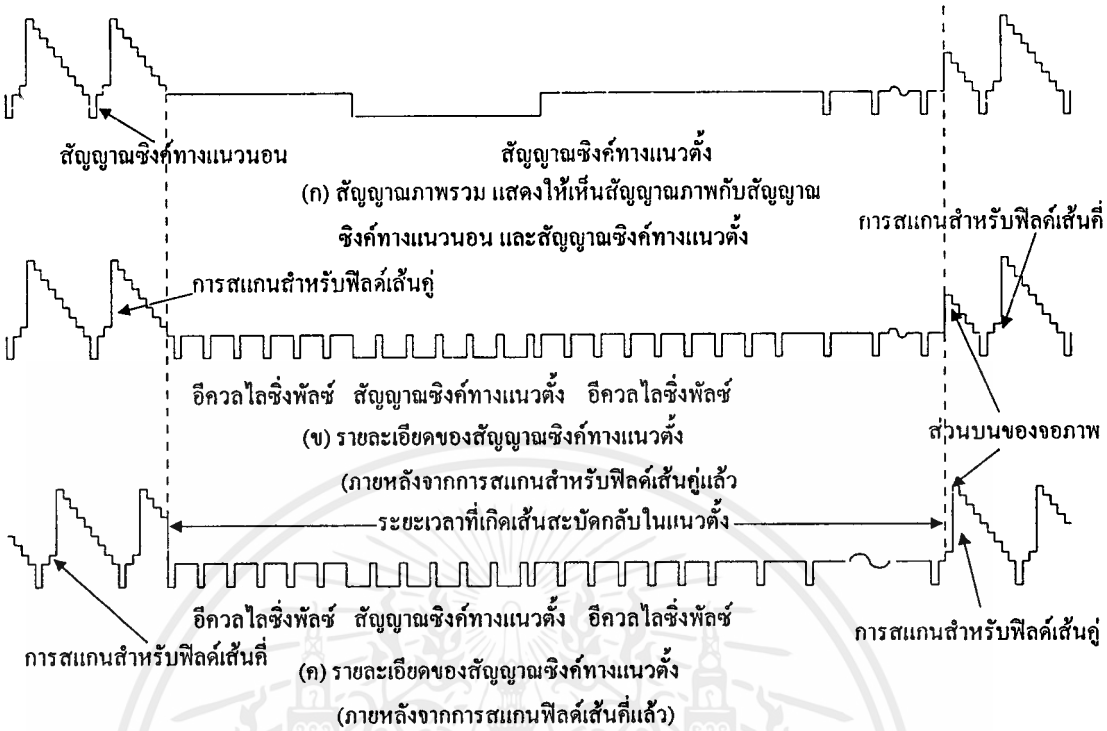


รูปที่ 2.20 การสแกนสองครั้งสำหรับภาพนิ่งแต่ละภาพ
โดยแบ่งหนึ่งเฟรมออกเป็นสองฟิลด์



รูปที่ 2.21 รูปร่างของสัญญาณโทรทัศน์ที่เกิดจากภาพขาวสลับดำเป็นแถบ ๆ

สัญญาณโทรทัศน์ที่มีสัญญาณภาพรวมกับสัญญาณอื่น ๆ หลายอย่าง ตามที่แสดงไว้ในรูปที่ 2.22 นี้ มีชื่อเรียกว่า สัญญาณภาพรวม



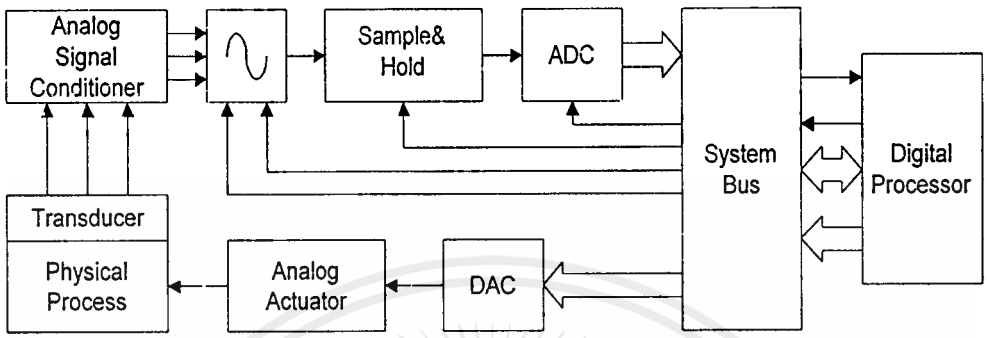
รูปที่ 2.22 สัญญาณภาพรวม

2.5 ทฤษฎีการแปลงสัญญาณ (Data Acquistion and Conversion)

ในอดีตที่ผ่านมารูปแบบของสัญญาณไฟฟ้าโดยมากมักจะอยู่ในรูปของสัญญาณแอนาลอกการนำเอาสัญญาณไฟฟ้ามาประมวลผล เพื่อให้เกิดรูปแบบที่ต้องการนั้น ต้องใช้อุปกรณ์ทางแอนาลอก แต่ปัจจุบันนี้เทคโนโลยีทางด้านดิจิตอลก้าวหน้าไปมากทำให้การประมวลผลสัญญาณทางดิจิตอล สามารถทำได้อย่างรวดเร็ว และมีประสิทธิภาพ

ดังนั้นการแปลงรูปแบบสัญญาณ (Conversation) จึงมีความจำเป็นในการแปลงสัญญาณแอนาลอกที่มีอยู่แล้วนั้นให้เป็นสัญญาณดิจิตอลโดยอุปกรณ์การแปลงสัญญาณแอนาลอกให้เป็นสัญญาณดิจิตอล และจะถูกประมวลผลโดยตัวประมวลผลสัญญาณดิจิตอล เช่น คอมพิวเตอร์ เป็นต้น จากผลลัพธ์ที่ได้อาจถูกนำมาแสดงผลได้โดยตรงเลย หรืออาจถูกแปลงให้มาอยู่ในรูปของสัญญาณแอนาลอกที่สามารถใช้งานได้ การที่จะแปลงสัญญาณดิจิตอลกลับไปเป็นสัญญาณแอนาลอกนั้น สามารถทำได้โดยใช้อุปกรณ์

แปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก สำหรับระบบที่มีการประมวลผลข้อมูลทางดิจิตอลแสดงดังรูปที่ 2.23



รูปที่ 2.23 ระบบที่มีการประมวลผลข้อมูลทางดิจิตอล

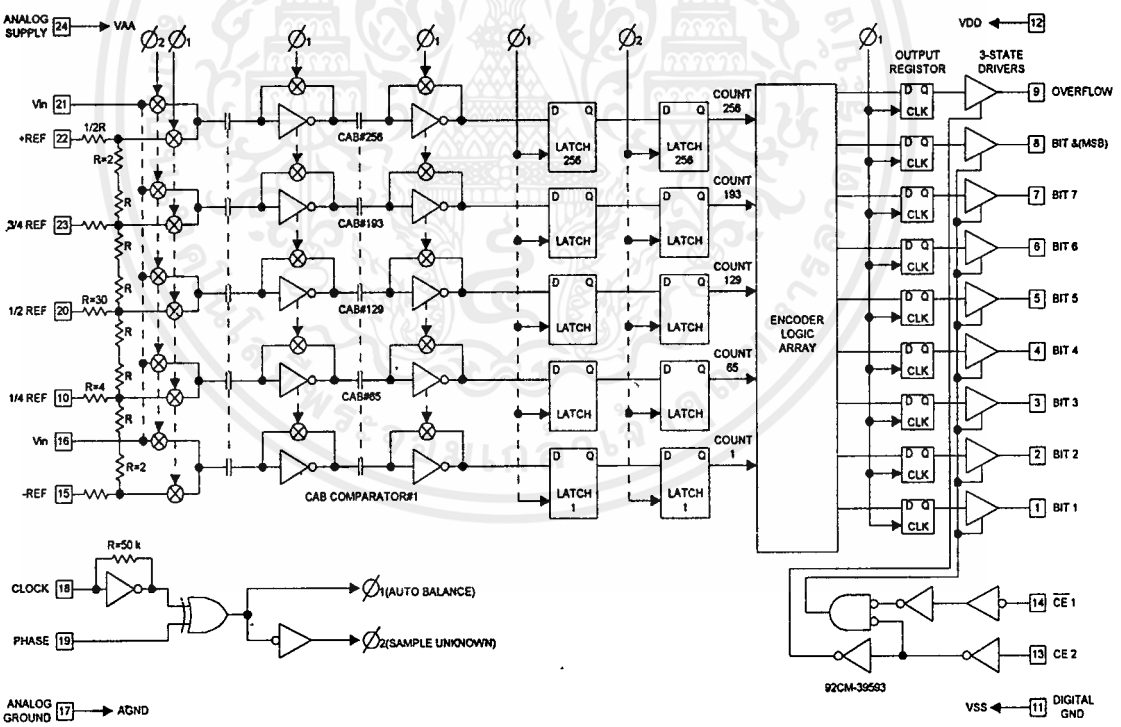
จากรูปที่ 2.23 การเปลี่ยนแปลงทางกายภาพในลักษณะใด ๆ ก็ตาม เช่น อุณหภูมิ ความดัน ความเร็ว จะถูกเปลี่ยนให้มาเป็นสัญญาณไฟฟ้าแบบแอนาลอก โดยทรานสดิวเซอร์ ทฤษฎีการสุ่มที่มีรูปแบบเหมาะสมกับลักษณะทางกายภาพนั้น ๆ จากนั้นสัญญาณทางไฟฟ้าก็จะถูกปรับให้อยู่ในรูปแบบ และขนาดที่เหมาะสมก่อน โดยวงจรต่าง ๆ เช่น วงจรขยาย หรือวงจรกรองสัญญาณเป็นต้น วงจรแซมเปิลแอนด์โฮลด์ จะสุ่มขนาดของสัญญาณแอนาลอกมาแล้วจะทำการโฮลด์สัญญาณนั้นไว้ชั่วขณะ โดยไม่จำเป็นต้องใช้วงจร ADC แล้วข้อมูลทางดิจิตอลจะถูกส่งต่อไปยังบัสของระบบ จากนั้นตัวโปรเซสเซอร์จะทำการประมวลผลข้อมูล แล้วเปลี่ยนผลลัพธ์ข้อมูลกลับมาเพื่อควบคุมกิจการทางกายภาพของระบบโดยผ่านตัวกระทำทางกล (Analog Actuator)

2.5.1 การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

ในโครงงานนี้ วงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลแบบแฟลช ขนาด 8 บิท มีความเร็วในการแปลงสัญญาณสูงมาก มีขนาด 24 ขา คำว่า แฟลช เป็นรูปแบบการแปลงสัญญาณ แอนาลอก เป็นสัญญาณดิจิตอลอีกรูปแบบหนึ่ง ซึ่งแบบแฟลชนี้มีความเร็วในการแปลงสูงกว่าแบบอื่น ๆ

คุณสมบัติของไอซี CA 3318

1. ใช้เทคโนโลยี CMOS/SOS
2. ใช้เทคนิคการแปลงข้อมูลแบบขนาน
3. อัตราการแปลงข้อมูล 15 MSPS ที่ 5 โวลท์
4. ให้สัญญาณเอาท์พุทขนาด 8 บิท
5. ใช้แหล่งจ่ายไฟชุดเดียว 4 ถึง 605 โวลท์
6. แยกระบบกราวด์ของสัญญาณแอนาลอกกับสัญญาณดิจิตอลออกจากกัน
7. กำลังงานสูญเสีย 200 มิลลิวัตต์
8. แรงดันอินพุทอยู่ในช่วง 0 ถึง 6.4 โวลท์
9. สัญญาณนาฬิกา 20 MHz



รูปที่ 2.24 ผังการทำงานภายในของไอซี CA 3318

2.5.2 การแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก

การเปลี่ยน D/A โดยทั่วไปจะสามารถแบ่งตาม ตัวต้านทาน ที่ต่ออยู่สองแบบคือ

1. แบ่งตามน้ำหนักที่อินพุท (Binary Weighted Resistor Ladder)

ในการเปลี่ยนสัญญาณดิจิตอลของเลขไบนารี่ ให้เป็นแรงดันแอนาลอกนั้น จะต้องใช้แรงดันค่าหนึ่ง ๆ สำหรับหนึ่งบิทของเลขไบนารี่ที่เข้ามา ค่าแรงดันที่ตำแหน่งบิทจะเป็นสัดส่วนกับน้ำหนักไบนารี่ ของบิทนั้น ๆ

2. R-2R แลดเดอร์

D/A ชนิดนี้เป็นอีกแบบหนึ่งซึ่งจะแก้ปัญหาในการหาความต้านทานซึ่งมีค่าที่แตกต่างกันมาก ๆ ให้เป็นวงจรที่ใช้ค่าความต้านทานต่างกันเพียง 2 ค่าเท่านั้น โดยการต่อวงจรที่เรียกว่า R-2R Network หรือ R-2R Ladder สัญญาณ อินพุท แต่ละบิทที่ป้อนเข้าวงจร จะทำให้เอาท์พุทที่ได้เปลี่ยนแปลงไป D/A R-2R Ladder เป็นแบบที่ใช้ในการแปลงสัญญาณดิจิตอลเป็นแอนาลอกซึ่งใช้ในปริญญานิพนธ์นี้

2.6 ทฤษฎีของการสุ่มข้อมูล (Sampling)

ในการสุ่มข้อมูลนั้น สัญญาณแอนาลอกจะถูกสุ่มเป็นระยะเวลาคงที่ กลุ่มของสัญญาณที่สุ่มจะแทนด้วยความเร็วสูง ซึ่งเกิดจากการตัดต่อสัญญาณแอนาลอกด้วยระยะเวลาอันสั้น ผลของการสุ่มด้วยความเร็วจะเหมือนกับการคูณขบวนสัญญาณพัลซ์ กับสัญญาณแอนาลอก ซึ่งจะได้สัญญาณที่ มอดูเลท (Modulate) ระหว่างขบวนพัลซ์กับสัญญาณแอนาลอก

ทฤษฎีของการสุ่ม (Nyquist Theorem) ได้กล่าวไว้ว่า " ถ้าหากสัญญาณต่อเนื่องที่มีความถี่ฮาร์โมนิค (Hamonic Frequency) ไม่เกินความถี่ฟันดาเมนทอล แล้วสัญญาณดังกล่าวจะสามารถเปลี่ยนกลับมาเช่นเดิม โดยไม่สูญเสียรายละเอียด หรือผิดเพี้ยนไปถ้าอัตราการสุ่มมากกว่า 2f "

2.7 การสร้างภาพบนจอโทรทัศน์

การสร้างภาพบนจอโทรทัศน์ในที่นี้ ใช้หลักการและวิธีการเช่นเดียวกันกับการออนสกรีนดิสเพลย์ของเครื่องรับโทรทัศน์

วิธีการในการสร้างภาพบนจอโทรทัศน์จะเห็นว่าวงจรจะต้องเชื่อมต่อกับไมโครโปรเซสเซอร์ภายนอกด้วยโดยมีหลักการทำงานดังนี้คือ วงจรจะรับเอาสัญญาณฮอร์พัลซ์ (hor.pulse) และเวอร์พัลซ์ (ver.pulse) จากเครื่องรับโทรทัศน์มา โดยเวอร์พัลซ์จะใช้เป็นสัญญาณในการซิงโครไนเซชั่นการทำงานระหว่างวงจรนี้กับเครื่องรับโทรทัศน์ สัญญาณฮอร์พัลซ์จะใช้เป็นสัญญาณในการกำหนดเส้นภาพที่ต้องการ จะให้ข้อมูลปรากฏออกมา ข้อมูลภาพต่าง ๆ ที่ต้องการจะแสดงจะถูกเก็บไว้ในวิดีโอแรม และจะถูกเลือกข้อมูลออกมาแสดงโดยอาศัยวงจรนับเป็นตัวเลือก

ข้อมูลภาพจากวิดีโอแรมจะเป็นข้อมูลแบบขนาน และจะถูกทำการเปลี่ยนให้เป็นข้อมูลแบบอนุกรมด้วยวงจร PISO เพราะว่าการทำงานของเครื่องรับโทรทัศน์ทำงานเป็นลำดับในแบบอนุกรมข้อมูลที่ถูกเปลี่ยนแล้วนี้จะส่งไปยังวงจรเชื่อมต่อกับเครื่องรับโทรทัศน์ โดยอาศัยสัญญาณนาฬิกาจากวงจรสร้างสัญญาณนาฬิกา ซึ่งจะได้กล่าวถึงในรายละเอียดแต่ละส่วนเพื่อความเข้าใจได้ง่ายขึ้น

2.7.1 วงจรกำเนิดสัญญาณนาฬิกา

สัญญาณนาฬิกาในวงจรนี้จะมีความสำคัญคือ จะเป็นตัวส่งข้อมูลออกจากวงจร PISO ซึ่งวงจรนับที่ใช้ในวิดีโอแรมนี้ ความถี่ของสัญญาณนาฬิกาที่ใช้จะมีค่าประมาณ 6 เมกะเฮิตซ์ วงจรสร้างสัญญาณนาฬิกาจะใช้วงจรอินเวอร์เตอร์เกทแบบชมิตทริกเกอร์อย่างง่าย ๆ ความถี่ของวงจรถูกกำหนดโดยค่าของตัวต้านทานและตัวเก็บประจุ โดยสมการ $T = R \times C$ มีหน่วยเป็นวินาที

2.7.2 วงจรหาร 8

วงจรส่วนนี้จะนำมาสร้างสัญญาณได้แก่ สัญญาณแลตซ์ (latch) เพื่อเป็นสัญญาณนาฬิกาให้กับวงจรแลตซ์ สัญญาณ (shift/load) เพื่อเป็นสัญญาณให้วงจรเปลี่ยนข้อมูลขนานเป็นอนุกรมและให้โหลดข้อมูลจากวงจรแลตซ์ และสัญญาณ CCLK

(column clock) เพื่อใช้เป็นสัญญาณนาฬิกาให้กับวงจรนับเพื่อใช้กำหนดตำแหน่งของวิดีโอแรม

2.7.3 วงจรโมโนสเตเบิ้ล

วงจรโมโนสเตเบิ้ลที่ใช้ในปริญญานิพนธ์นี้มีอยู่สองวงจรคือ วงจรโมโนสเตเบิ้ลทางแนวนอน และวงจรโมโนสเตเบิ้ลทางแนวตั้ง โดยมีสัญญาณอินพุทคือสัญญาณฮอร์พัลซ์และเวอร์พัลซ์ ตามลำดับ โดยที่โมโนสเตเบิ้ลทางแนวนอนจะทำหน้าที่หน่วงเวลาสัญญาณฮอร์พัลซ์ไว้ชั่วระยะเวลาหนึ่งเพื่อเป็นการกำหนดตำแหน่งการสแกนภาพทางแนวนอนให้เริ่มต้นสแกนภาพจากด้านซ้ายของจอที่ตำแหน่งใด ๆ ตามต้องการ โดยการเปลี่ยนแปลงค่าหน่วงเวลาของวงจรโมโนสเตเบิ้ลส่วนวงจรโมโนสเตเบิ้ลทางแนวตั้งจะทำหน้าที่หน่วงเวลาสัญญาณเวอร์พัลส์ไว้ชั่วระยะเวลาหนึ่งเช่นกัน เพื่อเป็นการกำหนดตำแหน่งการเริ่มต้นของภาพในแนวตั้งให้แสดงในตำแหน่งที่ต้องการ โดยการเปลี่ยนแปลงค่าหน่วงเวลาของวงจรโมโนสเตเบิ้ลอีกทั้งยังทำหน้าที่เป็นซิงโครไนเซชั่นระหว่างวงจรสร้างภาพกับเครื่องรับโทรทัศน์ด้วยเพราะถ้าหากไม่มีการซิงโครไนเซชั่นแล้ว การแสดงภาพไม่สามารถทำได้เลยเพราะการสแกนทางแนวตั้งและแนวนอนไม่สัมพันธ์กัน

2.7.4 วงจรฮอร์โมโนสเตเบิ้ล

วงจร H-MONO จะทำหน้าที่หน่วงเวลาสัญญาณฮอร์พัลซ์ เพื่อจะกำหนดตำแหน่งการเริ่มของเส้นภาพ เนื่องจากความยาวของเส้นภาพมีค่าประมาณ 60 ไมโครวินาที ดังนั้นเราจะหน่วงเวลาสัญญาณฮอร์พัลซ์ไว้ประมาณ 20 ไมโครวินาที

2.7.5 วงจรเวอร์โมโนสเตเบิ้ล

วงจรเวอร์โมโนสเตเบิ้ลจะทำหน้าที่ซิงโครไนเซชั่นและกำหนดตำแหน่งการเริ่มต้นของภาพทางแนวตั้งเราทราบแล้วว่าการสแกนใน 1 ฟิลด์จะใช้เวลาประมาณ 20 มิลลิวินาที

2.7.6 วงจรนับ 32 และ 256

วงจรนับที่ใช้จะมีอยู่สองวงจรคือวงจรนับ 32 และวงจรนับ 256 โดยวงจรนับ 32 จะใช้นับตำแหน่งของหน่วยความจำที่จะนำข้อมูลออกมาแสดงในแต่ละเส้นสแกนของการนับจะนับเป็น 32 ไบท์และในแต่ละไบท์จะมี 8 บิท ดังนั้นในแต่ละเส้นภาพจะสามารถแสดงข้อมูลได้เท่ากับ 256 บิทหรือ 256 จุดภาพในหนึ่งเส้นสแกน

ส่วนวงจรนับ 256 จะใช้เลื่อนตำแหน่งของหน่วยความจำครั้งละ 32 ไบท์ เพื่อแสดงข้อมูลในเส้นภาพต่อ ๆ ไปการนับ 256 นี้แสดงว่าวงจรจะสามารถแสดงเส้นภาพได้ทั้งหมด 256 เส้นภาพ นั่นแสดงว่ารายละเอียดของภาพที่ปรากฏบนจอภาพจะเท่ากับ 256 จุดภาพคูณด้วย 256 เส้นภาพ เท่ากับ 65,536 จุดภาพถ้าเทียบเป็นหน่วยความจำก็จะเท่ากับ 8,195 ไบท์หรือ 8 กิโลไบท์

2.7.7 วงจรมัลติเพล็กซ์ชนิดสองทาง

การส่งข้อมูลจากวิดีโอแรมขึ้นไปแสดงบนจอภาพนั้น จะอาศัยผลการนับจากวงจรนับ 32 และวงจรนับ 256 ส่วนข้อมูลภาพที่จะกำหนดบนวิดีโอแรมจะมาจากวงจรไมโครโปรเซสเซอร์

วงจรนับ 32 และวงจรนับ 256 เป็นการเชื่อมต่อกับไมโครโปรเซสเซอร์โดยการต่อวงจร ดังกล่าวกับบัสแอดเดรส (address bus) ของวิดีโอแรม สัญญาณบัสแอดเดรสทั้งสองไม่สามารถเชื่อมต่อเข้ากับวิดีโอแรมโดยตรงจะต้องกระทำผ่านวงจรมัลติเพล็กซ์ชนิดสองทางเพื่อเป็นการเลือกแอดเดรสจากแหล่งใดแหล่งหนึ่งเท่านั้น

ข้อมูลจากหน่วยความจำที่ใช้มีความจุเท่ากับ 8 กิโลไบท์ดังนั้นแอดเดรสบัสของตัวหน่วยความจำจะต้องมี 13 เส้น ตั้งแต่ A0 ถึง A13

2.7.8 วงจรบัฟเฟอร์ชนิดสองทาง

วงจรบัฟเฟอร์ชนิดสองทางมีหน้าที่ เชื่อมโยงบัสข้อมูลระหว่างไมโครโปรเซสเซอร์กับวิดีโอแรม เพื่อใช้ในการอ่านและเขียนข้อมูลลงบนวิดีโอแรม โดยมีการกำหนดทิศทางของข้อมูลได้จากไมโครโปรเซสเซอร์หรือเครื่องซิงเกิ้ลบอร์ด

2.7.9 วิดีโอแรม

วิดีโอแรม เป็นการใช้หน่วยความจำมาแทนตำแหน่งข้อมูลของภาพที่ปรากฏบนจอภาพตามที่เราได้กล่าวไปในตอนต้น ๆ แล้วว่า ความละเอียดของภาพที่จะใช้งานคือมีจำนวน 256 จุดภาพในเส้นภาพหนึ่งเส้น และมีจำนวนเส้นภาพเท่ากับ 256 เส้นภาพ ดังนั้นหน่วยความจำจึงมีความจุเท่ากับ 65,536 บิท หรือ 8 กิโลไบท์

หน่วยความจำขนาด 8 กิโลไบท์จะมีตำแหน่งแอดเดรสตั้งแต่ 0000H จนถึงทั้งหมดของจุดภาพและเส้นภาพในหนึ่งเส้นภาพจะแสดงข้อมูลเท่ากับ 256 จุดภาพซึ่งเท่ากับ 32 ไบท์ ซึ่งสามารถเปลี่ยนแปลงข้อมูลในแต่ละตำแหน่งได้ตามต้องการ

2.7.10 วงจรแลตซ์

วงจรแลตซ์จะทำหน้าที่คงสถานะข้อมูลจากวิดีโอแรมครั้งละ 8 บิทหรือ 1 ไบท์ ก่อนที่จะส่งข้อมูลให้กับวงจรเปลี่ยนข้อมูลจากข้อมูลขนานให้เป็นข้อมูลแบบอนุกรม วงจรแลตซ์

2.8.11 วงจรเปลี่ยนข้อมูลขนานเป็นข้อมูลอนุกรม

วงจรเปลี่ยนข้อมูลขนานเป็นอนุกรมหรือ PISO (parallel in serial out) จะทำหน้าที่เปลี่ยนข้อมูลแบบขนานจากวงจรแลตซ์ให้เป็นข้อมูลแบบอนุกรมเพื่อให้สามารถส่งข้อมูลนี้ไปยังเครื่องรับโทรทัศน์ให้สัมพันธ์กันได้

2.7.12 การส่งข้อมูลเพื่อขับหลอดภาพ

ข้อมูลสุดท้ายจากวงจรเปลี่ยนข้อมูลขนานเป็นอนุกรมจะถูกนำไปขับหลอดภาพ ในที่นี้จะเป็นการขยายสัญญาณไปขับแคโถดของหลอดภาพโทรทัศน์

เนื่องจากแคโถดของหลอดภาพโทรทัศน์จะมีอยู่สามแคโถด คือแคโถดสีแดง เขียวและสีน้ำเงิน การต่อกับแคโถดของหลอดภาพสามารถทำที่สีใดก็ได้ตามต้องการ แต่ส่วนใหญ่มักจะเลือกเป็นสีเขียว เพราะสามารถมองเห็นได้ชัดเจนและสวยมากกว่าสีอื่น ๆ

บทที่ 3
การออกแบบ การสร้าง และการทำงาน

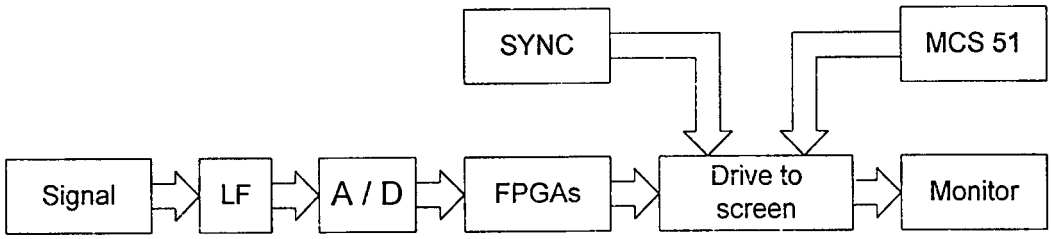
จากที่ได้ทราบการถึงทฤษฎีและหลักการที่เกี่ยวข้องกับการวิเคราะห์แถบความถี่มาบ้างแล้วในบทที่ 2 สามารถออกแบบการทำงานของ เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MH_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs ได้ดังนี้

จากผังการทำงานดังรูปที่ 3.1 สามารถอธิบายการทำงานได้ดังนี้ เริ่มจากสุ่มเอาสัญญาณที่ต้องการจะวิเคราะห์มาช่วงหนึ่ง เพื่อป้อนให้แก่วงจรในส่วนของวงจรกรองความถี่ต่ำที่ย่าน 0-1 MH_z สัญญาณที่ผ่านได้สูงสุดจะมีค่าประมาณ 1 MH_z สัญญาณที่ได้จะผ่านไปยังวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล (A/D) ที่มีเอาท์พุทขนาด 8 บิทหลังจากผ่านวงจรแปลงสัญญาณ แอนาลอกเป็นดิจิตอล แล้วจะนำผลลัพธ์ที่ได้ไปเก็บไว้ในหน่วยความจำตัวที่ 1 โดยที่วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอลและวงจรหน่วยความจำทั้งสองตัวจะได้รับสัญญาณนาฬิกาจาก คริสตอล (X'tal) ความถี่ 5 MH_z

หลังจากนั้นนำค่าที่เก็บอยู่ในหน่วยความจำตัวที่ 1 ไปคำนวณด้วยอัลกอริทึม FFT ของอุปกรณ์ FPGAs แล้วนำผลลัพธ์ที่ได้จากการคำนวณไปเก็บไว้ในหน่วยความจำอีกตัวหนึ่ง แล้วใช้ไมโครคอนโทรลเลอร์เบอร์ MCS 51 เป็นตัวอ่านผลลัพธ์จากหน่วยความจำนำไปประมวลผลและแสดงผลที่จอภาพของโทรทัศน์

ซึ่งเราสามารถแยกออกเป็นโครงสร้างแต่ละส่วนดังนี้

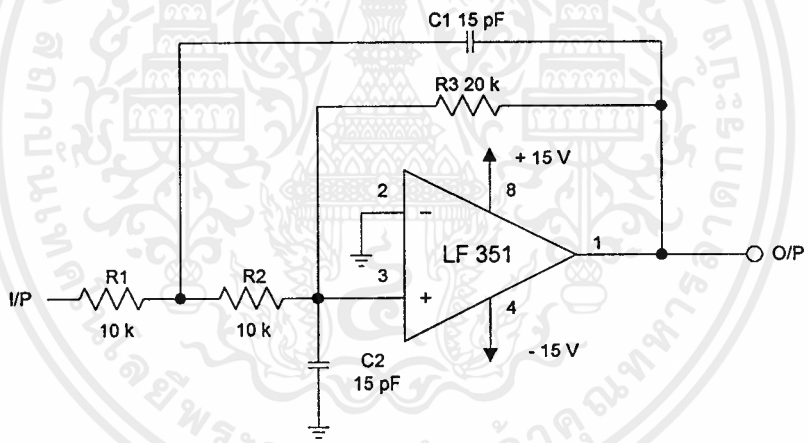
1. วงจรกรองความถี่ต่ำผ่าน
2. วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล
3. วงจรนับตำแหน่งและวงจรหน่วยความจำ
4. วงจรไอซี FPGAs
5. วงจรสร้างภาพบนจอโทรทัศน์
6. ไมโครคอนโทรลเลอร์ เบอร์ MSC 51



รูปที่ 3.1 บล็อคไดอะแกรมการทำงานของวงจรเครื่องวิเคราะห์แถบความถี่ย่าน 0-1 M H_z โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs

3.1 การออกแบบวงจรกรองความถี่ต่ำ

การออกแบบวงจรกรองความถี่ต่ำผ่านที่ใช้งานจริง จะใช้ออปแอมป์เบอร์ LF 351 ทำการต่อแบบ Sallen and Key



รูปที่ 3.2 วงจรกรองความถี่ 1 MH_z

สูตรการคำนวณค่าความถี่ f_c จะได้

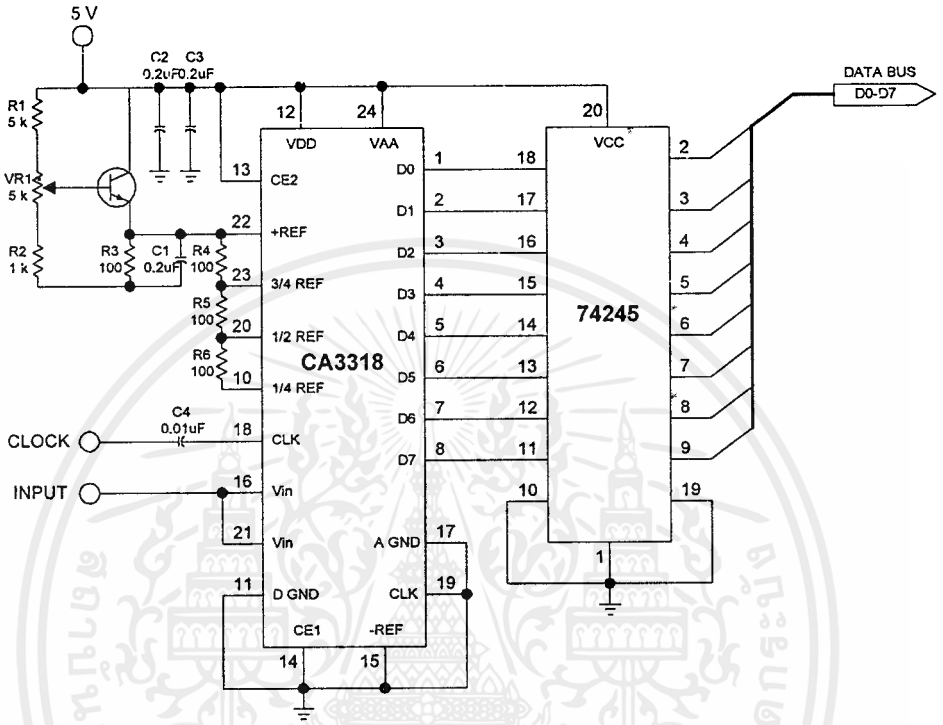
$$fc = \frac{1}{2\pi\sqrt{R1 \times R2 \times C1 \times C2}} \tag{3.1}$$

$$fc = \frac{1}{2\pi\sqrt{10k \times 10k \times 15pF \times 15pF}}$$

$$fc = 1.061\,MHz$$

3.2 การออกแบบวงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

3.2.1 การสร้างวงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล



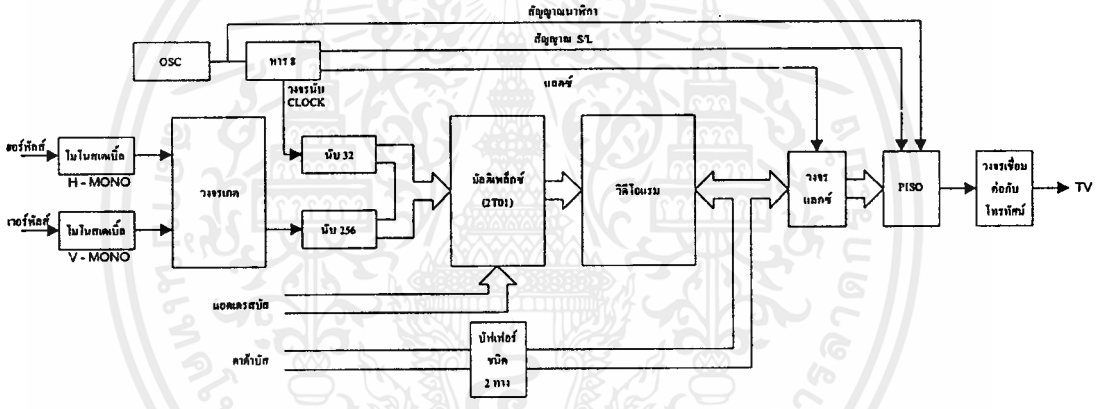
รูปที่ 3.3 วงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

3.2.2 การทำงานของวงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

สัญญาณวีดีโออินพุทจะถูกป้อนเข้าที่ขา 21 และขา 16 ที่ขา 22 จะถูกปรับให้อยู่ในช่วง 5 โวลท์ โดยใช้วงจรควบคุมแรงดันที่ขา 3/4 REF,1/2 REF และ 1/4 REF จะถูกต่อโดยชุดแบ่งแรงดัน เพื่อเป็นแรงดันอ้างอิงให้ชุดสวิทซ์อิเล็กทรอนิกส์ภายในตัวไอซี ซึ่งมีชุดสร้างแรงดันอ้างอิงทั้งทางด้านบวกและลบ โดยที่ภายในตัวไอซีจะมีชุดสวิทซ์อิเล็กทรอนิกส์อยู่ 256 ชุด โดยจะนำสัญญาณอินพุทมาเปรียบเทียบกับแรงดันแรงดันอ้างอิงของตัวเปรียบเทียบภายใน ข้อมูลทั้งหมดที่ได้จากตัวเปรียบเทียบ (เป็น 0 หรือ 1) แล้วส่งไปยัง ดีฟลิปฟลอปทั้ง 256 ชุด โดยตรง เป็นไปในลักษณะตัวเปรียบเทียบกับชุดที่ 1 แล้วส่งข้อมูลเข้ามายัง ดีฟลิปฟลอปชุดที่ 1 โดยที่ฟลิปฟลอปทำหน้าที่เป็นชิฟต์รีจิสเตอร์ทำงานในโหมดสัญญาณนาฬิกา (ตอบสนองต่อสัญญาณนาฬิกาเฉพาะ

ช่วงขอบขาขึ้นและขอบขาลงของพัลซ์เท่านั้น) ทำการแลตซ์ (LATCH) ข้อมูลไว้ชั่วขณะจนกว่าจะมีข้อมูลใหม่เข้ามาจึงจะเลื่อนข้อมูลนั้นส่งเข้าชุดเข้ารหัสเพื่อแปลงข้อมูลทั้ง 256 ค่าออกมาเป็นข้อมูลดิจิตอลขนาด 9 บิท (รวมบิทส่วนเกิน) ส่งต่อไปยังเอาท์พุทรีจิสเตอร์ ซึ่งใช้ดีฟลิปฟลอปทำหน้าที่นี้อีกก่อนส่งไปยังตัวขับ 3 สถานะเป็นเอาท์พุทต่อไปเอาท์พุทนี้สามารถควบคุมได้ด้วย CE_1 และ CE_2 การทำงานทั้งหมดนี้เราสามารถควบคุมได้ที่ขาควบคุมเฟส (ขา 19)

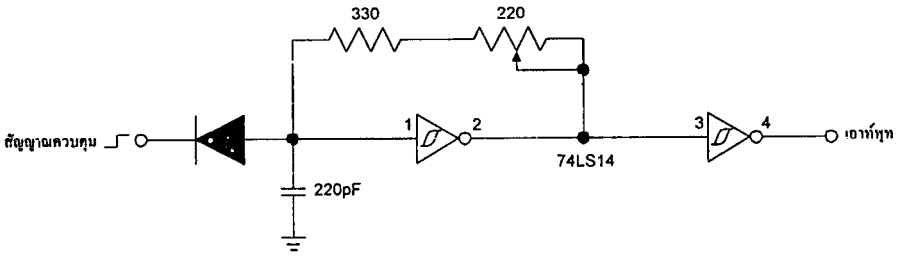
3.3 การออกแบบ การสร้างและการทำงานของวงจรสร้างภาพบนจอโทรทัศน์



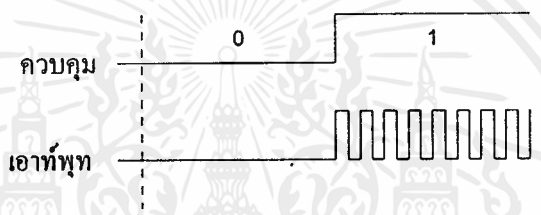
รูปที่ 3.4 บล็อคไดอะแกรมการสร้างภาพบนจอโทรทัศน์

3.3.1 การออกแบบวงจรกำเนิดสัญญาณนาฬิกา

ที่อินพุทของวงจรจะเห็นว่ามีไดโอดอยู่ด้วย ซึ่งมีจุดประสงค์เพื่อหยุดการทำงานของวงจร สร้างสัญญาณนาฬิกาในทุก ๆ เส้นภาพขณะเริ่มต้น กล่าวง่าย ๆ คือเป็นการทำให้การเริ่มต้นการสร้างสัญญาณนาฬิกาในทุก ๆ เส้นภาพเกิดขึ้นพร้อมกันทุก ๆ ครั้ง ดังแสดงวงจรในรูปที่ 3.5 และการควบคุมการสร้างสัญญาณนาฬิกาในรูปที่ 3.6



รูปที่ 3.5 วงจรสร้างสัญญาณนาฬิกามีขาควบคุมการทำงาน



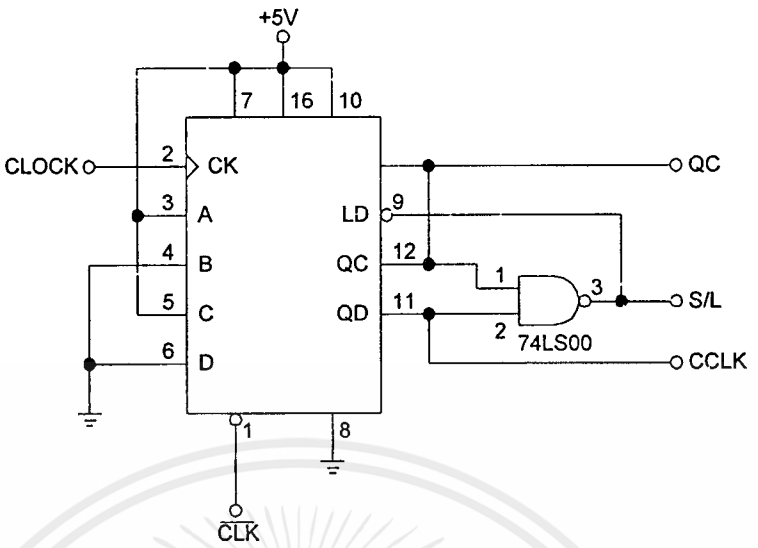
รูปที่ 3.6 สัญญาณทางเอาท์พุทของวงจร รูปที่ 3.5

3.3.2 การออกแบบวงจรหาร 8

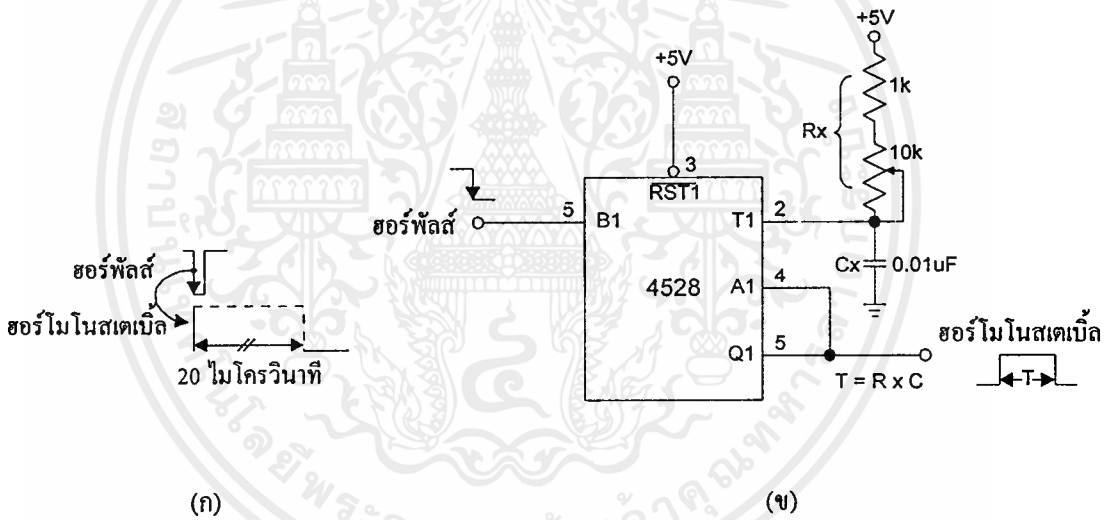
วงจรนับ ในส่วนนี้แสดงรายละเอียดในรูปที่ 3.7 จากวงจรนี้จะใช้ไอซีเบอร์ 74LS161 โดยจัดให้เป็นวงจรหาร 8 กำหนดให้มีการนับอยู่ในช่วง 5 ลงมาถึง C โดยให้ Q_C เป็นสัญญาณแลตซ์ Q_D เป็นสัญญาณ CCLK และนำเอา Q_C กับ Q_D มาแนนด์กันได้เป็นสัญญาณ S/L

3.3.3 วงจรฮอร์โมโนสเตเบิ้ล

การออกแบบจะใช้ไอซีเบอร์ 4528 อินพุทสัญญาณฮอร์พัลซ์เข้าที่อินพุท (ขา 5) เอาท์พุท (ขา 6) จะต่อเข้ามาที่อินพุท (ขา 4) ของไอซี 4528 เป็นการป้องกันการกระตุ้นซ้ำ (retrigerable) ส่วนคาบเวลาของโมโนสเตเบิ้ลนั้นถูกกำหนดโดยค่าของ R_X และ C_X จะเห็นได้ว่าวงจรจะได้รับการกระตุ้นเมื่ออินพุทเป็นขอบขาลง (“0”) ดังแสดงวงจรในรูปที่ 3.8



รูปที่ 3.7 วงจรหาร 8 สร้างสัญญาณ latch, S/L และ CCLK

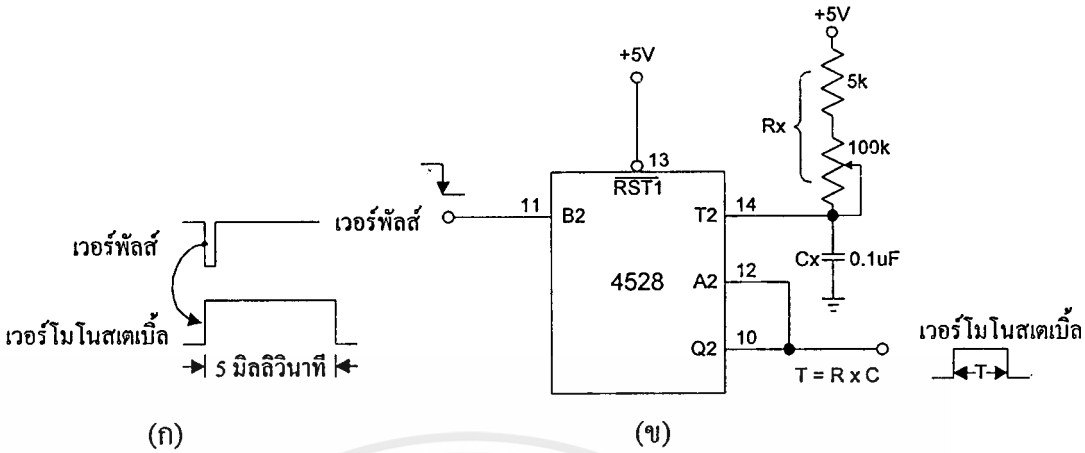


รูปที่ 3.8 (ก) ผังเวลาสัญญาณฮอร์โมโนสเตเบิ้ล

(ข) วงจรฮอร์โมโนสเตเบิ้ล

3.3.4 วงจรเวอร์โมโนสเตเบิ้ล

การออกแบบวงจรโมโนสเตเบิ้ลกำหนดให้มีคาบเวลาประมาณ 5 มิลลิวินาที โดยใช้ไอซี 4528 เช่นเดียวกับวงจรฮอร์โมโนสเตเบิ้ลและมีลักษณะการทำงานรวมทั้งวงจรที่เหมือนกัน เพียงต่างกันที่คาบการหน่วงเวลาเท่านั้นดังจะเห็นได้จากวงจรในรูปที่ 3.9

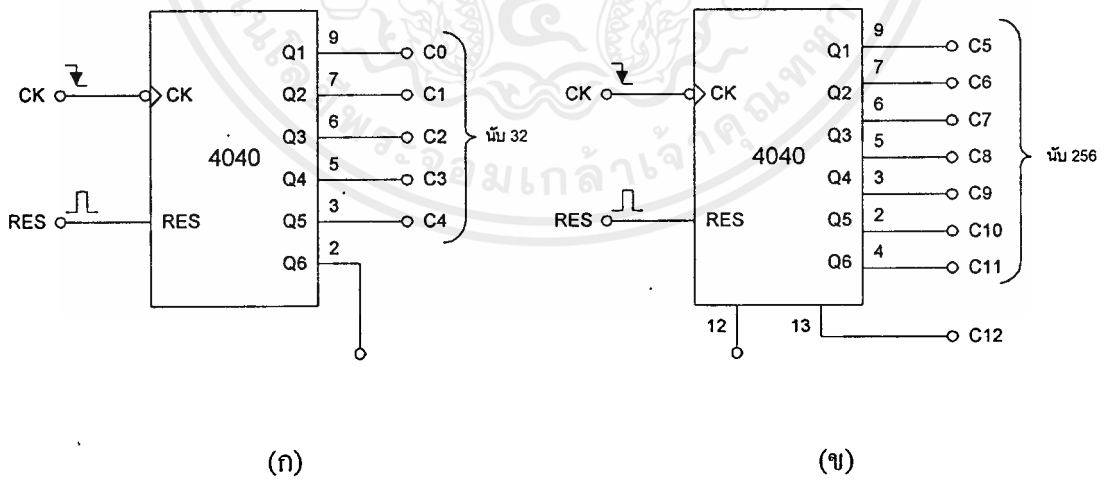


รูปที่ 3.9 (ก) ผังเวลาของสัญญาณเวอร์โมโนสเตเบิ้ล

(ข) วงจรเวอร์โมโนสเตเบิ้ล

3.3.5 วงจรนับ 32 และ 256

การออกแบบวงจรนับจะใช้ไอซีวงจรนับเบอร์ 4040 ซึ่งเป็นวงจรนับฐาน 12 สเตต ต่อเป็นวงจรนับ 32 และ 256 ตามลำดับดังแสดงในรูปที่ 3.10 วงจรส่วนนี้การออกแบบสามารถใช้วงจรพื้นฐานได้ทันที

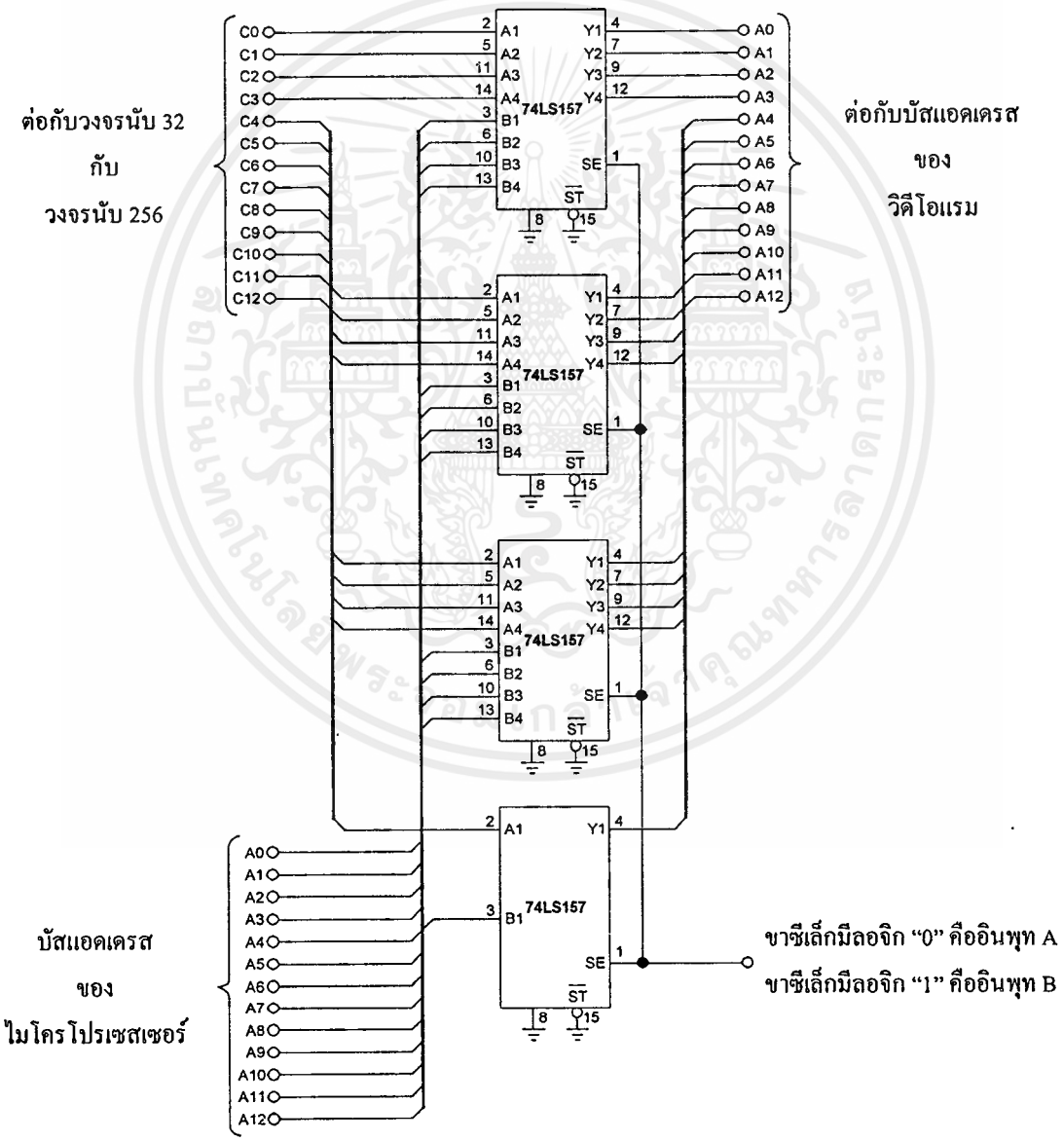


รูปที่ 3.10 (ก) วงจรนับ 32 สำหรับข้อมูลในแต่ละเส้น

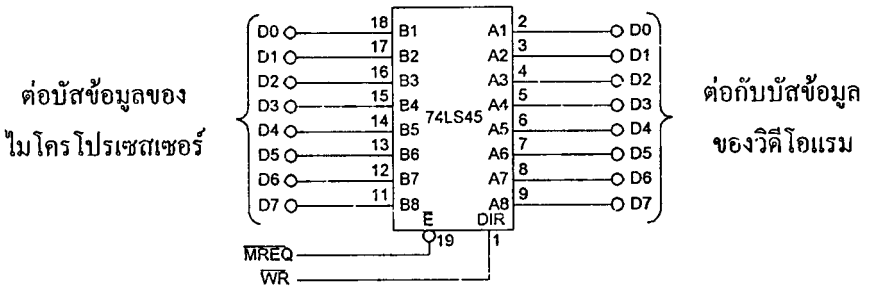
(ข) วงจรนับ 256 สำหรับจำนวนเส้นภาพ

3.3.6 วงจรมัลติเพล็กชนิดสองทาง

การออกแบบวงจรมัลติเพล็กชนิดสองทางในที่นี้จะใช้ไอซีเบอร์ 74LS157 ซึ่งเป็นไอซี มัลติเพล็กชนิดสองทางสี่ชุดจำนวนสี่ตัว อินพุทของวงจรจะมีสองอินพุท คือ A และ B โดยสามารถเลือกสัญญาณจากอินพุท A และ B ได้โดยการควบคุมที่ขาซีเล็ก (select) ถ้าขาซีเล็กมีสภาวะลอจิกเป็น “0” วงจรจะเลือกอินพุท A และหากขาซีเล็ก มีสภาวะลอจิกเป็น “1” วงจรจะเลือกอินพุท B ดังแสดงวงจรส่วนนี้ในรูปที่ 3.11



รูปที่ 3.11 วงจรมัลติเพล็กชนิดสองทาง



รูปที่ 3.12 วงจรบัฟเฟอร์ชนิดสองทาง

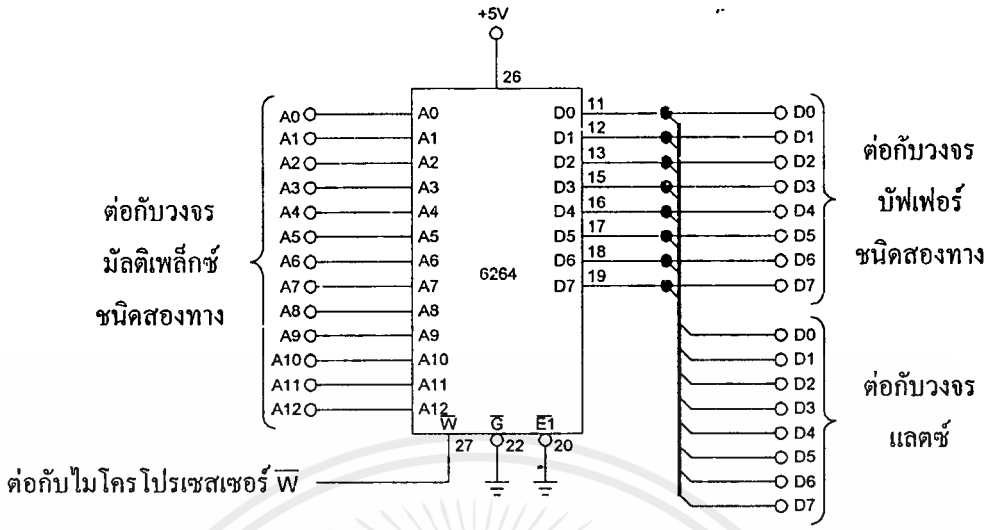
3.3.7 วงจรบัฟเฟอร์ชนิดสองทาง

วงจรบัฟเฟอร์ชนิดสองทางในที่นี้ได้ใช้ไอซีเบอร์ 74LS245 โดยอินพุท B จะต่อเข้ากับไมโครโปรเซสเซอร์ ส่วนอินพุท A ต่อกับบัสข้อมูลของวิดีโอแรมทิศทางการเคลื่อนที่ของข้อมูลจะถูกกำหนดโดยขา DIR เมื่อขา DIR มีลอจิกเป็น “1” ข้อมูลจะผ่านจาก A ไป B และหากขา DIR จะมีลอจิกเป็น “0” ข้อมูลจะผ่านจาก B ไปยัง A ขา DIR นี้จะต่อกับขาสัญญาณ (write) ของไมโครโปรเซสเซอร์เพื่อบังคับทิศทางของข้อมูล

ส่วนขา E จะถูกต่อกับขา MREQ (memory request) ของไมโครโปรเซสเซอร์เพื่อสั่งให้ไอซีเบอร์ 74LS245 ทำงานหรือไม่ทำงานตามต้องการ ในภาวะปกติหากขา E ได้รับลอจิก “1” ตัวไอซีจะไม่ทำงานซึ่งเปรียบเสมือนวงจรถูกแยกตัวออกจากระบบ และถ้าเมื่อใดขา E นี้ ได้รับลอจิก “0” ตัวไอซีจะทำงานและขณะนี้จะสามารถบังคับทิศทางได้ที่ขา DIR ทันทีวงจรบัฟเฟอร์ชนิดสองทางนี้แสดงดังรูปที่ 3.12

3.3.8 วิดีโอแรม

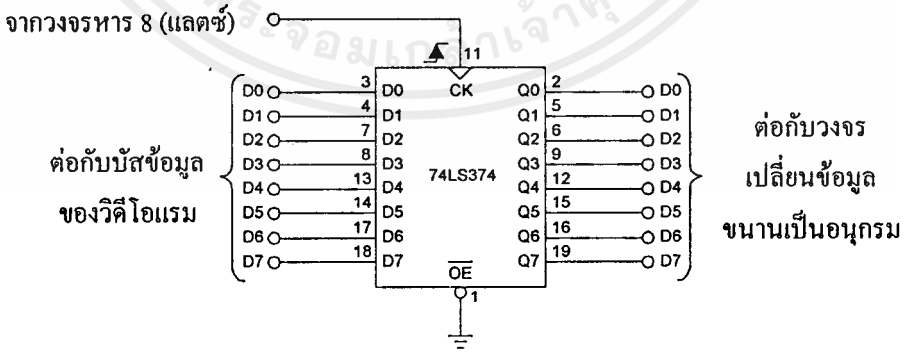
หน่วยความจำวิดีโอแรมที่ใช้ในที่นี้มีความจุเท่ากับ 8 กิโลไบท์ ดังนั้นจะใช้ไอซี RAM เบอร์ 6264 ซึ่งเป็นหน่วยความจำแบบขนาน ความจุเท่ากับ 8 กิโลไบท์ ใช้แหล่งจ่ายไฟประมาณ 5 โวลท์ สามารถเชื่อมต่อกับไอซี TTL ได้โดยตรง วงจรของวิดีโอแรมแสดงดังรูปที่ 3.13



รูปที่ 3.13 วงจรหน่วยความจำของวิดีโอแรม

3.3.9 วงจรแลตซ์

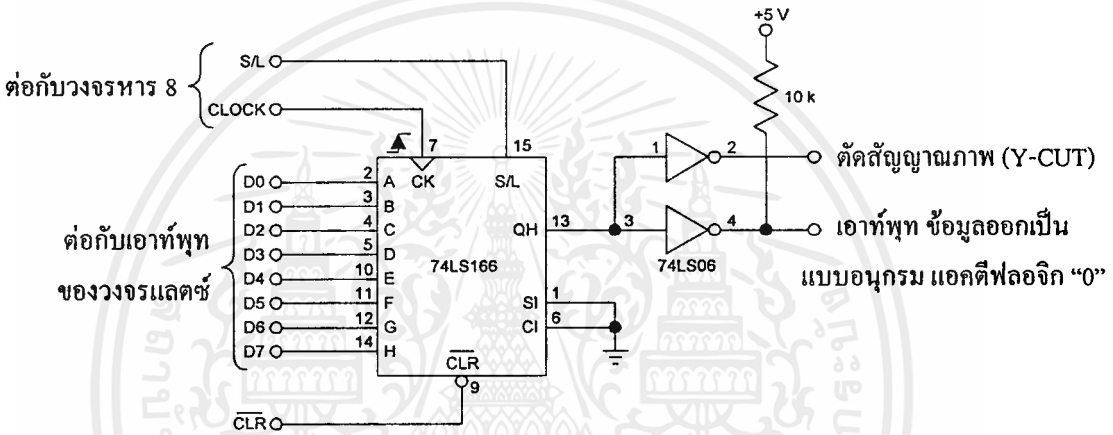
ในที่นี้จะใช้ D-ฟลิปฟลอป เบอร์ 74LS374 ซึ่งภายในมี D-ฟลิปฟลอป อยู่ 8 ตัว เท่ากับ 8 บิทพอดี ดังแสดงไว้ในรูปที่ 3.14 ซึ่งข้อมูลทางอินพุทจะถูกแลตซ์ก็ต่อเมื่อมีการเปลี่ยนแปลงของสัญญาณแลตซ์เป็นขอบขาขึ้น สัญญาณแลตซ์นี้จะได้มาจากวงจรหาร 8



รูปที่ 3.14 วงจรแลตซ์ข้อมูลขนาด 8 บิท

3.3.10 วงจรเปลี่ยนข้อมูลขนานเป็นข้อมูลอนุกรม

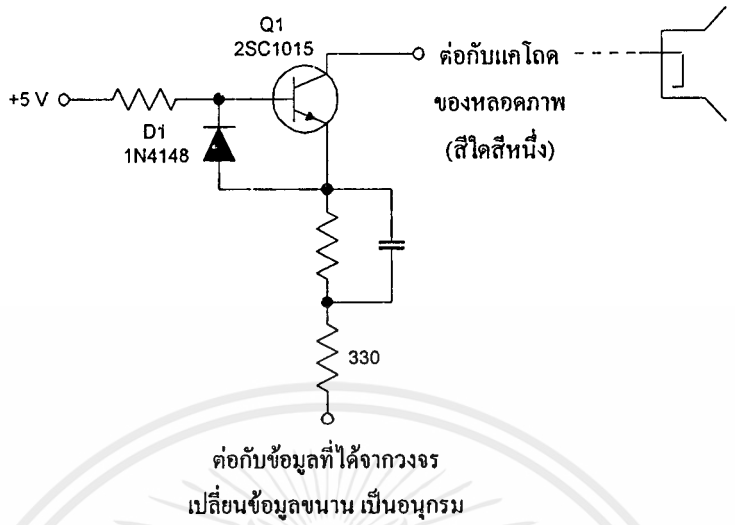
การทำงานของวงจรนี้จะสัมพันธ์กันกับแผนผังเวลาในวงจรหาร 8 โดยสัญญาณ S/L จะเป็นสัญญาณในการโหลดข้อมูล และสัญญาณ CLOCK จะทำหน้าที่ส่งข้อมูลออกไปยังเอาท์พุทเพื่อส่งไปยังเครื่องรับโทรทัศน์ต่อไป วงจรเปลี่ยนข้อมูลขนานเป็นอนุกรมนี้แสดงไว้ในรูปที่ 3.15



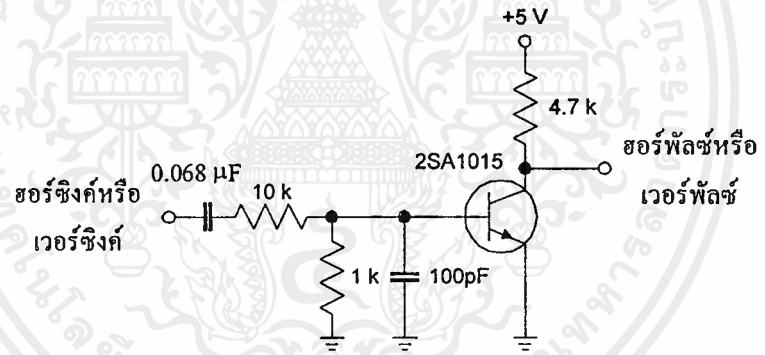
รูปที่ 3.15 วงจรเปลี่ยนข้อมูลขนานเป็นอนุกรม 8 บิท

3.3.11 การส่งข้อมูลเพื่อขับหลอดภาพ

วงจรในรูปที่ 3.16 เป็นวงจรขับหลอดภาพ จากรูปทรานซิสเตอร์ Q1 ต่อวงจรเป็นสวิตซ์มีไบแอสให้กับขาเบสให้พร้อมรับการกระตุ้นหรือควบคุมจากสัญญาณทางขาอิมิตเตอร์ โดยจากข้อมูลที่ได้สุดท้ายจะเห็นว่าข้อมูลนี้แอคตีฟลอจิก "0" ข้อมูลนี้จะใช้ป้อนเข้าที่ขาอิมิตเตอร์ หากข้อมูลมีลอจิก "1" ทรานซิสเตอร์ไม่ทำงานจึงไม่มีแสงปรากฏบนจอภาพ (แสงของวงจรสร้างภาพ) หากข้อมูลมีลอจิก "0" ทรานซิสเตอร์ทำงานทำให้แรงดันที่ขาแคโถดของหลอดภาพต่ำลงใกล้ 0 โวลท์ ทำให้มีแสงปรากฏบนจอภาพตามข้อมูลในตำแหน่งนั้น ๆ

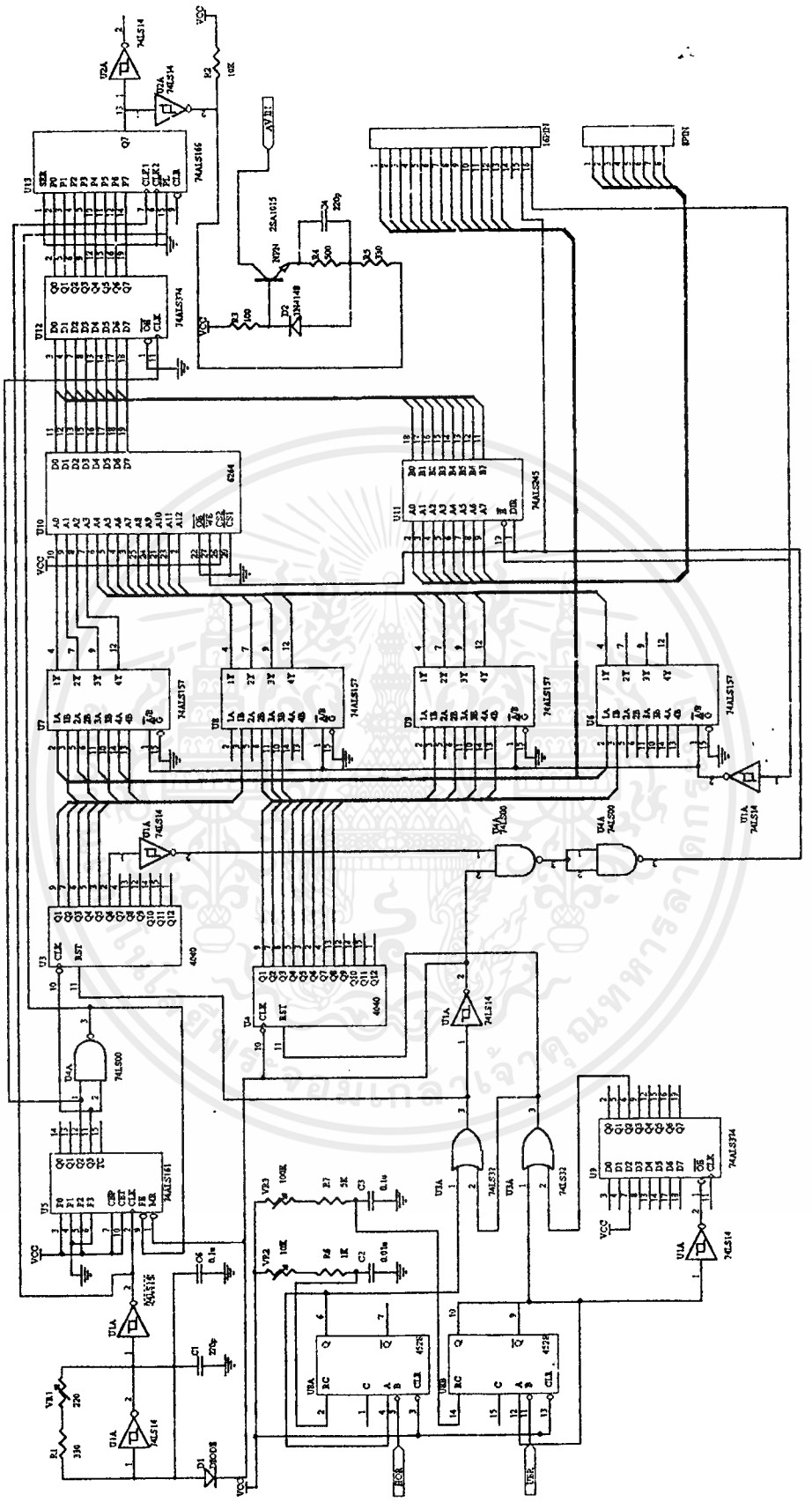


รูปที่ 3.16 วงจรขับหลอดภาพ



รูปที่ 3.17 วงจรสร้างสัญญาณ ฮอร์พัลซ์ และสัญญาณเวอร์พัลซ์

ถึงขั้นตอนนี้แล้วก็จะได้วงจรสมบูรณ์ของวงจรสร้างภาพบนจอโทรทัศน์เป็นดังรูปที่ 3.18 ที่เหลืออยู่ก็จะเป็นการนำไปเชื่อมต่อกับเครื่องรับโทรทัศน์เท่านั้น การเชื่อมต่อมีอยู่สองส่วนคือ ส่วนของสัญญาณ OSD กับสัญญาณ Y-CUT และอีกส่วนหนึ่งคือการเชื่อมต่อสัญญาณซิงค์ การเชื่อมต่อสัญญาณ OSD สามารถทำได้โดยตรงที่แคโถดของหลอดภาพดังวงจรที่ 3.18 แสดงไว้แล้ว ส่วนของการต่อสัญญาณ Y-CUT สามารถทำได้โดยการต่อเข้ากับขาเบสของทรานซิสเตอร์ในภาควิดีโอแอมป์ของโทรทัศน์ได้ทันที



รูปที่ 3.18 วงจรสมบูรณ์ของวงจรการสร้างภาพบนจอโทรทัศน์

ส่วนการต่อระบบของสัญญาณซิงค์จำเป็นจะต้องมีวงจรสร้างสัญญาณฮอร์พัลซ์และวงจรการสร้างสัญญาณเวอร์พัลซ์เสียก่อนดังแสดงวงจรไว้ในรูปที่ 3.18 โดยอินพุทจะเป็นสัญญาณเวอร์ซิงค์และฮอร์ซิงค์จากเครื่องรับโทรทัศน์เอง วงจรในส่วนนี้จะใช้เหมือนกันทั้งสองวงจร สำหรับสัญญาณฮอร์พัลซ์และสัญญาณเวอร์พัลซ์ ถ้าในเครื่องรับที่จะนำมาใช้มีระบบ OSD อยู่แล้ว วงจรดังกล่าวก็ไม่ต้องสร้างเพิ่มเติม เราสามารถนำสัญญาณฮอร์พัลซ์และสัญญาณเวอร์พัลซ์จากคอนโทรลเลอร์ของเครื่องรับนั้นมาใช้ได้โดยตรง

3.4 โปรแกรม

3.4.1 อัลกอริทึมโปรแกรมลบเลขขนาด 8 บิท

การลบเลขในระบบคอมพิวเตอร์นั้นจะใช้วิธีการ Complement เข้ามาช่วยในการแก้ปัญหา การลบเลขขนาด 8 บิทนั้นจะใช้วิธีการ 1'Complement เข้ามาช่วยในการลบเลขโดยมีหลักการดังนี้

เริ่มต้น รับค่าอินพุทเข้ามา 2 ค่า ค่าละ 8 บิท (a_sub,b_sub) นำค่าอินพุทที่รับเข้ามาไปเก็บไว้ในตัวแปรขนาด 16 บิท โดยนำไปเก็บไว้ใน 8 บิทล่าง ส่วน 8 บิทบนเราจะเก็บค่า '0' ไว้ จากนั้นนำค่าข้อมูลตัวที่จะนำมาลบทำวิธีการ 1'Complement โดยการต่อ not gate ให้กับทุก ๆ บิทของตัวลบ (not sort_sub2) เพื่อทำ 1'Complement หลังจากเสร็จแล้วจะนำค่าที่ได้ไปทำการบวกกับค่าตัวตั้ง (sort_sub1) การบวกก็จะใช้วิธีการคล้ายกันกับโปรแกรมการบวกเลขขนาด 8 บิท คือทำการบวกที่บิท 0 ก่อน เพื่อที่จะหาตัวทดแรก (cin) เมื่อได้ตัวทดแรกแล้วจึงริ่มบวกเลขส่วนที่เหลือโดยใช้ xor gate เปรียบเทียบบิท และ and gate หาค่าตัวทด เมื่อครบทั้ง 16 บิท บิทสุดท้ายที่ได้จะเป็นตัวชี้ว่าค่าที่ลบกันแล้วนั้นมีเครื่องหมายเป็นลบหรือบวก โดยถ้าค่าตัวทดตัวสุดท้าย (cin2 บิทสุดท้าย) เป็น 1 จะเป็นตัวชี้ว่ามีผลลัพธ์เป็นบวก ถ้าตัวทดตัวสุดท้ายเป็น 0 จะมีผลลัพธ์เป็นลบ

กรณีถ้าตัวทดเป็น 1

เราจะนำค่าที่ได้นี้ไปทำการบวกกับค่าผลลัพธ์ (sum2) ก่อนที่จะนำไปทำการ 1'Complement กลับ เช่น ผลลัพธ์ที่ได้เป็น

ตัวทด ผลลัพธ์

1 0101 +

→ 1

0110

ผลลัพธ์ที่ได้จะนำไป xor กับค่า 1'Complement ตัวทดตัวสุดท้าย (not cin2) จะได้ผลลัพธ์สุดท้าย

กรณีถ้าตัวทดเป็น 0 เช่น

ตัวทด ผลลัพธ์

0 0101 +

→ 0

0101

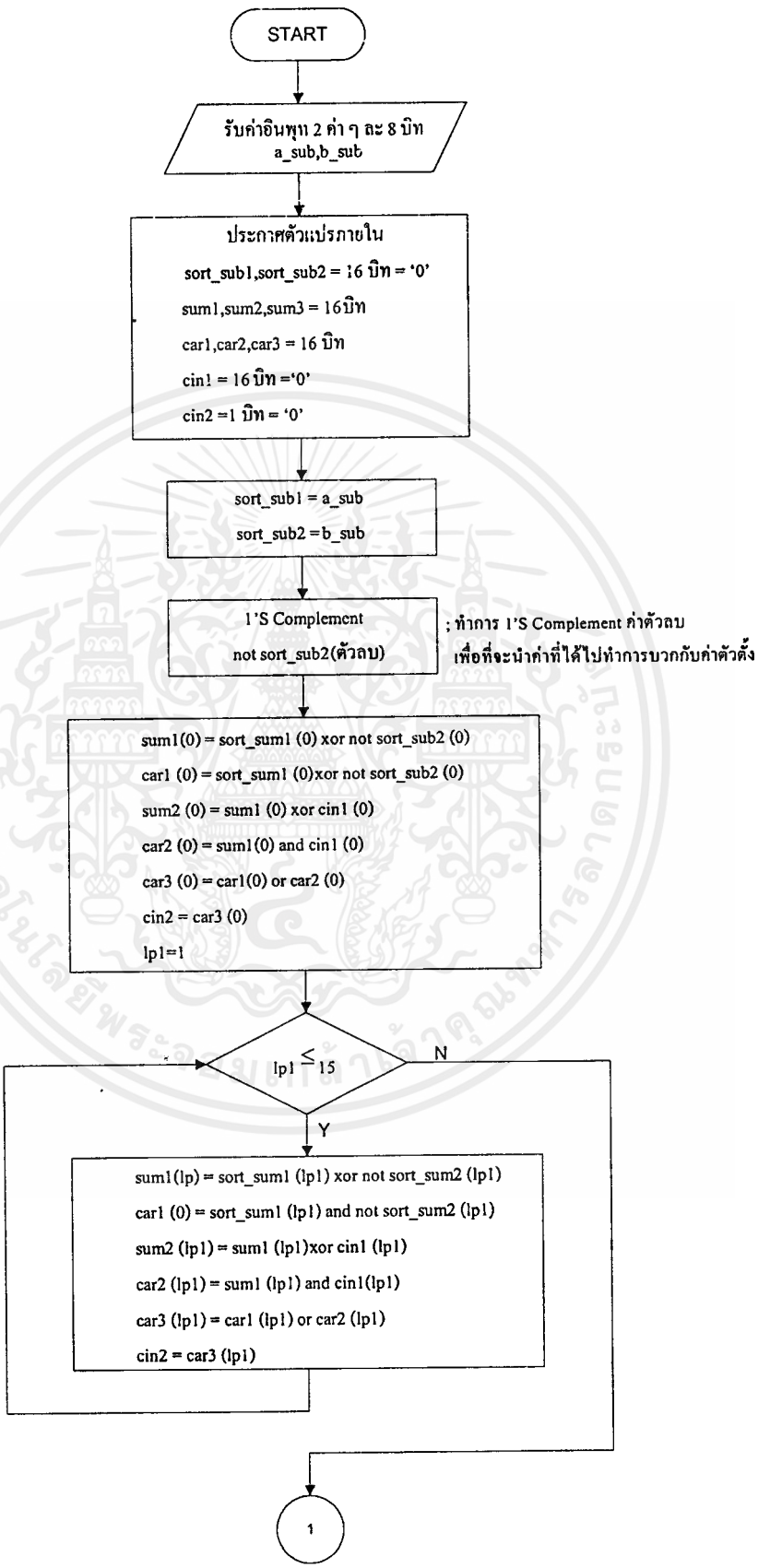
ผลลัพธ์ที่ได้นำไป xor กับ 1'Complement ตัวทดตัวสุดท้าย ในที่นี้ตัวทด = 0 จะได้ ตัวทด 0 = 1'S Complement = 1

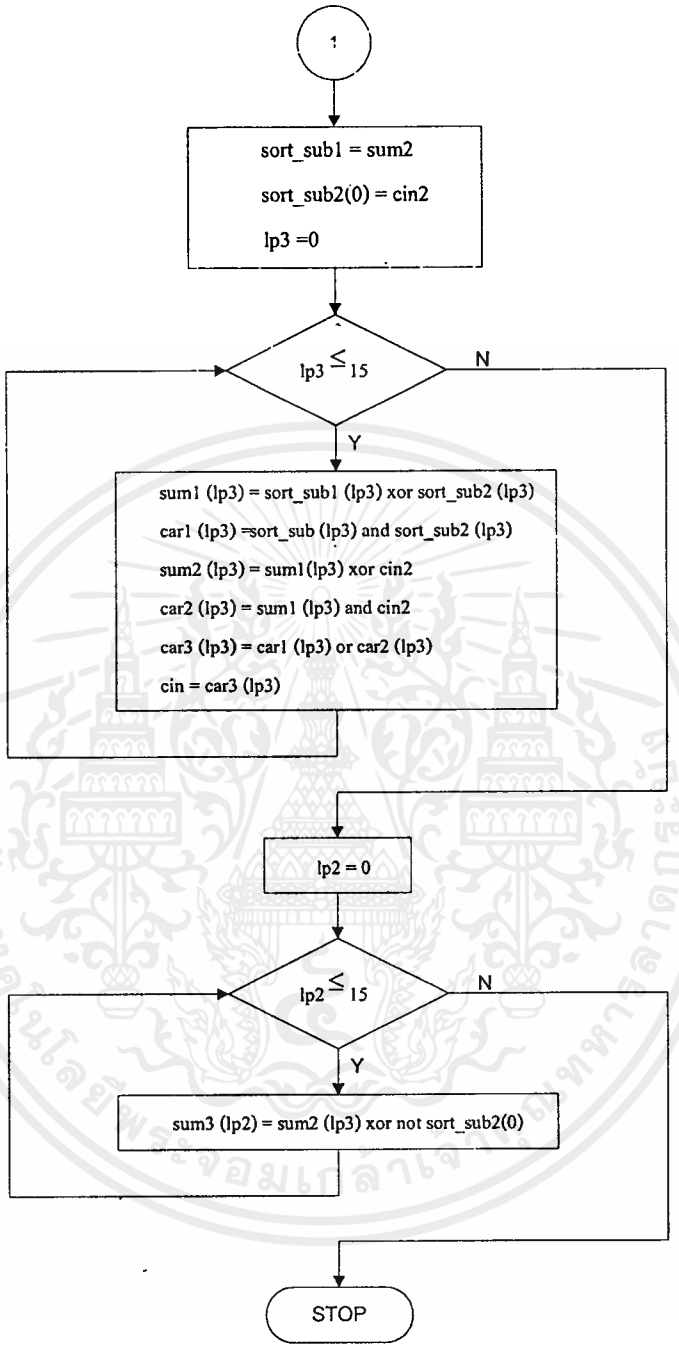
0101 xor

1

1010

ผลลัพธ์สุดท้ายที่ได้จะเป็นคำตอบ แต่จะเป็นคำตอบที่ติดลบ





รูปที่ 3.19 ผังงานโปรแกรมลบเลขขนาด 8 บิท

3.4.2 อัลกอริทึมโปรแกรมคูณเลขขนาด 8 บิท

การคูณเลขขนาด 8 บิท จะใช้วิธีการบวกเลขเข้ามาช่วยในการคำนวณพร้อมทั้งการเลื่อนตำแหน่งข้อมูล เริ่มต้นรับข้อมูล 2 ค่า ค่าละ 8 บิท in_pro1,in_pro2 แล้วนำไปเก็บไว้ในตัวแปรขนาด 16 บิท เก็บไว้ที่ 8 บิทล่าง การคูณเริ่มจากการตรวจสอบค่าแต่ละบิทของตัวคูณ in_pro2(lp) โดยที่ใช้ lp เป็นตัวบ่งชี้ตำแหน่งข้อมูลที่ lp ชี้จะอ้างถึงตำแหน่งที่จะเลื่อนด้วย หากค่า in_pro2 มีค่าเป็น '1' จะนำค่า in_pro1 ไปเก็บไว้ในตัวแปรที่ lp ชี้อยู่คือ pro1(lp+1) = in_pro1(i) เมื่อเลื่อนข้อมูลเสร็จจึงนำไปบวกกับค่าข้อมูลที่ได้รับการบวกก่อนหน้านั้นแล้ว เช่น

1. ตรวจสอบบิทแรกของตัวคูณมีค่าเท่าใด ในที่นี้มีค่าเท่ากับ '1' นำค่าทั้งตัวตั้งเก็บในตัวแปรจะได้ pro1 = 1101 และกำหนดค่าเริ่มต้นตัวแปรเก็บผลลัพธ์จะได้ pro3 = 0000 จากนั้นนค่าที่มีมาบวกกัน

$$\begin{aligned} pro3 &= pro3+pro1 \\ &= 0000+1101 \\ &= 1101 \end{aligned}$$

ในบิทที่ 2 มีค่าเป็น 0 จะได้ pro1 = 0000$\underline{0}$ (เลื่อนข้อมูล 1 ตำแหน่ง)

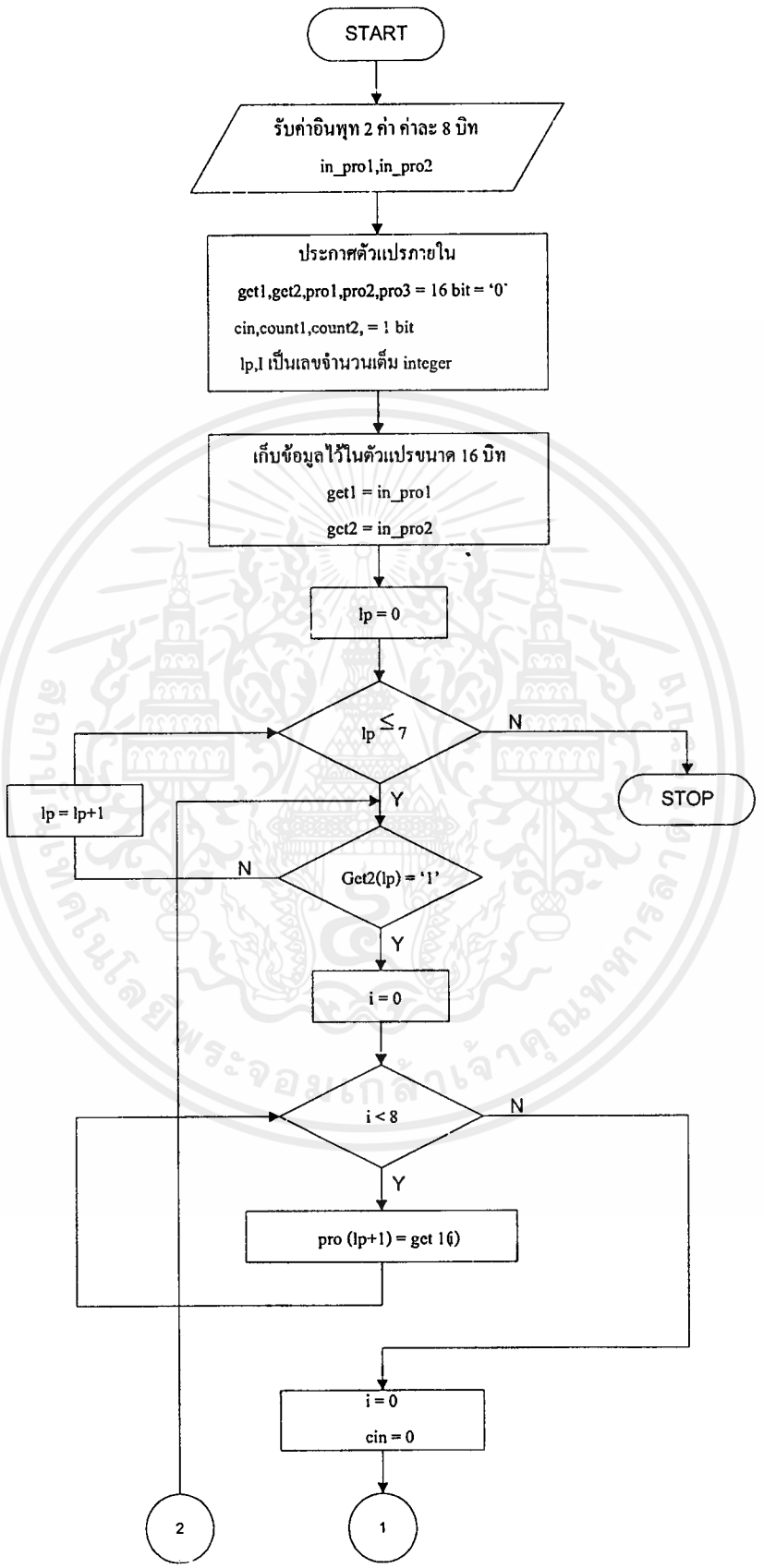
$$\begin{aligned} pro3 &= pro3+pro1 \\ &= 1101+0000\underline{0} \\ &= 01101 \end{aligned}$$

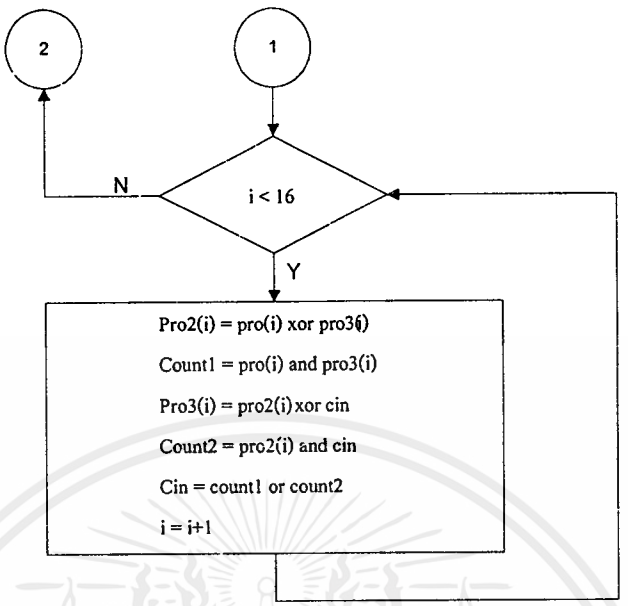
ในบิทที่ 3 มีค่าเป็น 1 จะได้ pro1 = 1101$\underline{00}$ (เลื่อนข้อมูล 1 ตำแหน่ง)

$$\begin{aligned} pro3 &= pro3+pro1 \\ &= 01101+110100 \\ &= 111001 \end{aligned}$$

ในบิทที่ 4 มีค่าเป็น 0 เมื่อนำมาบวกกันจะได้เท่ากับ pro3

สรุป ผลลัพธ์ที่ได้จะเป็นการรวมข้อมูลแต่ละครั้งที่มีการเลื่อนตำแหน่งของข้อมูล



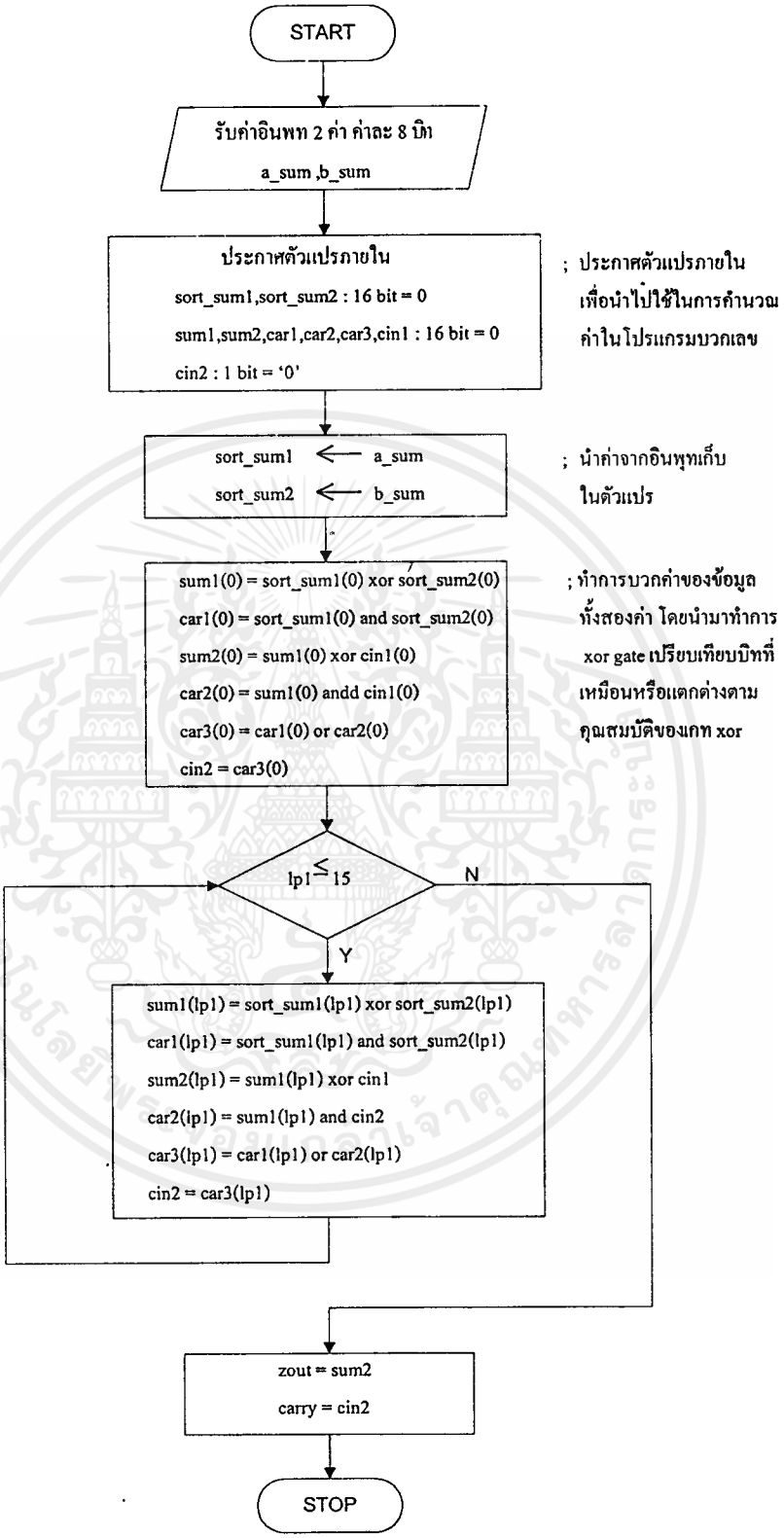


รูปที่ 3.20 ผังงานโปรแกรมคูณเลขขนาด 8 บิท

3.4.3 อัลกอริทึมโปรแกรมบวกเลขขนาด 8 บิท

การทำงานของโปรแกรมบวกเลขขนาด 8 บิท เริ่มต้นโดยการรับค่าอินพุทขนาด 8 บิท 2 ค่า a_sum และ b_sum นำค่าอินพุทที่ได้เก็บไว้ในตัวแปรขนาด 16 บิท โดยเก็บไว้ที่ 8 บิทล่างทั้ง 2 ค่า ส่วน 8 บิทบนจะเก็บค่า '0' เริ่มบวกเลขโดยใช้โปรแกรม Full adder โดยบวกเฉพาะบิท '0' ก่อนเพื่อจะหาค่า carry in (cin) ตัวทดบิทแรกโดยใช้ cin2 เป็นตัวแปร การบวกจะใช้เกท xor เพื่อเปรียบเทียบความเหมือนหรือแตกต่างกันตามคุณสมบัติของเกทที่มีอยู่ โดยบิทที่เหมือนกันเอาท์พุทได้ '0' บิทที่แตกต่างกันเอาท์พุทได้ '1'

หลังจากได้ค่าตัวทดบิทแรกแล้วทำการบวกเลขบิทที่เหลือโดยใช้การวนลูป เริ่มที่ 1 ถึง 15 เพราะบิทแรกเราได้บวกแล้ว เมื่อทำการบวกเสร็จแล้วนำค่าออกแสดงผล ค่าบวกสุดท้ายที่อยู่ที่ตัวแปร sum2 และตัวทดหากค่าบิทสุดท้ายมีค่าเท่ากับ '1' ทั้ง 2 ตัวตรวจสอบได้ที่ตัวแปร cin2 หรือ ตัวแปร car3 บิทสุดท้าย



รูปที่ 3.21 ผังงานโปรแกรมบวกเลขขนาด 8 บิท

3.4.4 อัลกอริทึมโปรแกรมคำนวณแถบความถี่ขนาด 8 จุด

การทำงานของโปรแกรมจะมีส่วนที่ควบคุมการทำงานอยู่ 2 ส่วน คือ clk และ reset clk จะให้เปรียบสัญญาณนาฬิกาโดยใช้ขอบขาขึ้นเป็นหลัก ส่วน reset จะมีสภาวะที่ใช้ควบคุมอยู่ 2 ส่วน คือ ถ้า reset เป็น '0' จะเป็นส่วนของการรับค่าข้อมูลขนาด 8 บิท จำนวน 8 ค่า แล้วเก็บไว้ในตัวแปรขนาด 16 บิท ส่วนถ้า reset เป็น '1' จะเป็นส่วนของการคำนวณ ขั้นแรกจะทำการสลับตำแหน่งคู่คี่ตามทฤษฎี FFT ที่แบ่งส่วนของข้อมูลออกเป็น 2 ส่วน เมื่อได้ตำแหน่งของข้อมูลที่ต้องการแล้ว ทำการคำนวณค่าต่าง ๆ โดยจับคู่ข้อมูลเป็น 4 คู่ โดยคำนวณตามสูตร

Pm = [Pr+Qr.cos(k)+Qi.sin(k)]+j[Pi+Qi.cos(k)-Qi.sin(k)]

Qm = [Pr-Qr.cos(k)-Qi.sin(k)]+j[Pi-Qi.cos(k)+Qi.sin(k)]

ค่า Pr และ Pi จะเป็นข้อมูลตัวแรก โดย Pr เป็น 8 บิทล่าง และ Pi เป็น 8 บิทบน

ค่า Qr และ Qi จะเป็นข้อมูลตัวที่ 2 โดย Qr เป็น 8 บิทล่าง และ Qi เป็น 8 บิทบน

ค่า cos และ sin จะทำการสเกลค่าไว้แล้วล่วงหน้า โดยการกำหนดค่าเริ่มต้น 0-255 ระดับใน 1 ลูกคลื่นแบ่ง 8 ส่วนได้ส่วนละ x = 45°

หาค่า cos ได้โดย cos(x)x255 ทั้งหมด 8 ค่า

หาค่า sin ได้โดย sin(x)x255 ทั้งหมด 8 ค่า

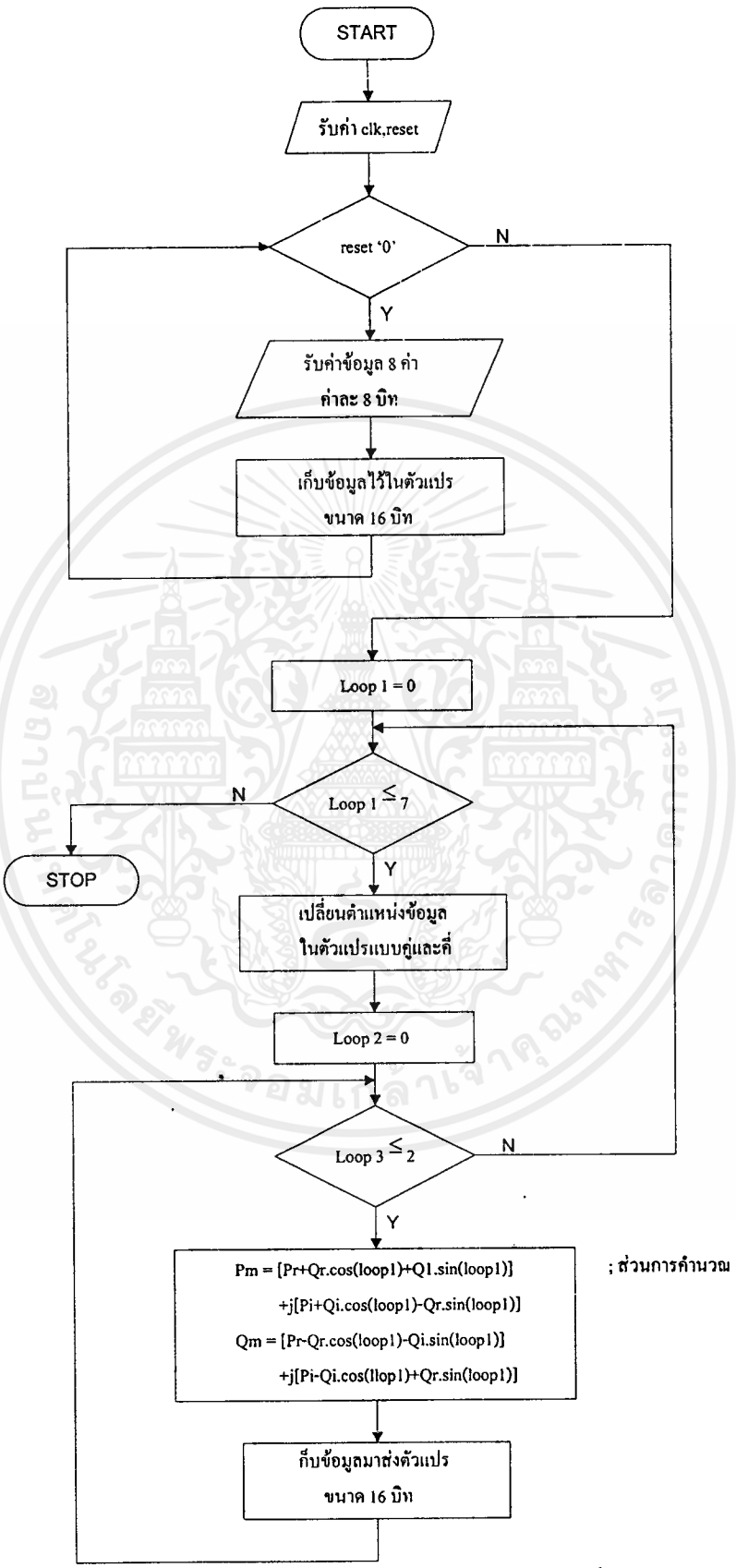
ส่วนการบวก ลบ และคูณ จะใช้โปรแกรมที่เขียนไว้แล้วมาประยุกต์ โดยทุกโปรแกรมจะมีอินพุทและเอาท์พุทเป็นของตัวเอง

ปัญหาที่พบคือ ค่าการลบของโปรแกรมภายในส่วนคำนวณจะมีปัญหา แบ่งออกเป็น 2 ส่วน คือ

ถ้า 1.ตัวตั้ง – ตัวลบ = ผลลัพธ์เป็นบวก

2.ตัวตั้ง – ตัวลบ = ผลลัพธ์ติดลบ

การแก้ปัญหาโดยแบ่งส่วนของการลบเป็น 2 ส่วนแต่ใช้ที่เก็บผลลัพธ์ตัวเดียวกัน



รูปที่ 3.22 ผังงานโปรแกรมคำนวณแถบความถี่ขนาด 8 จุด

3.5 การแปลงฟัซท ฟูเรียร์ ทรานฟอร์ม (Fast Fourier Transform หรือ FFT)

การประมวลผลสัญญาณเชิงเลข ได้มีการพัฒนาเพื่อใช้งานหลายด้านเช่น การประมวลสัญญาณเสียง การประมวลสัญญาณภาพเป็นต้น การแปลงฟูริเยอร์เต็มหน่วย (DFT) นั้นเราสามารถนำมาใช้ในการคำนวณประสานได้ แต่การคำนวณ DFT นั้นเมื่อจำนวนลำดับข้อมูลมีมากก็ใช้เวลาในการคำนวณมาก เช่นข้อมูลเข้ามายาว 8 คำ จะต้องคำนวณจำนวนเชิงซ้อน $8 \times 8 = 64$ ครั้ง และจะต้องนำมาบวกกับจำนวนเชิงซ้อนอีก $8 \times (8-1) = 56$ ครั้ง

จากปัญหาที่พบได้มีการพัฒนาการคำนวณ DFT ให้เร็วขึ้นเรียกว่า การแปลงฟัซทฟูริเยอร์ (FFT) ได้มีการพัฒนาจากผลงานของคูลีย์ (J.W. Cooley) กับทูคีย์ (J.W. Tuke) ในปี ค.ศ.1965 จากวิธีการ FFT จะลดการคำนวณจำนวนเชิงซ้อนลง เช่น ข้อมูล 8 ค่า คำนวณเชิงซ้อนจะได้ $8 \log_2 8 = 24$ ครั้ง

หลักการเบื้องต้นของ FFT เป็นการจัดแบ่งกลุ่มลำดับสัญญาณในโดเมนเวลา X(m) ที่มีขนาด N จุด ออกเป็นสองลำดับสัญญาณที่มีความยาว N/2 จุดเท่ากัน เรียกว่าลำดับสัญญาณคู่และลำดับสัญญาณคี่ ดังนี้

ให้เลขลำดับคู่ คือ $X_E(m) = X_{(2m)}$; $m = 0,1\ldots,(N/2)-1$ (3.2)

เลขลำดับคี่ คือ $X_O(m) = X_{(2m-1)}$; $m = 0,1\ldots,(N/2)-1$ (3.3)

ด้วยการแบ่งเช่นนี้ ถ้าให้ ω_N แทนค่า ω ของลำดับยาว N จุด ทำให้การคำนวณการแปลง DFT ของลำดับสัญญาณ X(m) ที่ยาว N จุด เขียนได้ดังนี้

$$X(x) = \sum_{m=0}^{(N/2)-1} X(2m)(\omega_N)^{2mk} + \sum_{m=0}^{(N/2)-1} X(2m+1)(\omega_N)^{(2m+1)k} \qquad (3.4)$$

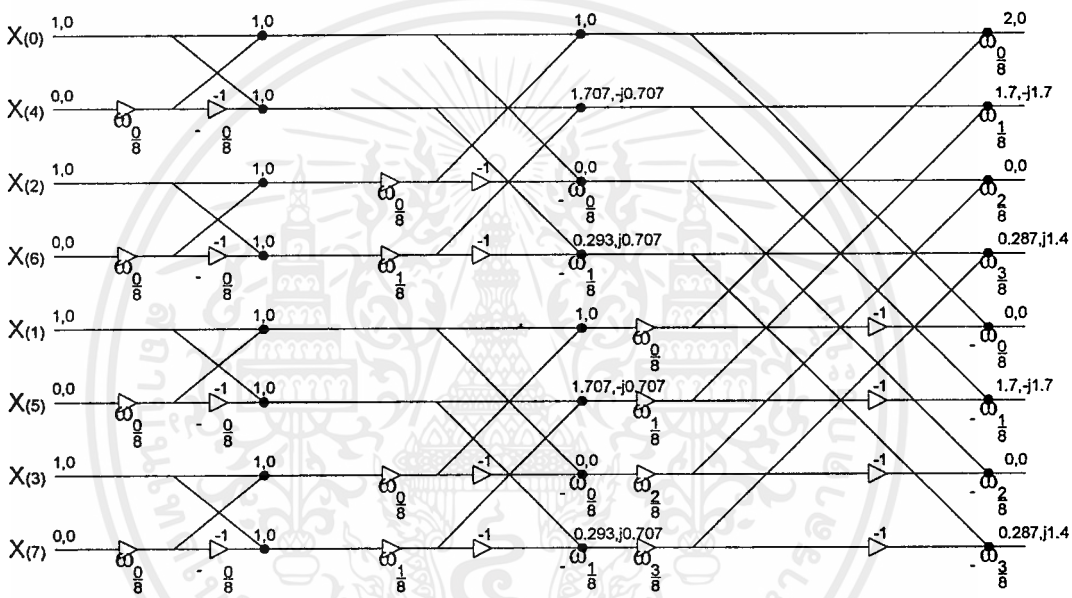
โดยให้ $(\omega_N)^2$ เป็น

$$(\omega_N)^2 = (e^{j2\pi/N})^2 = e^{j4\pi/N} = \omega_{N/2} \qquad (3.5)$$

ซึ่ง $\omega_{N/2}$ หรือค่า ω ของลำดับยาว N/2 จุด จึงจัดสมการที่ 3.4 ได้ใหม่คือ

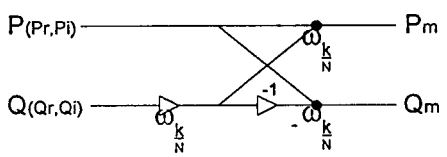
$$X(k) = \sum_{m=0}^{(N/2)-1} X(2m)(\omega_{N/2}^{mk}) + \omega_{k/2} \sum_{m=0}^{(N/2)-1} X(2m+1)\omega_{N/2}^{mk} \qquad (3.6)$$

ตัวอย่าง การคำนวณ FFT ขนาด 8 จุดโดยกำหนดให้ X(n) = {1,1,1,1,0,0,0,0}



รูปที่ 3.23 Flowgraph ของ FFT ขนาด 8 จุด

การคำนวณสามารถแบ่งออกเป็นการคำนวณทีละคู่ได้โดยใช้สมการที่ 3.7 และสมการที่ 3.8

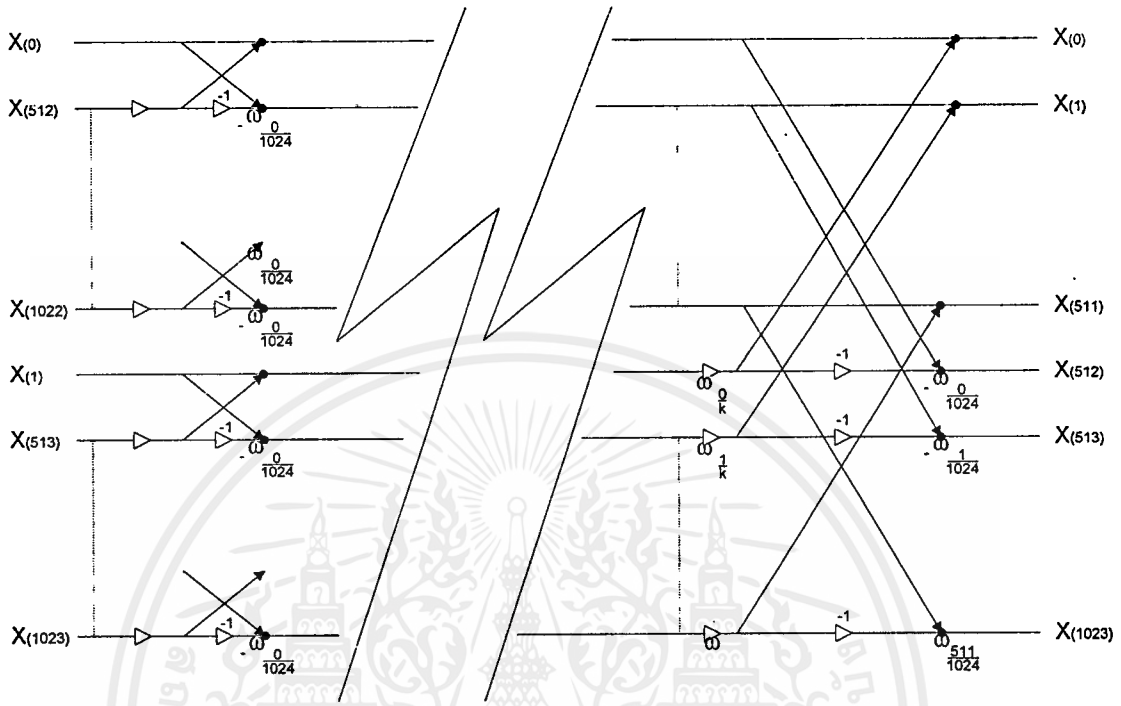


รูปที่ 3.24 Flowgraph การคำนวณทีละคู่

$$Pm = [Pr + Qr\cos(k) + Qi\sin(k)] + j\,[Pi + Qi\cos(k) - Qr\sin(k)] \qquad (3.7)$$

$$Qm = [Pr - Qr\cos(k) - Qi\sin(k)] + j\,[Pi - Qi\cos(k) + Qr\sin(k)] \qquad (3.8)$$

การคำนวณ 1024 จุด



รูปที่ 3.25 Flowgraph ของ FFT ขนาด 1024 จุดอย่างย่อ

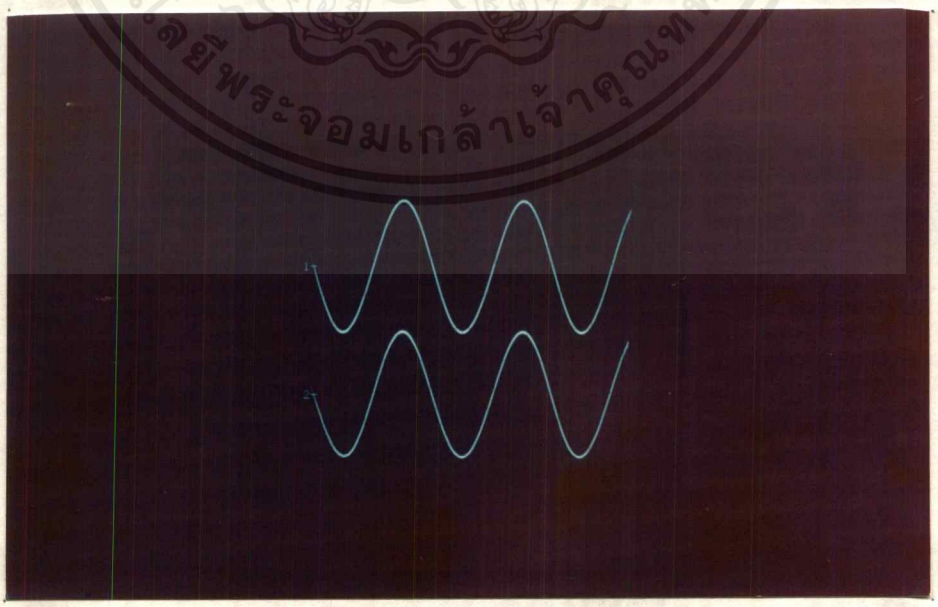
บทที่ 4

การทดลอง และผลการทดลอง

เครื่องวิเคราะห์แถบความถี่ที่ออกแบบและสร้างขึ้นเพื่อใช้งานในย่านความถี่ต่ำ ย่าน 0 ถึง 1 MH_z แสดงผลในลักษณะแถบความถี่บนจอโทรทัศน์ ในบทนี้เป็นการทดลองและทดสอบการทำงานของเครื่องวิเคราะห์แถบความถี่ โดยแบ่งการทดสอบเป็นส่วน ๆ ดังนี้

4.1 วงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล (A/D)

เครื่องวิเคราะห์แถบความถี่จะใช้ ไอซีเบอร์ CA3318 ซึ่งเป็นไอซีที่ใช้ในการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล ซึ่งจากการทดลองสัญญาณเอาท์พุทที่ได้จากวงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลจะมีขนาด 8 บิทการทดสอบของสัญญาณเอาท์พุทที่ได้โดยการนำไปเข้าวงจร R2R ladder ซึ่งเป็นวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอกซึ่งเอาท์พุทของวงจร R2R ladder จะมีลักษณะใกล้เคียงกับสัญญาณอินพุทที่เข้าวงจร (A/D)



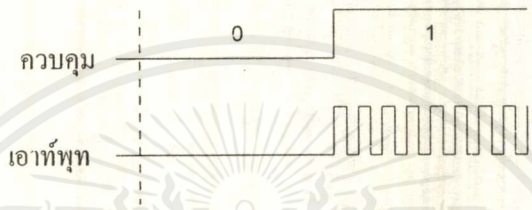
รูปที่ 4.1 สัญญาณ A/D และสัญญาณ D/A ที่ได้จากการทดสอบ

4.2 วงจรการสร้างภาพบนจอโทรทัศน์

แบ่งการทดสอบออกเป็นส่วน ๆ ดังนี้

4.2.1 วงจรกำเนิดสัญญาณนาฬิกา

จากการทดสอบเมื่อเราป้อนสัญญาณควบคุมให้แก่วงจร สัญญาณเอาท์พุทที่ได้จะเป็นดังรูปที่ 4.2



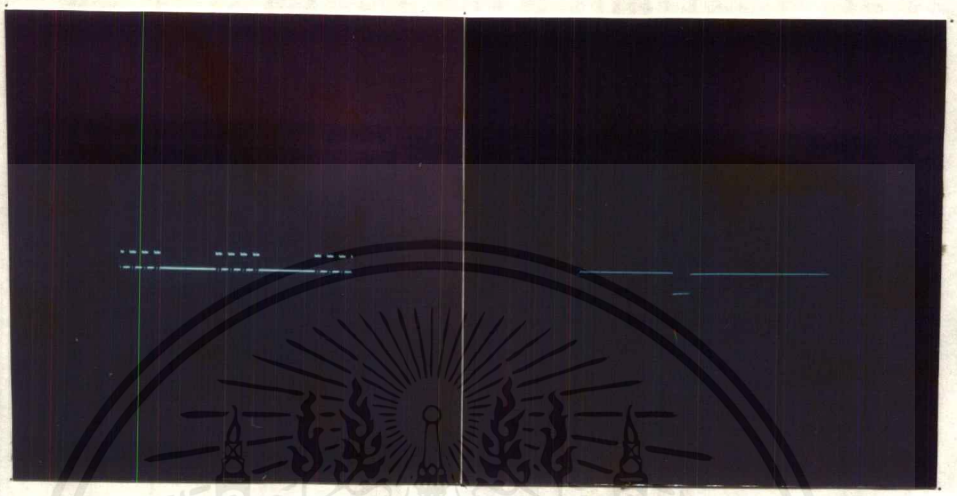
รูปที่ 4.2 สัญญาณควบคุมและสัญญาณเอาท์พุทที่ได้จากการทดสอบ

4.2.2 วงจรหาร 8

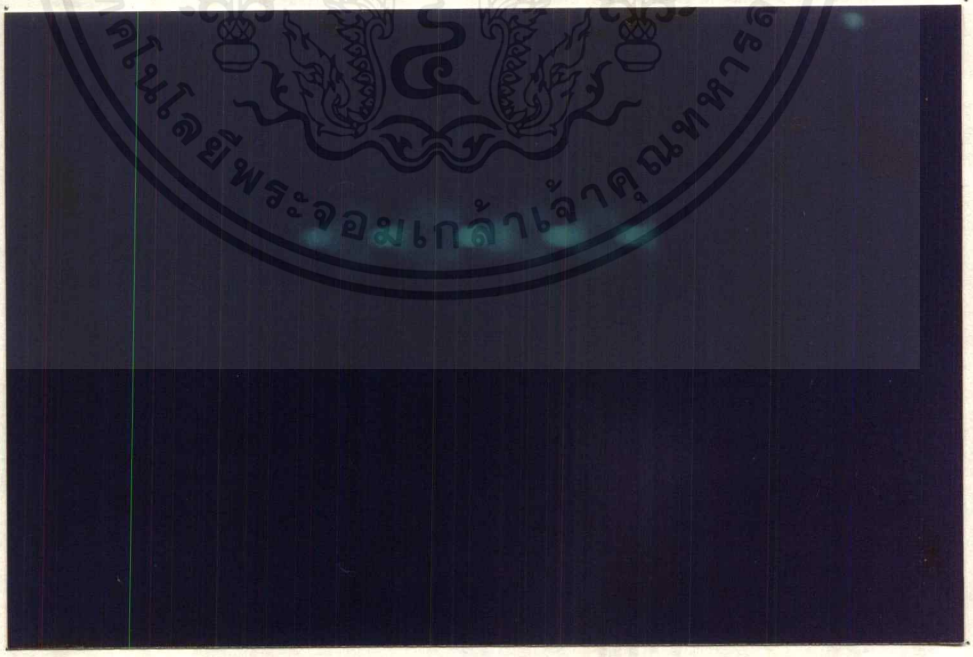
การทดสอบวงจรหาร 8 เราจะได้เอาท์พุทออกมา 3 สัญญาณที่ต้องการคือ สัญญาณ แลตซ์ วัดได้ที่ขา QC สัญญาณS/L วัดได้ที่ขา LD และสัญญาณCCLK วัดได้ที่ขา QD ตามลำดับ

4.2.3 วงจรฮอร์โมโนสเตเบิ้ล และวงจรเวอร์โมโนสเตเบิ้ล

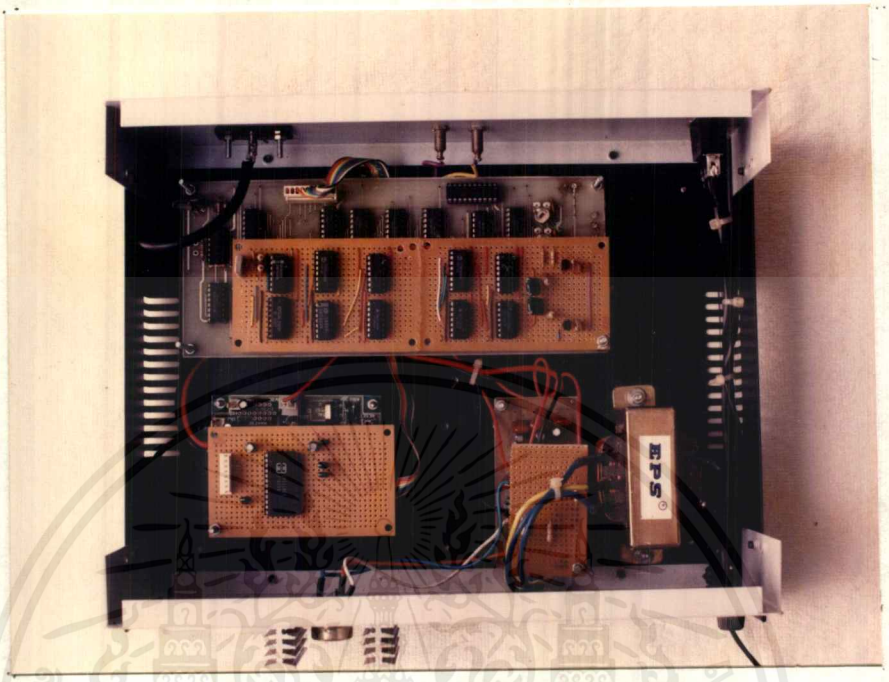
จากการทดสอบวงจรฮอร์โมโนสเตเบิ้ลโดยการป้อนฮอร์พัลซ์เข้าที่ขา 5 ของไอซี 4528 และวัดสัญญาณฮอร์โมโนสเตเบิ้ลที่ขา Q1 จะมีคาบเวลา 20 ไมโครวินาที ส่วนวงจรเวอร์โมโนสเตเบิ้ลนั้นจะใช้ไอซีเบอร์ 4528 เช่นเดียวกัน เมื่อเราป้อนสัญญาณเวอร์พัลซ์ เข้าที่ขา B2 และวัดสัญญาณเวอร์โมโนสเตเบิ้ลที่ขา Q2 จะมีคาบเวลาประมาณ 5 มิลลิวินาที



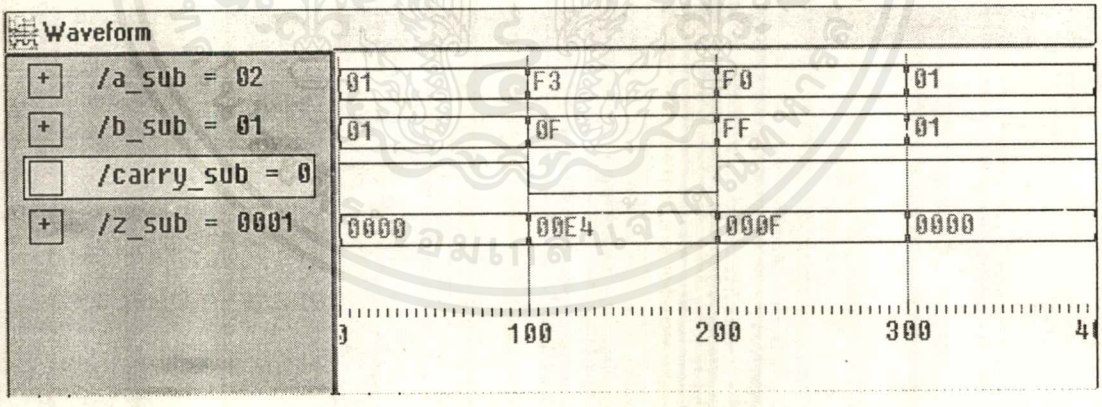
รูปที่ 4.3 สัญญาณฮอร์พัลซ์และสัญญาณเวอร์พัลซ์ที่ได้จากการทดสอบ



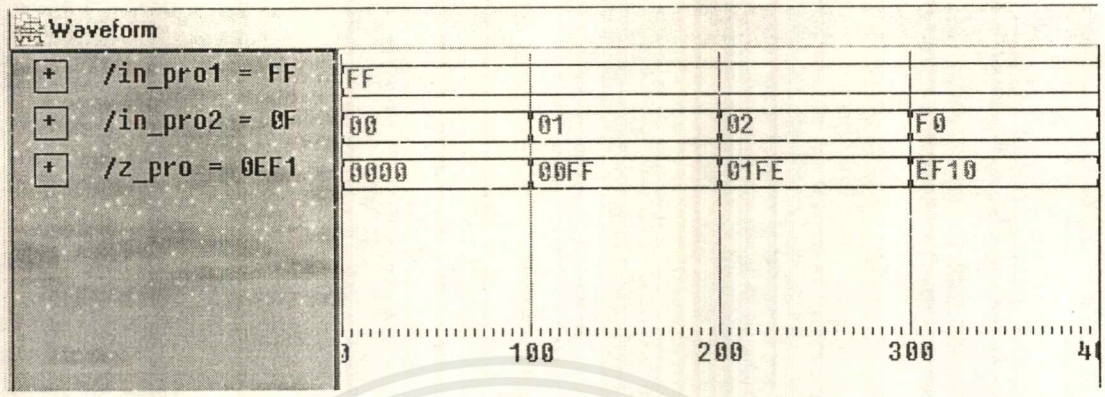
รูปที่ 4.4 สัญญาณภาพที่ได้จากการทดสอบ



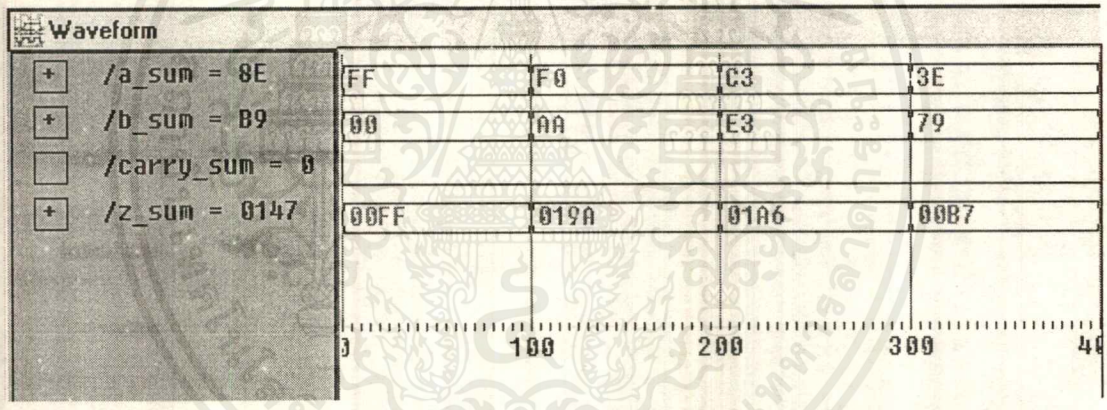
รูปที่ 4.5 ภาพวงจรรวม



รูปที่ 4.6 ผลการทดลองวงจรลบเลขขนาด 8 บิท



รูปที่ 4.7 ผลการทดลองวงจรคูณเลขขนาด 8 บิท



รูปที่ 4.8 ผลการทดลองวงจรบวกเลขขนาด 8 บิท

บทที่ 5
สรุปและวิจารณ์

5.1 สรุป

เครื่องวิเคราะห์แถบความถี่ย่าน 0-1 MHz โดยใช้ภาษา VHDL และอุปกรณ์ FPGAs สร้างขึ้นเพื่อศึกษาการประยุกต์ใช้งานด้านการประมวลผล วิเคราะห์แถบความถี่ โดยระบบประมวลผลสัญญาณเชิงเลข โดยขอบเขตที่วางไว้ข้างต้นคือ สามารถวัดความถี่ในย่าน 0-1 MHz และแสดงผลออกทางโทรทัศน์

จากการศึกษาและทดลองสร้าง ปรากฏว่าผลที่ได้อยู่ในระดับที่น่าพอใจ กล่าวคือเครื่องวิเคราะห์แถบความถี่นี้ สามารถวัดได้ในระดับหนึ่ง

ข้อดีของเครื่องวิเคราะห์แถบความถี่นี้คือ มีราคาถูกกว่าเครื่องที่มีขายตามท้องตลาด

5.2 ปัญหาที่พบ

5.2.1 ในส่วนของฮาร์ดแวร์

ปัญหา	อุปกรณ์มีราคาแพง และตรงกับช่วงปิดงบประมาณ
แนวทางแก้ไข	ทางภาควิชาควรให้การช่วยเหลือเรื่องงบประมาณในการทำปริญญานิพนธ์มากกว่านี้
ปัญหา	วงจรแสดงผลออกทางจอโทรทัศน์ที่หามาได้มีความเที่ยงตรงไม่เพียงพอ
แนวทางแก้ไข	ค้นคว้าหาวงจรที่มีประสิทธิภาพมากกว่านี้
ปัญหา	อุปกรณ์บางอย่างหาซื้อยาก ทางร้านไม่สั่งนำมาขาย เพราะใช้กันในงานเฉพาะอย่าง
แนวทางแก้ไข	ต้องให้ทางร้านค้าสั่งให้จากที่อื่น ทำให้ราคาแพงยิ่งขึ้น
ปัญหา	อุปกรณ์ที่หาซื้อมาได้มีคุณภาพต่ำ เมื่อนำมาใช้งานเกี่ยวกับความถี่จะพบปัญหามาก

แนวทางแก้ไข	หาซื้ออุปกรณ์ที่มีคุณภาพจากร้านที่น่าเชื่อถือ แต่ราคาจะแพงกว่ากันมาก

5.2.2 ในส่วนของซอฟท์แวร์

ปัญหา	เนื่องจากมีความรู้เรื่องภาษา VHDL และการใช้งานโปรแกรมที่ใช้โปรแกรมอุปกรณ์ FPGAs ไม่เพียงพอจึงทำให้เกิดความหล้าช้าในการเขียนโปรแกรม
แนวทางแก้ไข	ให้อาจารย์สอนและทำการทดลองเขียนโปรแกรมบ่อย ๆ

5.3 แนวทางในการพัฒนา

1. ทำการปรับปรุง และแก้ไขการเขียนโปรแกรมที่ใช้อัลกอริทึมของ FFT ให้ลงสู่ระดับเกทมากยิ่งขึ้น ซึ่งจะทำให้การสร้างเป็นวงจรภายในอุปกรณ์ FPGAs มีขนาดเล็กลงและทำงานได้ดียิ่งขึ้น

2. ส่วนของวงจรควบคุมการแสดงผลที่หน้าจอภาพของโทรทัศน์ ควรออกแบบให้มีความละเอียดมากยิ่งขึ้น

3. ควรเพิ่มขนาดของวิดีโอแรมเพื่อที่จะพัฒนาให้สามารถเก็บหน้าจอได้หลาย ๆ หน้าจอ

ภาคผนวก ก
โปรแกรม

โปรแกรมลบ

```
-----------------------------------------------
-------program  subtracter data 8 bit adv--------
----------by  ganda  wanchai----------
-----------------------------------------------

library ieee;
use ieee.std_logic_1164.all ;

entity suda69 is
        port(
           a_sub:in  std_logic_vector(7 downto 0);
           b_sub:in  std_logic_vector(7 downto 0);
           carry_sub :out std_logic;
--         z_sub1    :out std_logic_vector(7 downto 0);
           z_sub     :out std_logic_vector(15 downto 0)
          );
end suda69;

architecture ganda69 of suda69 is
begin
s1:process(a_sub,b_sub)
        variable sort_sub1,sort_sub2 : std_logic_vector(15 downto 0):="0000000000000000";
        variable sum1,sum2,sum3      : std_logic_vector(15 downto 0);
        variable car1,car2,car3      : std_logic_vector(15 downto 0);
        variable cin1                : std_logic_vector(15 downto 0):="0000000000000000";
        variable cin2                : std_logic;

        begin
        sort_sub1:="0000000000000000";
        sort_sub2:="0000000000000000";
        cin1    :="0000000000000000";
        cin2:='0';
        sort_sub1(7 downto 0) := a_sub(7 downto 0);
```

```
sort_sub2(7 downto 0) := b_sub(7 downto 0);
sum1(0) := sort_sub1(0) xor not sort_sub2(0);
car1(0) := sort_sub1(0) and not sort_sub2(0);
sum2(0) := sum1(0) xor cin1(0);
car2(0) := sum1(0) and cin1(0);
car3(0) := car1(0) or car2(0);
cin2    := car3(0);

for lp1 in 1 to 15 loop
        sum1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
        car1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
        sum2(lp1) := sum1(lp1) xor cin2;
        car2(lp1) := sum1(lp1) and cin2;
        car3(lp1) := car1(lp1) or car2(lp1);
        cin2      := car3(lp1);
end loop;
sort_sub1:=sum2;
sort_sub2(0):=cin2;
for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
cin2:='0';

for lp3 in 0 to 15 loop
        sum1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
        car1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
        sum2(lp3) := sum1(lp3) xor cin2;
        car2(lp3) := sum1(lp3) and cin2;
        car3(lp3) := car1(lp3) or car2(lp3);
        cin2      := car3(lp3);
end loop;

for lp2 in 0 to 15 loop
        sum3(lp2) := sum2(lp2) xor not sort_sub2(0);
end loop;
```

```
--                  z_sub1(7 downto 0)  <= sum3(7 downto 0);
                    z_sub  <= sum3;
                    carry_sub <=not sort_sub2(0);

          end process s1;
end ganda69;
```

โปรแกรมบวก

```
-------------------------------------------
----------program  adder data 8 bit ------------
------------by  ganda  wanchai-----------------
-------------------------------------------

library ieee;
use ieee.std_logic_1164.all ;

entity suda71 is
          port(
            a_sum    :in  std_logic_vector(7 downto 0);
            b_sum    :in  std_logic_vector(7 downto 0);
--          carry_sum :out std_logic;
            z_sum    :out std_logic_vector(15 downto 0)
--          z_sum2   :out std_logic_vector(7 downto 0)
            );
end suda71;

architecture ganda71 of suda71 is
begin
s1:process(a_sum,b_sum)
          variable sort_sum1,sort_sum2 : std_logic_vector(15 downto 0):="0000000000000000";
          variable sum1,sum2        : std_logic_vector(15 downto 0);
          variable car1,car2,car3   : std_logic_vector(15 downto 0);
          variable cin1             : std_logic_vector(15 downto 0):="0000000000000000";
          variable cin2             : std_logic;
```

```
        begin
        sort_sum1(7 downto 0) := a_sum(7 downto 0);
        sort_sum2(7 downto 0) := b_sum(7 downto 0);
        sum1(0) := sort_sum1(0) xor sort_sum2(0);
        car1(0) := sort_sum1(0) and sort_sum2(0);
        sum2(0) := sum1(0) xor cin1(0);
        car2(0) := sum1(0) and cin1(0);
        car3(0) := car1(0) or car2(0);
        cin2    := car3(0);

        for lp1 in 1 to 15 loop
                sum1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
                car1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
                sum2(lp1) := sum1(lp1) xor cin2;
                car2(lp1) := sum1(lp1) and cin2;
                car3(lp1) := car1(lp1) or car2(lp1);
                cin2      := car3(lp1);
        end loop;

                z_sum  <= sum2;
--              z_sum2(7 downto 0)  <= sum2(15 downto 8);
--              carry_sum <= cin2;

        end process s1;
end ganda71;
```

โปรแกรมคูณ

```
-------------------------------------------------
--------program  multiplied data 8 bit ----------
---------- by   ganda wanchai ------------------
-------------------------------------------------

library ieee;
use ieee.std_logic_1164.all ;
```

```
entity suda72 is
        port(
           in_pro1  :in  std_logic_vector(7 downto 0);
           in_pro2  :in  std_logic_vector(7 downto 0);
           z_pro    :out std_logic_vector(15 downto 0)
          );
end suda72;

architecture ganda72 of suda72 is
begin
s1:process(in_pro1,in_pro2)
        variable get1   : std_logic_vector(15 downto 0);
        variable get2   : std_logic_vector(15 downto 0);
        variable pro1   : std_logic_vector(15 downto 0);
        variable pro2   : std_logic_vector(15 downto 0);
        variable pro3   : std_logic_vector(15 downto 0);
        variable cin    : std_logic:='0';
        variable count1 : std_logic;
        variable count2 : std_logic;
        variable lp,i   : integer;

        begin
        for lpf in 0 to 15 loop
                pro1(lpf) :='0';
                pro2(lpf) :='0';
                pro3(lpf) :='0';
                get1(lpf) :='0';
                get2(lpf) :='0';
        end loop;

        get1(7 downto 0) := in_pro1(7 downto 0);
        get2(7 downto 0) := in_pro2(7 downto 0);

        lp:=0;
```

```
        while(lp<8)  loop
                if(get2(lp)='1') then
                        for lf in 0 to 15 loop pro1(lf):='0' ; end loop;
                        i:=0;
                        while(i<8)loop
                                pro1(lp+i):=get1(i);
                                i:=i+1;
                        end loop;

                        i:=0;
                        cin:='0';

                        while(i<16)loop
                                pro2(i)  := pro1(i) xor pro3(i);
                                count1   := pro1(i) and pro3(i);
                                pro3(i)  := pro2(i) xor cin;
                                count2   := pro2(i) and cin;
                                cin      := count1 or count2;
                                i:=i+1;
                        end loop;
                end if;
                lp:=lp+1;
        end loop;
        z_pro<=pro3;

        end process s1;
end ganda72;
```

โปรแกรม SPECTRUM 8 POINTS BY FFT ALGORITHM

```
----------------------------------------------------------------
--------program compute spectrum  8 points  by fft algorithm---------
----------------------  by ganda wanchai  -------------------------
----------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

entity suda91 is
        port (
           clk     :in std_logic;
           reset   :in std_logic;
           out_real :out std_logic_vector(63 downto 0);
           out_imag :out std_logic_vector(63 downto 0)
           );
end suda91;


architecture ganda91 of suda91 is
begin
g1: process(clk,reset)

          variable var_in_real : std_logic_vector(63 downto 0);
          variable var_ch_real : std_logic_vector(255 downto 0);
          variable var_real : std_logic_vector(255 downto 0);
--        variable var_imag : std_logic_vector(127 downto 0);
          variable var_cos  : std_logic_vector(127 downto 0);
          variable var_sin  : std_logic_vector(127 downto 0);
-- variable of program subtractor 8 bits
          variable sort_sub1,sort_sub2 : std_logic_vector(15 downto 0):="0000000000000000";
          variable sums1,sums2,sums3   : std_logic_vector(15 downto 0);
          variable cars1,cars2,cars3   : std_logic_vector(15 downto 0);
          variable cins1               : std_logic_vector(15 downto 0):="0000000000000000";
```

```
        variable cins2          : std_logic;
        variable carry_sub_pr   : std_logic;
        variable carry_sub_pi   : std_logic;
        variable carry_sub_qr   : std_logic;
        variable carry_sub_qi   : std_logic;

-- variable of program adder 8 bits
        variable sort_sum1,sort_sum2 : std_logic_vector(15 downto 0):="0000000000000000";
        variable sumd1,sumd2         : std_logic_vector(15 downto 0);
        variable card1,card2,card3   : std_logic_vector(15 downto 0);
        variable cind1               : std_logic_vector(15 downto 0):="0000000000000000";
        variable cind2               : std_logic;

--variable of multiple 8 bits
        variable get1   : std_logic_vector(15 downto 0);
        variable get2   : std_logic_vector(15 downto 0);
        variable pro1   : std_logic_vector(15 downto 0);
        variable pro2   : std_logic_vector(15 downto 0);
        variable pro3   : std_logic_vector(15 downto 0);
        variable cin    : std_logic:='0';
        variable count1 : std_logic;
        variable count2 : std_logic;
        variable lp,i   : integer;

--variable of subtractor  adder and multiple
        variable var_pro1_pr  : std_logic_vector(15 downto 0);
        variable var_qr_c_pm  : std_logic_vector(15 downto 0);
        variable var_qi_s_pm  : std_logic_vector(15 downto 0);
        variable var_sum_pr_pm  : std_logic_vector(15 downto 0);
        variable var_pro1_pi  : std_logic_vector(15 downto 0);
        variable var_qi_c_pm  : std_logic_vector(15 downto 0);
        variable var_qr_s_pm  : std_logic_vector(15 downto 0);
        variable var_sum_pi_pm  : std_logic_vector(15 downto 0);
        variable var_pro2_qr  : std_logic_vector(15 downto 0);
```

```
        variable var_sum_qr_qm  : std_logic_vector(15 downto 0);
        variable var_pro2_qi  : std_logic_vector(15 downto 0);
        variable var_sum_qi_qm  : std_logic_vector(15 downto 0);

--start to process data with algorithm fft 8 points
        begin
var_cos(63 downto
0):="1011010001111111010010100000000001001010011111110110100111111111";
var_sin(63 downto
0):="0100101000000000010010100111111111011010011111111101101000111111";
var_in_real(63 downto
0):="1111111111111110111111001111100000000000000000000000000000000000";

if(clk'event and clk = '1')then
        if(reset = '0')then
                for lp1 in 0 to 255 loop var_real(lp1):= '0';end loop;
                var_real(  7 downto  0):=var_in_real( 7 downto  0);
                var_real( 39 downto 32):=var_in_real(15 downto  8);
                var_real( 71 downto 64):=var_in_real(23 downto 16);
                var_real(103 downto 96):=var_in_real(31 downto 24);
                var_real(135 downto 128):=var_in_real(39 downto 32);
                var_real(167 downto 160):=var_in_real(47 downto 40);
                var_real(199 downto 192):=var_in_real(55 downto 48);
                var_real(231 downto 224):=var_in_real(63 downto 56);
        end if;

        if(reset = '1')then

        for lc in 0 to 7 loop
                for m2 in 0 to 3 loop
                        var_ch_real( 15 downto   0):=var_real( 15 downto   0);
                        var_ch_real( 79 downto  64):=var_real( 31 downto  16);
                        var_ch_real( 47 downto  32):=var_real( 47 downto  32);
```

```
                var_ch_real(111 downto  96):=var_real( 63 downto  48);
                var_ch_real( 31 downto  16):=var_real( 79 downto  64);
                var_ch_real( 95 downto  80):=var_real( 95 downto  80);
                var_ch_real( 63 downto  48):=var_real(111 downto  96);
                var_ch_real(127 downto 112):=var_real(127 downto 112);


-- compute pr+qr*cos(k)+qi*sin(k)
-- 1 compute qr*cos(k)
        for lpf11 in 0 to 15 loop
                pro1(lpf11) :='0';  pro2(lpf11) :='0';  pro3(lpf11) :='0';
                get1(lpf11) :='0';  get2(lpf11) :='0';
        end loop;
        for m11 in 0 to 7 loop
                get1(m11) := var_ch_real((m2*32)+16+m11);
                get2(m11) := var_cos((lc*8)+m11);
        end loop;--end loop m11
        lp:=0;
        while(lp<8)  loop
                if(get2(lp)='1') then
                        for lf in 0 to 15 loop pro1(lf):='0' ; end loop;
                        i:=0;
                        while(i<8)loop
                                pro1(lp+i):=get1(i);
                                i:=i+1;
                        end loop;

                        i:=0;
                        cin:='0';

                        while(i<16)loop
                                pro2(i)  := pro1(i) xor pro3(i);
                                count1   := pro1(i) and pro3(i);
                                pro3(i)  := pro2(i) xor cin;
```

```
                count2  := pro2(i) and cin;
                cin     := count1 or count2;
                i:=i+1;
            end loop;
        end if;
        lp:=lp+1;
    end loop;
    var_qr_c_pm:=pro3;

-- 2 compute qi*sin(k)
    for lpf12 in 0 to 15 loop
        pro1(lpf12) :='0';  pro2(lpf12) :='0';  pro3(lpf12) :='0';
        get1(lpf12) :='0';  get2(lpf12) :='0';
    end loop;
    for m12 in 0 to 7 loop
        get1(m12) := var_ch_real((m2*32)+24+m12);
        get2(m12) := var_sin((lc*8)+m12);
    end loop;--end loop m12
    lp:=0;
    while(lp<8)  loop
        if(get2(lp)='1') then
            for lf in 0 to 15 loop pro1(lf):='0' ; end loop;
            i:=0;
            while(i<8)loop
                pro1(lp+i):=get1(i);
                i:=i+1;
            end loop;

            i:=0;
            cin:='0';

            while(i<16)loop
                pro2(i)  := pro1(i) xor pro3(i);
                count1   := pro1(i) and pro3(i);
```

```
                                    pro3(i)  := pro2(i) xor cin;
                                    count2   := pro2(i) and cin;
                                    cin      := count1 or count2;
                                    i:=i+1;
                                end loop;
                        end if;
                        lp:=lp+1;
                end loop;
                var_qi_s_pm:=pro3;

--3 compute qr*cos(k)  +  qi*sin(k)

                sort_sum1 := var_qr_c_pm;--input from qr*cos(k)
                sort_sum2 := var_qi_s_pm;--input from qi*sin(k)

                sumd1(0) := sort_sum1(0) xor sort_sum2(0);
                card1(0) := sort_sum1(0) and sort_sum2(0);
                sumd2(0) := sumd1(0) xor cind1(0);
                card2(0) := sumd1(0) and cind1(0);
                card3(0) := card1(0) or card2(0);
                cind2    := card3(0);

                for lp1 in 1 to 15 loop
                        sumd1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
                        card1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
                        sumd2(lp1) := sumd1(lp1) xor cind2;
                        card2(lp1) := sumd1(lp1) and cind2;
                        card3(lp1) := card1(lp1) or card2(lp1);
                        cind2      := card3(lp1);
                end loop;

                        var_sum_pr_pm := sumd2;
```

```
--4 compute ( pr + (qr*cos(k)+qi*sin(k) )
        for m14 in 0 to 7 loop
                sort_sum1(m14) := var_ch_real((m2*32)+m14);--input from pr
        end loop;
                sort_sum2 := var_sum_pr_pm;--input from qi*sin(k)

        sums1(0) := sort_sum1(0) xor sort_sum2(0);
        cars1(0) := sort_sum1(0) and sort_sum2(0);
        sums2(0) := sums1(0) xor cins1(0);
        cars2(0) := sums1(0) and cins1(0);
        cars3(0) := cars1(0) or cars2(0);
        cins2    := cars3(0);

        for lp1 in 1 to 15 loop
            sums1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
            cars1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
            sums2(lp1) := sums1(lp1) xor cins2;
            cars2(lp1) := sums1(lp1) and cins2;
            cars3(lp1) := cars1(lp1) or cars2(lp1);
            cins2      := cars3(lp1);
        end loop;

                var_pro1_pr:= sums2;

-- end compute pr+qr*cos(k)+qi*sin(k)

-- compute pi+qi*cos(k)+qr*sin(k)
-- 1 compute qi*cos(k)
        for lpf11 in 0 to 15 loop
                pro1(lpf11) :='0';  pro2(lpf11) :='0';  pro3(lpf11) :='0';
                get1(lpf11) :='0';  get2(lpf11) :='0';
        end loop;
        for m11 in 0 to 7 loop
                get1(m11) := var_ch_real((m2*32)+24+m11);
```

```
                get2(m11) := var_cos((lc*8)+m11);
        end loop;--end loop m11
        lp:=0;
        while(lp<8)  loop
                if(get2(lp)='1') then
                        for lf in 0 to 15 loop pro1(lf):='0' ; end loop;
                        i:=0;
                        while(i<8)loop
                                pro1(lp+i):=get1(i);
                                i:=i+1;
                        end loop;

                        i:=0;
                        cin:='0';

                        while(i<16)loop
                                pro2(i)  := pro1(i) xor pro3(i);
                                count1   := pro1(i) and pro3(i);
                                pro3(i)  := pro2(i) xor cin;
                                count2   := pro2(i) and cin;
                                cin      := count1 or count2;
                                i:=i+1;
                        end loop;
                end if;
                lp:=lp+1;
        end loop;
        var_qi_c_pm:=pro3;


-- 2 compute qr*sin(k)
        for lpf11 in 0 to 15 loop
                pro1(lpf11) :='0';  pro2(lpf11) :='0';  pro3(lpf11) :='0';
                get1(lpf11) :='0';  get2(lpf11) :='0';
        end loop;
        for m12 in 0 to 7 loop
```

```
                get1(m12) := var_ch_real((m2*32)+16+m12);
                get2(m12) := var_sin((lc*8)+m12);
        end loop;--end loop m12
        lp:=0;
        while(lp<8)  loop
                if(get2(lp)='1') then
                        for lf in 0 to 15 loop pro1(lf):='0' ; end loop;
                        i:=0;
                        while(i<8)loop
                                pro1(lp+i):=get1(i);
                                i:=i+1;
                        end loop;

                        i:=0;
                        cin:='0';

                        while(i<16)loop
                                pro2(i)  := pro1(i) xor pro3(i);
                                count1   := pro1(i) and pro3(i);
                                pro3(i)  := pro2(i) xor cin;
                                count2   := pro2(i) and cin;
                                cin      := count1 or count2;
                                i:=i+1;
                        end loop;
                end if;
                lp:=lp+1;
        end loop;
        var_qr_s_pm:=pro3;

--3 compute qi*cos(k)  -  qr*sin(k)

        sort_sub1:="0000000000000000";
        sort_sub2:="0000000000000000";
        cins1    :="0000000000000000";
```

```
cins2:='0';
sort_sub1 := var_qi_c_pm;--input from qi*cos(k)
sort_sub2 := var_qr_s_pm;--input from qr*sin(k)

sums1(0) := sort_sub1(0) xor not sort_sub2(0);
cars1(0) := sort_sub1(0) and not sort_sub2(0);
sums2(0) := sums1(0) xor cins1(0);
cars2(0) := sums1(0) and cins1(0);
cars3(0) := cars1(0) or cars2(0);
cins2    := cars3(0);

for lp1 in 1 to 15 loop
        sums1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
        cars1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
        sums2(lp1) := sums1(lp1) xor cins2;
        cars2(lp1) := sums1(lp1) and cins2;
        cars3(lp1) := cars1(lp1) or cars2(lp1);
        cins2      := cars3(lp1);
end loop;
sort_sub1:=sums2;
sort_sub2(0):=cins2;
for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
cins2:='0';

for lp3 in 0 to 15 loop
        sums1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
        cars1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
        sums2(lp3) := sums1(lp3) xor cins2;
        cars2(lp3) := sums1(lp3) and cins2;
        cars3(lp3) := cars1(lp3) or cars2(lp3);
        cins2      := cars3(lp3);
end loop;

for lp2 in 0 to 15 loop
```

```
                sums3(lp2) := sums2(lp2) xor not sort_sub2(0);
        end loop;

                var_sum_pi_pm := sums3;
                carry_sub_pr :=not sort_sub2(0);


--4 compute ( pi + (qi*cos(k)-qr*sin(k) )
--case 1 qi*cos(k)  more than qr*sin(k
        if(carry_sub_pr='0')then
                for m14 in 0 to 7 loop
                        sort_sum1(m14) := var_ch_real((m2*32)+8+m14);--input from pi
                end loop;
                        sort_sum2 := var_sum_pi_pm;--input from qi*sin(k)

                sumd1(0) := sort_sum1(0) xor sort_sum2(0);
                card1(0) := sort_sum1(0) and sort_sum2(0);
                sumd2(0) := sumd1(0) xor cind1(0);
                card2(0) := sumd1(0) and cind1(0);
                card3(0) := card1(0) or card2(0);
                cind2    := card3(0);
                for lp1 in 1 to 15 loop
                        sumd1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
                        card1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
                        sumd2(lp1) := sumd1(lp1) xor cind2;
                        card2(lp1) := sumd1(lp1) and cind2;
                        card3(lp1) := card1(lp1) or card2(lp1);
                        cind2      := card3(lp1);
                end loop;
                var_pro1_pi := sumd2;
 --case 2 qr*cos(k)  less than qi*sin(k
            elsif(carry_sub_pr='1')then
                        sort_sub1:="0000000000000000";
                        sort_sub2:="0000000000000000";
```

```
cins1    :="0000000000000000";
cins2:='0';
for m14 in 0 to 7 loop
        sort_sub1(m14) := var_ch_real((m2*32)+8+m14);--input from pi
end loop;
        sort_sub2 := var_sum_pi_pm;--input from qi*sin(k)

sums1(0) := sort_sub1(0) xor not sort_sub2(0);
cars1(0) := sort_sub1(0) and not sort_sub2(0);
sums2(0) := sums1(0) xor cins1(0);
cars2(0) := sums1(0) and cins1(0);
cars3(0) := cars1(0) or cars2(0);
cins2    := cars3(0);

for lp1 in 1 to 15 loop
        sums1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
        cars1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
        sums2(lp1) := sums1(lp1) xor cins2;
        cars2(lp1) := sums1(lp1) and cins2;
        cars3(lp1) := cars1(lp1) or cars2(lp1);
        cins2      := cars3(lp1);
end loop;
sort_sub1:=sums2;
sort_sub2(0):=cins2;
for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
cins2:='0';

for lp3 in 0 to 15 loop
        sums1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
        cars1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
        sums2(lp3) := sums1(lp3) xor cins2;
        cars2(lp3) := sums1(lp3) and cins2;
        cars3(lp3) := cars1(lp3) or cars2(lp3);
        cins2      := cars3(lp3);
```

```
                end loop;

                for lp2 in 0 to 15 loop
                        sums3(lp2) := sums2(lp2) xor not sort_sub2(0);
                end loop;

                var_pro1_pi := sums3;
                carry_sub_pi  := not sort_sub2(0);

        end if;

--end  compute pi+qi*cos(k)+qr*sin(k)

-- compute pr-qr*cos(k)-qi*sin(k)
--1 compute qr*cos(k)-qi*sin(k)
        sort_sub1:="0000000000000000";
        sort_sub2:="0000000000000000";
        cins1    :="0000000000000000";
        cins2:='0';

        sort_sub1 := var_qr_c_pm;--input from qi*cos(k)
        sort_sub2 := var_qi_s_pm;--input from qr*sin(k)

        sums1(0) := sort_sub1(0) xor not sort_sub2(0);
        cars1(0) := sort_sub1(0) and not sort_sub2(0);
        sums2(0) := sums1(0) xor cins1(0);
        cars2(0) := sums1(0) and cins1(0);
        cars3(0) := cars1(0) or cars2(0);
        cins2    := cars3(0);

        for lp1 in 1 to 15 loop
                sums1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
                cars1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
                sums2(lp1) := sums1(lp1) xor cins2;
```

```
            cars2(lp1) := sums1(lp1) and cins2;
            cars3(lp1) := cars1(lp1) or cars2(lp1);
            cins2     := cars3(lp1);
        end loop;
        sort_sub1:=sums2;
        sort_sub2(0):=cins2;
        for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
        cins2:='0';

        for lp3 in 0 to 15 loop
            sums1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
            cars1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
            sums2(lp3) := sums1(lp3) xor cins2;
            cars2(lp3) := sums1(lp3) and cins2;
            cars3(lp3) := cars1(lp3) or cars2(lp3);
            cins2     := cars3(lp3);
        end loop;

        for lp2 in 0 to 15 loop
            sums3(lp2) := sums2(lp2) xor not sort_sub2(0);
        end loop;

            var_sum_qr_qm := sums3;
            carry_sub_qr :=not sort_sub2(0);


--2 compute pr-qr*cos(k)-qi*sin(k)
--case 1 qr*cos(k)  more than qi*sin(k)
        if(carry_sub_pr='0')then
            sort_sub1:="0000000000000000";
            sort_sub2:="0000000000000000";
            cind1    :="0000000000000000";
            cind2:='0';
```

```
for m14 in 0 to 7 loop
        sort_sub1(m14) := var_ch_real((m2*32)+m14);--input from pi
end loop;
        sort_sub2 := var_sum_qr_qm;--input from qi*sin(k)

sumd1(0) := sort_sub1(0) xor not sort_sub2(0);
card1(0) := sort_sub1(0) and not sort_sub2(0);
sumd2(0) := sumd1(0) xor cind1(0);
card2(0) := sumd1(0) and cind1(0);
card3(0) := card1(0) or card2(0);
cind2    := card3(0);

for lp1 in 1 to 15 loop
        sums1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
        cars1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
        sums2(lp1) := sums1(lp1) xor cins2;
        cars2(lp1) := sums1(lp1) and cins2;
        cars3(lp1) := cars1(lp1) or cars2(lp1);
        cins2      := cars3(lp1);
end loop;
sort_sub1:=sums2;
sort_sub2(0):=cins2;
for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
cins2:='0';

for lp3 in 0 to 15 loop
        sums1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
        cars1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
        sums2(lp3) := sums1(lp3) xor cins2;
        cars2(lp3) := sums1(lp3) and cins2;
        cars3(lp3) := cars1(lp3) or cars2(lp3);
        cins2      := cars3(lp3);
end loop;
```

```
                for lp2 in 0 to 15 loop
                        sums3(lp2) := sums2(lp2) xor not sort_sub2(0);
                end loop;

                var_pro2_qr := sums3;
                carry_sub_qr  := not sort_sub2(0);


--case 2 qr*cos(k)  less than qi*sin(k)

        elsif(carry_sub_pr='1')then
                for m14 in 0 to 7 loop
                        sort_sum1(m14) := var_ch_real((m2*32)+m14);--input from pi
                end loop;
                        sort_sum2 := var_sum_qr_qm;--input from qi*sin(k)

                sumd1(0) := sort_sum1(0) xor sort_sum2(0);
                card1(0) := sort_sum1(0) and sort_sum2(0);
                sumd2(0) := sumd1(0) xor cind1(0);
                card2(0) := sumd1(0) and cind1(0);
                card3(0) := card1(0) or card2(0);
                cind2    := card3(0);
                for lp1 in 1 to 15 loop
                        sumd1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
                        card1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
                        sumd2(lp1) := sumd1(lp1) xor cind2;
                        card2(lp1) := sumd1(lp1) and cind2;
                        card3(lp1) := card1(lp1) or card2(lp1);
                        cind2      := card3(lp1);
                end loop;

                var_pro2_qr := sumd2;
        end if;
```

```
--end compute pr-qr*cos(k)-qi*sin(k)

-- compute pi-qi*cos(k)+qr*sin(k)
--1 compute qi*cos(k)+qr*sin(k)

        sort_sum1 := var_qi_c_pm;--input from qi*cos(k)
        sort_sum2 := var_qr_s_pm;--input from qr*sin(k)

        sumd1(0) := sort_sum1(0) xor sort_sum2(0);
        card1(0) := sort_sum1(0) and sort_sum2(0);
        sumd2(0) := sumd1(0) xor cind1(0);
        card2(0) := sumd1(0) and cind1(0);
        card3(0) := card1(0) or card2(0);
        cind2    := card3(0);

        for lp1 in 1 to 15 loop
                sumd1(lp1) := sort_sum1(lp1)  xor  sort_sum2(lp1);
                card1(lp1) := sort_sum1(lp1)  and  sort_sum2(lp1);
                sumd2(lp1) := sumd1(lp1) xor cind2;
                card2(lp1) := sumd1(lp1) and cind2;
                card3(lp1) := card1(lp1) or card2(lp1);
                cind2      := card3(lp1);
        end loop;

                var_sum_qi_qm := sumd2;


--2 compute pi-qi*cos(k)+qr*sin(k)

                sort_sub1:="0000000000000000";
                sort_sub2:="0000000000000000";
                cins1     :="0000000000000000";
                cins2:='0';
```

```
for m14 in 0 to 7 loop
        sort_sub1(m14) := var_ch_real((m2*32)+8+m14);--input from pi
end loop;
        sort_sub2 := var_sum_qi_qm;--input from qi*sin(k)

sums1(0) := sort_sub1(0) xor not sort_sub2(0);
cars1(0) := sort_sub1(0) and not sort_sub2(0);
sums2(0) := sums1(0) xor cins1(0);
cars2(0) := sums1(0) and cins1(0);
cars3(0) := cars1(0) or cars2(0);
cins2    := cars3(0);

for lp1 in 1 to 15 loop
        sums1(lp1) := sort_sub1(lp1)  xor not sort_sub2(lp1);
        cars1(lp1) := sort_sub1(lp1)  and not sort_sub2(lp1);
        sums2(lp1) := sums1(lp1) xor cins2;
        cars2(lp1) := sums1(lp1) and cins2;
        cars3(lp1) := cars1(lp1) or cars2(lp1);
        cins2      := cars3(lp1);
end loop;
sort_sub1:=sums2;
sort_sub2(0):=cins2;
for lp4 in 1 to 15 loop sort_sub2(lp4):='0';end loop;
cins2:='0';

for lp3 in 0 to 15 loop
        sums1(lp3) := sort_sub1(lp3)  xor  sort_sub2(lp3);
        cars1(lp3) := sort_sub1(lp3)  and  sort_sub2(lp3);
        sums2(lp3) := sums1(lp3) xor cins2;
        cars2(lp3) := sums1(lp3) and cins2;
        cars3(lp3) := cars1(lp3) or cars2(lp3);
        cins2      := cars3(lp3);
end loop;
```

```
                for lp2 in 0 to 15 loop
                        sums3(lp2) := sums2(lp2) xor not sort_sub2(0);
                end loop;

                var_pro2_qi := sums3;
                carry_sub_qi  := not sort_sub2(0);

-- end  compute pi-qi*cos(k)+qr*sin(k)

                for elp in 0 to 15 loop
                        var_real((m2*64)+elp)   := var_pro1_pr(elp);
                        var_real((m2*64)+16+elp) := var_pro1_pi(elp);
                        var_real((m2*64)+32+elp) := var_pro2_qr(elp);
                        var_real((m2*64)+48+elp) := var_pro2_qi(elp);
                end loop;

                end loop;--loop m2
        end loop;--loop lc
        for lpp1 in 0 to 7 loop
        for lpp2 in 0 to 7 loop
                out_real((lpp1*8)+lpp2) <= var_real((lpp1*32)+8+lpp2);
                out_imag((lpp1*8)+lpp2) <= var_real((lpp1*32)+24+lpp2);
        end loop; --end loop lpp2
        end loop;--end loop lpp1
        end if;
end if;
end process g1;
end ganda91;
```

โปรแกรมแสดงผลบนจอโทรทัศน์

```
CPU   "8051.TBL"
     HOF   "INT8"
     INCL  "TABLE.ASM"
          ORG 10H
     XF:    DS 3
     YF:    DS 3
     HLL:   DS 1
     VLL:   DS 1
     SBUFF: DS 1

          ORG 30H
     HFFTH: DS 1
     HFFTL: DS 1
     MFFTL: DS 1
     BUFFL: DS 1
     BUFFH: DS 1

          ORG 0000H
          AJMP   START

          ORG 0100H
     START: LCALL DELAY_IN1
          LCALL CLR_RAM
          LCALL D_WAIT
          LCALL D_PLY
          LCALL D_WAIT
   DELAY_IN1: PUSH  PSW
          MOV   PSW,#08H
          MOV   R6,#09H
     DLY4: MOV   R7,#00H
          NOP
          NOP
          DJNZ  R7,$
```

```
        DJNZ  R6,DLY4
        POP   PSW
        RET
CLR_RAM: MOV   DPTR,#8000H ;CLR RAM
        MOV   A,#0H
 CRAM_1: MOVX  @DPTR,A
        MOV   R2,DPH
        INC   DPTR
        CJNE  R2,#0A0H,CRAM_1
        RET
 D_WAIT: PUSH  PSW
        MOV   PSW,#08H
        MOV   R6,#0F0H
  D_W1: MOV   R7,#00H
        NOP
        NOP
        DJNZ  R7,$
        DJNZ  R6,D_W1
        POP   PSW
        RET
 D_PLY: MOV   XF,#08H     ;SHOW FREQUENCY (VER) UP
        MOV   YF,#24H
        MOV   DPTR,#_FF
        LCALL SFON
        MOV   DPTR,#_FR
        LCALL SFON
        MOV   DPTR,#_FE
        LCALL SFON
        MOV   DPTR,#_FQ
        LCALL SFON
        MOV   DPTR,#_FU
        LCALL SFON
        MOV   DPTR,#_FE
        LCALL SFON
```

```
MOV   DPTR,#_FN
LCALL SFON
MOV   DPTR,#_FC
LCALL SFON
MOV   DPTR,#_FY
LCALL SFON

MOV   XF,#17H       ;SHOW VOLTAGE (VER)
MOV   YF,#24H
MOV   DPTR,#_FV
LCALL SFON
MOV   DPTR,#_FO
LCALL SFON
MOV   DPTR,#_FL
LCALL SFON
MOV   DPTR,#_FT
LCALL SFON
MOV   DPTR,#_FA
LCALL SFON
MOV   DPTR,#_FG
LCALL SFON
MOV   DPTR,#_FE
LCALL SFON

MOV   XF,#0FH       ;SHOW UNIT FREQUENCY (Hz)
MOV   YF,#2FH
MOV   DPTR,#_FH
LCALL SFON
MOV   DPTR,#_FZ
LCALL SFON

MOV   XF,#1DH       ;SHOW UNIT VOLTAGE (V)
MOV   YF,#2FH
MOV   DPTR,#_FV
```

```
LCALL SFON

MOV   XF,#0EH       ;FREQUENCY (VER) DOWN
MOV   YF,#0C9H
MOV   DPTR,#_FF
LCALL SFON
MOV   DPTR,#_FR
LCALL SFON
MOV   DPTR,#_FE
LCALL SFON
MOV   DPTR,#_FQ
LCALL SFON
MOV   DPTR,#_FU
LCALL SFON
MOV   DPTR,#_FE
LCALL SFON
MOV   DPTR,#_FN
LCALL SFON
MOV   DPTR,#_FC
LCALL SFON
MOV   DPTR,#_FY
LCALL SFON

MOV   XF,#06H       ;SHOW LINE HOR (Y)
MOV   YF,#36H
MOV   HLL,#08BH
LCALL HLINE

MOV   XF,#07H       ;SHOW LINE VER (X)
MOV   YF,#0C0H
MOV   VLL,#17H
LCALL VLINE
NOP
RET
```

```
HLINE: MOV   SBUFF,#01H       ;SEND HOR LINE
HL_11: LCALL COX
       MOV   XF+1,XF
       MOV   YF+1,YF
HL_12: LCALL SF_A
       INC   YF+1
       DJNZ  HLL,HL_12
       RET

VLINE: MOV   SBUFF,#0FFH      ;SEND VER LINE
VL_11: LCALL COX
       MOV   XF+1,XF
       MOV   YF+1,YF
VL_12: LCALL SF_A
       INC   XF+1
       DJNZ  VLL,VL_12
       RET

 COX: MOV   A,XF
       JNZ   COX_1
       MOV   A,#20H
COX_1: DEC   A
       MOV   XF,A
       RET

 SF_A: MOV   A,YF+1
       MOV   B,#20H
       MUL   AB
       CLR   C
       ADD   A,XF+1
       MOV   XF,A
       JNC   SF_A1
       INC   B
```

```
SF_A1: MOV   A,#80H
       ADD   A,B
       MOV   YF,A
       PUSH  DPL
       PUSH  DPH
       MOV   A,SBUFF
       MOV   DPH,YF
       MOV   DPL,XF
       MOVX  @DPTR,A
       POP   DPH
       POP   DPL
       RET

SFON:  PUSH  PSW
       MOV   PSW,#0H
       MOV   R2,#8          ;SEND VER FONT
       LCALL COX
       MOV   YF+1,YF
       MOV   YF+2,YF
       MOV   XF+1,XF
SF_1:  CLR   A
       MOVC  A,@A+DPTR
       MOV   SBUFF,A
       LCALL SF_A
       INC   YF+1
       DJNZ  R2,SF_1
       MOV   YF,YF+2
       INC   XF+1
       INC   XF+1
       NOP
       MOV   XF,XF+1
       POP   PSW
       RET
```

```
    ORG 0D00H
_F0: DB  38H,44H,4CH,54H,64H,44H,38H,00H
_F1: DB  10H,30H,50H,10H,10H,10H,10H,00H
_F2: DB  38H,44H,04H,08H,10H,20H,7CH,00H
_F3: DB  38H,44H,04H,18H,04H,44H,38H,00H
_F4: DB  08H,18H,28H,48H,7CH,08H,08H,00H
_F5: DB  7CH,40H,78H,04H,04H,44H,38H,00H
_F6: DB  18H,20H,40H,78H,44H,44H,38H,00H
_F7: DB  7CH,04H,08H,10H,10H,10H,10H,00H
_F8: DB  38H,44H,44H,38H,44H,44H,38H,00H
_F9: DB  38H,44H,44H,3CH,04H,08H,30H,00H
_FA: DB  38H,44H,44H,7CH,44H,44H,44H,00H
_FC: DB  38H,44H,40H,40H,40H,44H,38H,00H
_FE: DB  7CH,40H,40H,78H,40H,40H,7CH,00H
_FF: DB  7CH,40H,40H;78H,40H,40H,40H,00H
_FG: DB  38H,44H,40H,5CH,44H,44H,3CH,00H
_FH: DB  44H,44H,44H,7CH,44H,44H,44H,00H
_FI: DB  38H,10H,10H,10H,10H,10H,38H,00H
_FL: DB  40H,40H,40H,40H,40H,40H,7CH,00H
_FN: DB  44H,44H,64H,54H,4CH,44H,44H,00H
_FO: DB  38H,44H,44H,44H,44H,44H,38H,00H
_FQ: DB  38H,44H,44H,44H,54H,48H,34H,00H
_FR: DB  78H,44H,44H,78H,50H,48H,44H,00H
_FS: DB  38H,44H,40H,38H,04H,44H,38H,00H
_FT: DB  7CH,10H,10H,10H,10H,10H,10H,00H
_FU: DB  44H,44H,44H,44H,44H,44H,38H,00H
_FV: DB  44H,44H,44H,44H,44H,28H,10H,00H
_FY: DB  44H,44H,28H,10H,10H,10H,10H,00H
_FZ: DB  00H,00H,7CH,08H,10H,20H,7CH,00H

    END
```

ภาคผนวก ข
รายการข้อมูลและคุณสมบัติของอุปกรณ์

SCL4528B

CMOS DUAL MONOSTABLE MULTIVIBRATOR

FEATURES

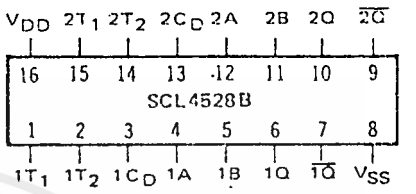
- Two Independent Multivibrators on One Chip
- Triggerable from Leading- or Trailing-Edge Pulse
- Retriggerable
- Resettable
- Q and $\overline{Q}$ Buffered Outputs Available
- Wide Range of Output Pulse Widths

DESCRIPTION

The SCL4528B Dual Multivibrator provides stable retriggerable/resettable one-shot operation for any fixed-voltage timing application. Timing for the circuit is controlled by an external resistor-capacitor combination (R_X-C_X). Adjustment of these components permits generation of output pulse widths from nanoseconds to minutes. Leading-edge and trailing-edge Trigger inputs are provided, and both positive-going and negative-going pulses are available from complementary outputs.

Timing pulses may be terminated at any time by applying a low logic level to the Reset input C_D.

CONNECTION DIAGRAM
(all packages)



Add suffix for package:

C	16-pin Cerdip
D	16-pin Ceramic
E	16-pin Epoxy
F	16-pin Flat
H	Chip

RECOMMENDED OPERATING CONDITIONS

For maximum reliability:

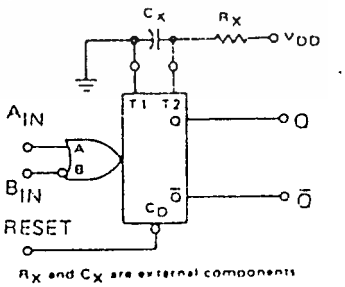
DC Supply Voltage	$V_{DD} - V_{SS}$	3 to 15	Vdc
Operating Temperature	T_A		
C, D, F, H Device		-55 to +125	°C
E Device		-40 to +85	°C

FUNCTION TABLE

INPUTS			OUTPUTS	
C_D	A	B	Q	$\overline{Q}$
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	↑	H	⎍	⎎
H	L	↓	⎍	⎎

H = High Level (Steady State)
L = Low Level (Steady State)
↑ = Transition, Low-to-High
↓ = Transition, High-to-Low
X = Irrelevant (Inc. Transitions)
⎍ = One High-Level Pulse
⎎ = One Low-Level Puise

BLOCK DIAGRAM
(one of two devices)



RX and CX are external components

ELECTRICAL CHARACTERISTICS

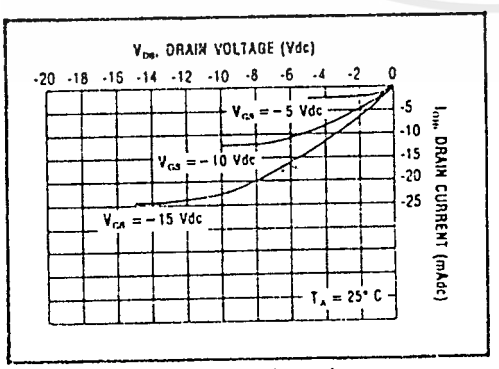
STATIC CHARACTERISTICS [1]

PARAMETER		V_{DD} (Vdc)	CONDITIONS	T_{LOW} [2] Min.	T_{LOW} [2] Max.	+25°C Min.	+25°C Typ.	+25°C Max.	T_{HIGH} [2] Min.	T_{HIGH} [2] Max.	Units
QUIESCENT DEVICE CURRENT	I_{DD}	5	$V_{IN} = V_{SS}$ or V_{DD}	–	5	–	0.05	5	–	150	μAdc
		10	All valid input	–	10	–	0.1	10	–	300	
		20	combinations	–	20	–	0.2	20	–	600	

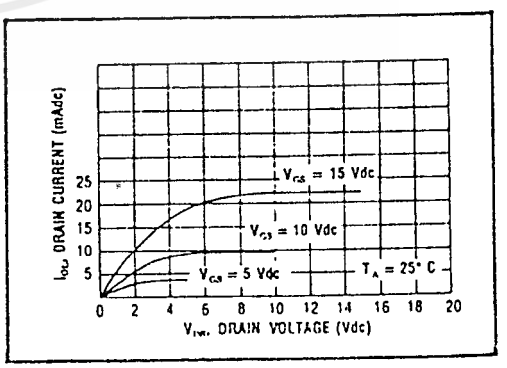
NOTES: [1] Remaining Static Electrical Characteristics are listed under "SCL4000B Series Family Specifications"
[2] T_{LOW} = -55°C for C, D, F, H device.
= -40°C for E device.
T_{HIGH} = +125°C for C, D, F, H device.
= + 85°C for E device.

DYNAMIC CHARACTERISTICS (C_L = 50pF, T_A = 25°C)

PARAMETER		Cx (pF)	Rx (kΩ)	V_{DD} (Vdc)	Min.	Typ.	Max.	Units
PROPAGATION DELAY TIME	t_{PLH}, t_{PHL}							
From A or B		15	5	5	–	270	540	ns
				10	–	90	180	
				15	–	70	140	
		1000	10	5	–	510	1020	ns
				10	–	170	340	
				15	–	120	240	
From C_D		15	5	5	–	270	540	ns
				10	–	90	180	
				15	–	70	140	
		1000	10	5	–	550	1100	ns
				10	–	300	600	
				15	–	250	500	
OUTPUT TRANSITION TIME	t_{TLH}, t_{THL}	–	–	5	–	130	260	ns
				10	–	65	130	
				15	–	50	100	
Note: $\overline{Q}$ Output	t_{TLH}	15	5	5	–	130	260	ns
				10	–	65	130	
				15	–	50	100	
		1000	10	5	–	270	540	ns
				10	–	240	480	
				15	–	220	440	
MINIMUM INPUT PULSE WIDTH A or B Input	PW_{in}	–	–	5	–	70	140	ns
				10	–	30	60	
				15	–	25	50	
OUTPUT PULSE WIDTH MATCH	ΔPW_{out}							
Same package		1000	10	5	–	± 7.5	±15	%
				10	–	±10	±20	
				15	–	±10	±20	
Different packages		1000	10	5	–	–	±50	%
				10	–	–	±50	
				15	–	–	±50	



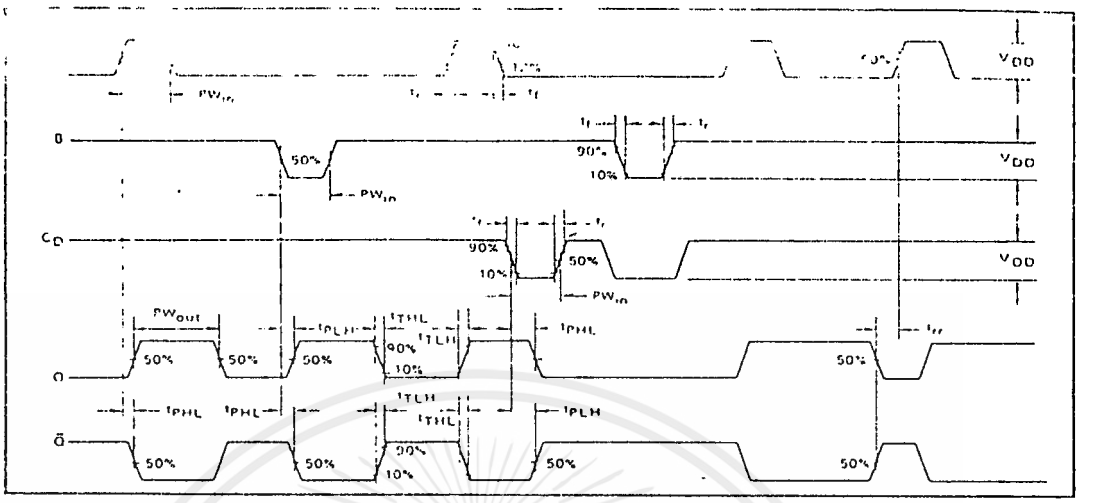
Typical P-Channel
Source Current Characteristics



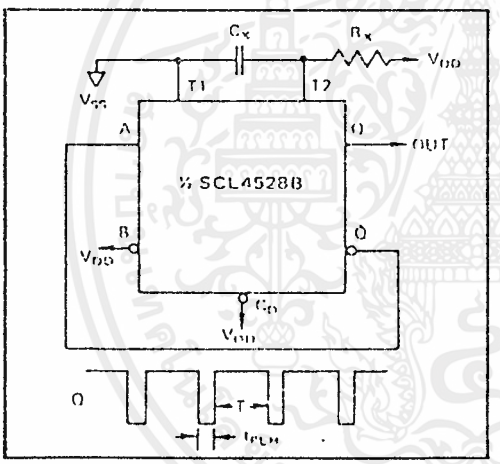
Typical N-Channel
Sink Current Characteristics

SCL4528B

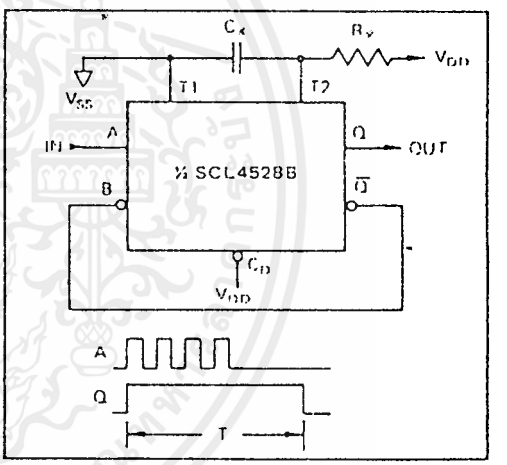
AC ELECTRICAL WAVEFORMS



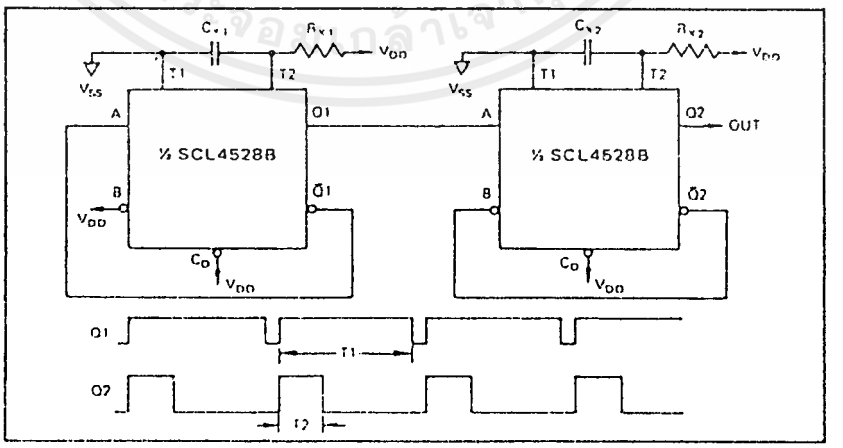
APPLICATIONS INFORMATION



Astable Operation



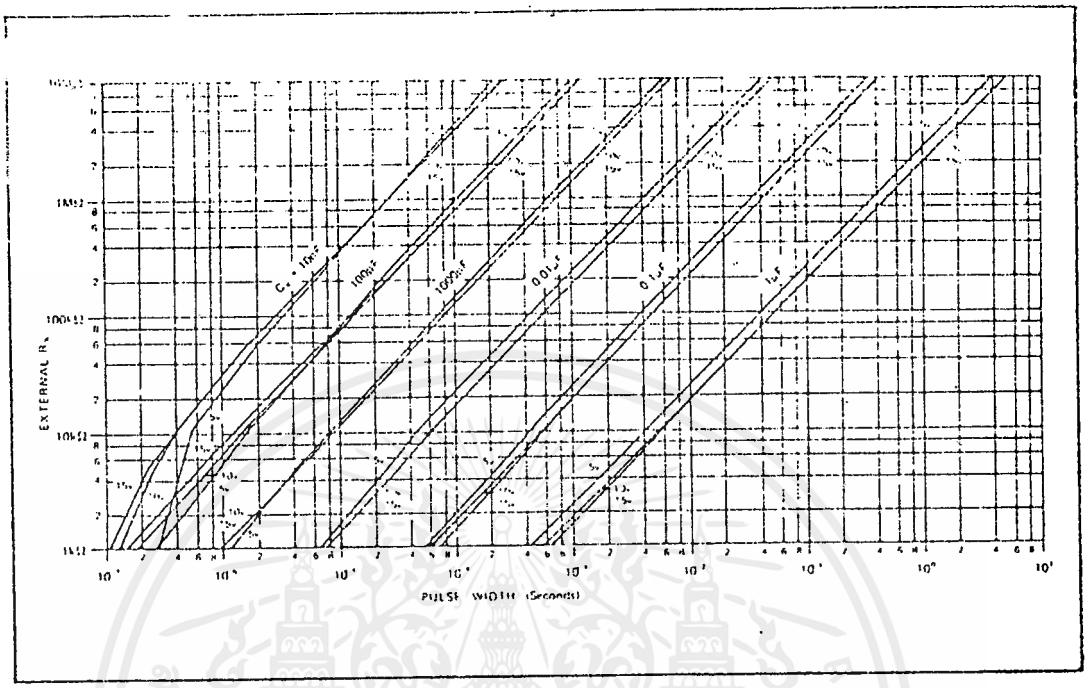
Connection for Non-Retriggerable Operation



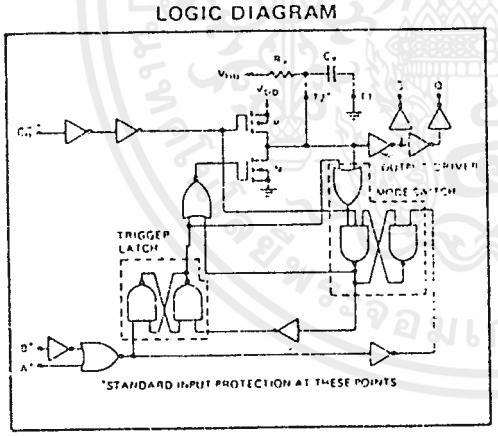
Astable Multivibrator with Adjustable Period and Duty Cycle

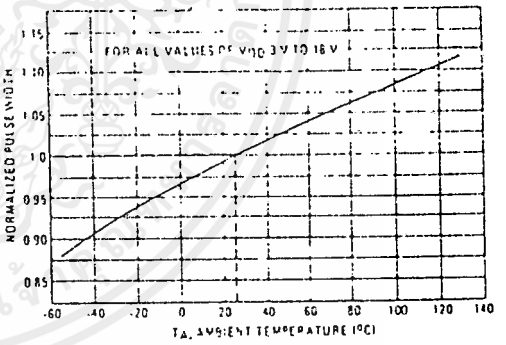
SCL4528B

SCL4528B PULSE WIDTH VS R_X, C_X, V_{DD}



LOGIC DIAGRAM





Normalized Pulse Width versus Temperature

Notes:

There is no effective maximum limit on R_X; recommended minimum value for R_X is 1KΩ. There are no restrictions on the value of C_X.

For proper operation all unused inputs should be tied to a logic level. The mode point (T2) of a unused half of device should be tied high through an external resistor to V_{DD}.

HARRIS SEMICONDUCTOR

CA3318C

CMOS Video Speed 8-Bit Flash A/D Converter

...ber 1993

Features

- CMOS Low Power with SOS Speed (150mW Typ.)
- Parallel Conversion Technique
- 15MHz Sampling Rate (67ns Conversion Time)
- 8-Bit Latched Tri-State Output with Overflow Bit
- ±1 LSB Accuracy (Typ.)
- Single Supply Voltage (4V to 7.5V)
- 2 Units in Series Allow 9-Bit Output
- 2 Units in Parallel Allow 30MHz Sampling Rate

Applications

- TV Video Digitizing (Industrial/Security/Broadcast)
- High-Speed A/D Conversion
- Ultrasound Signature Analysis
- Transient Signal Analysis
- High Energy Physics Research
- High Speed Oscilloscope Storage/Display
- General Purpose Hybrid ADCs
- Optical Character Recognition
- Radar Pulse Analysis
- Motion Signature Analysis
- µP Data Acquisition Systems

Description

The CA3318C is a CMOS parallel (FLASH) analog-to-digital converter designed for applications demanding both low power consumption and high speed digitization.

The CA3318 operates over a wide full scale input voltage range of 4V up to 7.5V with maximum power consumption depending upon the clock frequency selected. When operated from a 5V supply at a clock frequency of 15MHz, the typical power consumption of the CA3318 is 150mW.

The intrinsic high conversion rate makes the CA3318 ideally suited for digitizing high speed signals. The overflow bit makes possible the connection of two or more CA3318s in series to increase the resolution of the conversion system. A series connection of two CA3318s may be used to produce a 9-bit high speed converter. Operation of two CA3318s in parallel doubles the conversion speed (i.e., increases the sampling rate from 15MHz to 30MHz).

256 paralleled auto balanced voltage comparators measure the input voltage with respect to a known reference to produce the parallel bit outputs in the CA3318.

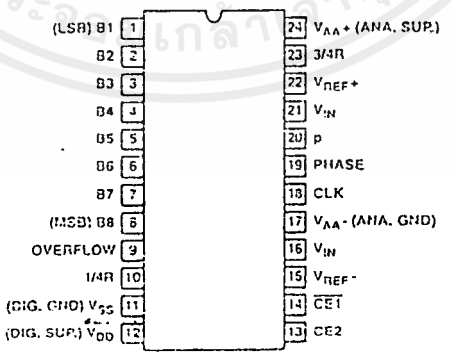
255 comparators are required to quantize all input voltage levels in this 8-bit converter, and the additional comparator is required for the overflow bit.

Ordering Information

PART NUMBER	LINEARITY (INL)	SAMPLING RATE	TEMPERATURE RANGE	PACKAGE
CA3318CE	±1.5 LSB	15MHz (67ns)	-40°C to +85°C	24 Lead Plastic DIP
CA3318CM	±1.5 LSB	15MHz (67ns)	-40°C to +85°C	24 Lead Plastic SOIC
CA3318CD	±1.5 LSB	15MHz (67ns)	-40°C to +85°C	24 Lead Ceramic DIP

Pinout

CA3318C (PDIP, CDIP, SOIC)
TOP VIEW



CAUTION: These devices are sensitive to electrostatic discharge. Users should follow proper I.C. Handling Procedures.

File Number 3103

Absolute Maximum Ratings

Supply Voltage Between V_{DD} and V_{AA}+) . . . . . -0.5V to +8V
(Referenced to V_{SS} or V_{AA}- Terminal, Whichever is More Negative)
Input Voltage Range
CE2 and $\overline{CE1}$ . . . . . V_{AA}- -0.5V to V_{DD} + 0.5V
Clock, Phase, V_{REF}-, 1/2 Ref. . . . . V_{AA}- -0.5V to V_{AA}+ + 0.5V
Clock, Phase, V_{REF}-, 1/4 Ref. . . . . V_{SS}- -0.5V to V_{DD} + 0.5V
V_{IN}, 3/4 REF, V_{REF}+ . . . . . V_{AA}- -0.5V to V_{AA}+ + 7.5V
Output Voltage Range . . . . . V_{SS} - 0.5V to V_{DD} + 0.5V
Bits 1-6, Overflow (Outputs Off)
DC Input Current . . . . . ±20mA
Clock, Phase, $\overline{CE1}$, CE2, V_{IN}, Bits 1-8, Overflow
Operating Voltage Range (V_{DD} or V_{AA}+) . . . . 4V Min to 7.5V Max
Recommended V_{AA}+ Operating Range . . . . . V_{DD} ± 1V
Recommended V_{AA}- Operating Range . . . . . V_{SS} ± 1V
Storage Temperature Range . . . . . -65°C to +150°C
Lead Temperature (Soldering 10s) . . . . . +265°C

Thermal Information

Thermal Resistance	θ_{JA}	θ_{JC}
Ceramic DIP Package	55°C/W	11°C/W
Plastic DIP Package	80°C/W	-
Plastic SOIC Package	75°C/W	-

Maximum Power Dissipation 0.67W
Operating Temperature Range (T_A) . . . . . -40°C to +85°C
Junction Temperature
Ceramic Package . . . . . +175°C
Plastic Package . . . . . +150°C

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

Electrical Specifications

At +25°C, V_{AA}+ = V_{DD} = 5V, V_{REF}+ = 6.4V, V_{REF}- = V_{AA}- = V_{SS}, CLK = 15MHz, All Reference Points Adjusted, Unless Otherwise Specified.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
SYSTEM PERFORMANCE					
Resolution		8	-	-	Bits
Integral Linearity Error		-	-	± 1.5	LSB
Differential Linearity Error		-	-	+1, -0.8	LSB
Offset Error, Unadjusted	V_{IN} = V_{REF}- + 1/2 LSB	-0.5	4.5	6.4	LSB
Gain Error Unadjusted	V_{IN} = V_{REF}- - 1/2 LSB	-1.5	0	1.5	LSB
DYNAMIC CHARACTERISTICS					
Maximum Input Bandwidth	(Note 1) CA3318C	2.5	5.0	-	MHz
Maximum Conversion Speed	CLK = Square Wave	15	17	-	MSPS
Signal to Noise Ratio (SNR) = RMS Signal / RMS Noise	F_S = 15MHz, f_{IN} = 100kHz	-	47	-	dB
	F_S = 15MHz, f_{IN} = 4MHz	-	43	-	dB
Signal to Noise Ratio (SINAD) = RMS Signal / RMS Noise+Distortion	F_S = 15MHz, f_{IN} = 100kHz	-	45	-	dB
	F_S = 15MHz, f_{IN} = 4MHz	-	35	-	dB
Total Harmonic Distortion, THD	F_S = 15MHz, f_{IN} = 100kHz	-	-46	-	dBc
	F_S = 15MHz, f_{IN} = 4MHz	-	-36	-	dBc
Effective Number of Bits (ENOB)	F_S = 15MHz, f_{IN} = 100kHz	-	7.2	-	Bits
	F_S = 15MHz, f_{IN} = 4MHz	-	5.5	-	Bits
Differential Gain Error	Unadjusted	-	2	-	%
Differential Phase Error	Unadjusted	-	1	-	%
ANALOG INPUTS					
Full Scale Range, V_{IN} and (V_{REF}+) - (V_{REF}-)	Notes 2, 4	4		7	V
Input Capacitance, V_{IN}		-	30	-	pF
Input Current, V_{IN}, (See Text)	V_{IN} = 5.0V, V_{REF}+ = 5.0V	-	-	3.5	mA
REFERENCE INPUTS					
Ladder Impedance		270	500	800	Ω

Electrical Specifications At +25°C, V_{DD} = 5V, V_{REF} = 4.4V, V_{REF} = V_{AA} = V_{SS}, CLK = 15MHz, Reference ... unless Otherwise Specified (Continued)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
DIGITAL INPUTS					
Low Level Input Voltage, V_{OL}					
$\overline{CE1}$, CE2	Note 4	-	-	$0.2V_{DD}$	V
Phase, CLK	Note 4	-	-	$0.2V_{AA}$	V
High Level Input Voltage, V_{IH}					
$\overline{CE1}$, CE2	Note 4	$0.7V_{DD}$	-	-	V
Phase, CLK	Note 4	$0.7V_{AA}$	-	-	V
Input Leakage Current, I_I (Except CLK Input	Note 3	-	±0.2	±5	µA
Input Capacitance, C_I		-	3	-	pF
DIGITAL OUTPUTS					
Output Low (Sink) Current	V_O = 0.4V	4	10	-	mA
Output High (Source) Current	V_O = 4.5V	-4	-5	-	mA
Tri-State Output Off-State Leakage Current, I_{OZ}		-	±0.2	±5	µA
Output Capacitance, C_O		-	4	-	pF
TIMING CHARACTERISTICS					
Auto Balance Time (ϕ1)		33	-	∞	ns
Sample Time (ϕ2)	Note 4	25	-	500	ns
Aperture Delay		-	15	-	ns
Aperture Jitter		-	100	-	ps
Data Valid Time, T_D	Note 4	-	50	65	ns
Data Hold Time, T_H	Note 4	25	40	-	ns
Output Enable Time, T_{EN}		-	18	-	ns
Output Disable Time, T_{DIS}		-	18	-	ns
POWER SUPPLY CHARACTERISTICS					
Device Current (I_{DD} + I_A) (Excludes I_{REF})	Continuous Conversion (Note 4)	-	30	60	mA
	Auto Balance (ϕ1)	-	30	60	mA

NOTES:

1. A full scale sine wave input of greater than $F_{CLOCK}/2$ or the specified input bandwidth (whichever is less) may cause an erroneous code. The -3dB bandwidth for frequency response purposes is greater than 30MHz.
2. V_{IN} (Full Scale) or V_{REF}+ should not exceed V_{AA}+ + 1.5V for accuracy.
3. The clock input is a CMOS inverter with a 50kΩ feedback resistor and may be AC coupled with $1V_{P-P}$ minimum source.
4. Parameter not tested, but guaranteed by design or characterization.

Timing Waveforms

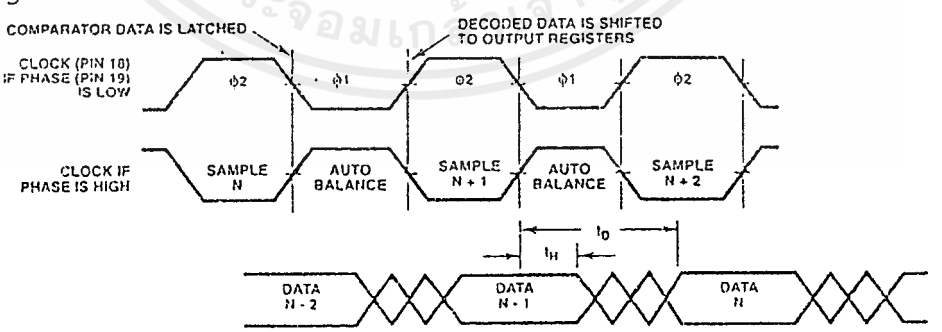


FIGURE 1. INPUT TO OUTPUT TIMING DIAGRAM

Timing Waveforms (Continued)

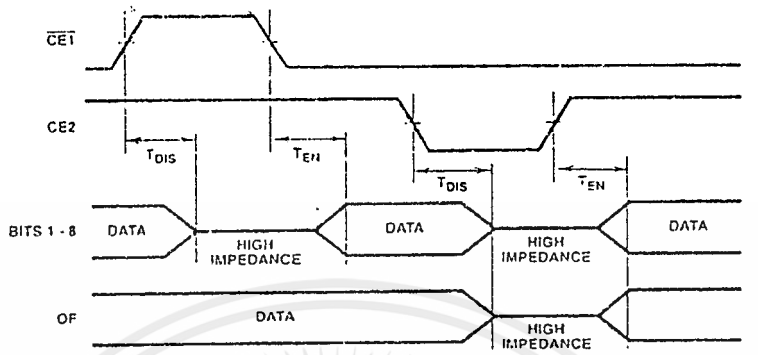


FIGURE 2. OUTPUT ENABLE TIMING DIAGRAM

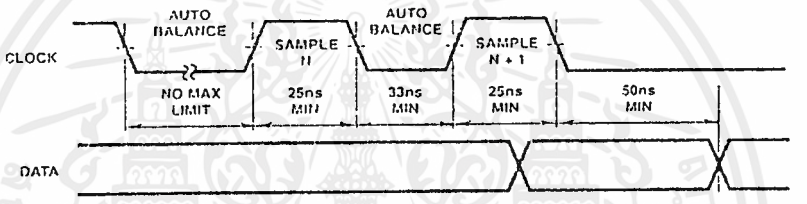


FIGURE 3A. STANDBY IN INDEFINITE AUTO BALANCE (SHOWN WITH PHASE = LOW)

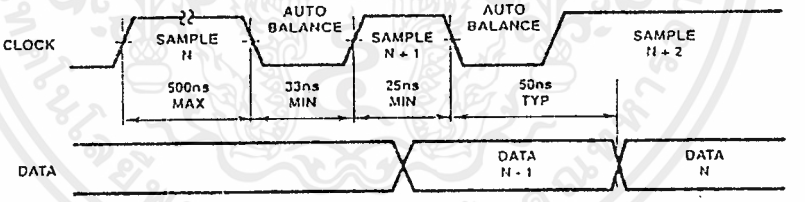


FIGURE 3B. STANDBY IN SAMPLE (SHOWN WITH PHASE = LOW)

FIGURE 3. PULSE MODE OPERATION

Typical Performance Curves

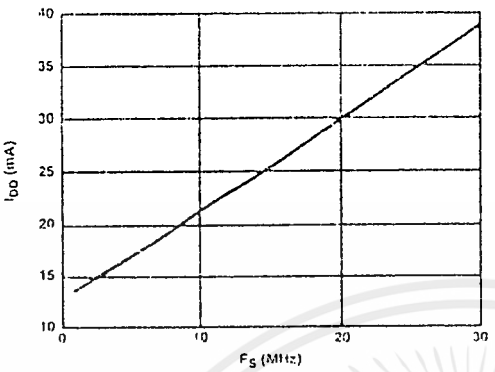


FIGURE 4. DEVICE CURRENT vs SAMPLE FREQUENCY

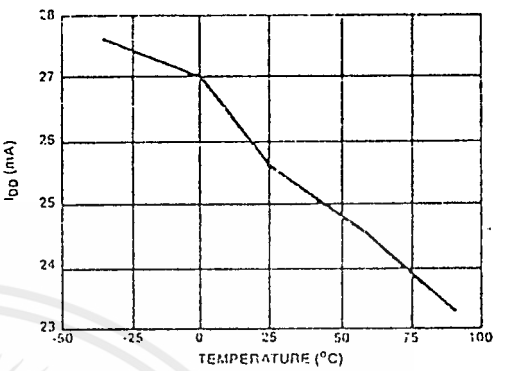


FIGURE 5. DEVICE CURRENT vs TEMPERATURE

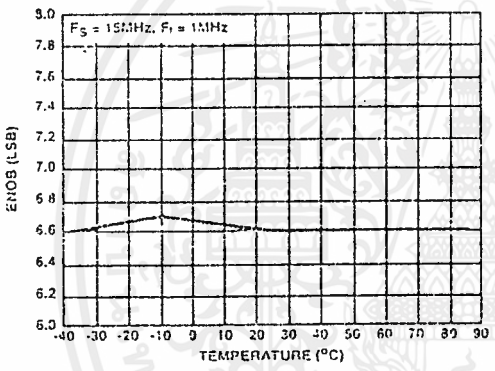


FIGURE 6. ENOB vs TEMPERATURE

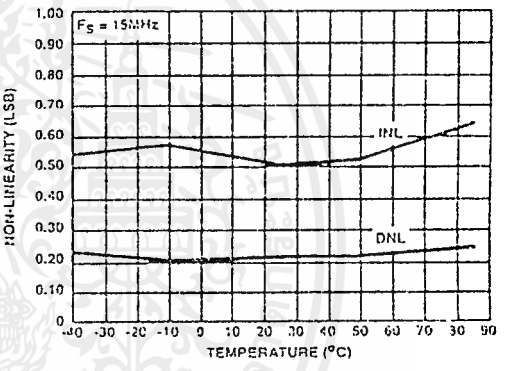


FIGURE 7. NON-LINEARITY vs TEMPERATURE

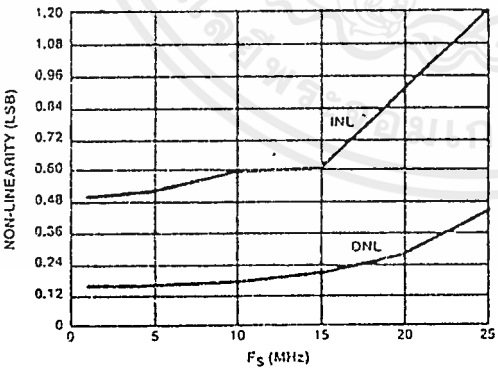


FIGURE 8. NON-LINEARITY vs SAMPLE FREQUENCY

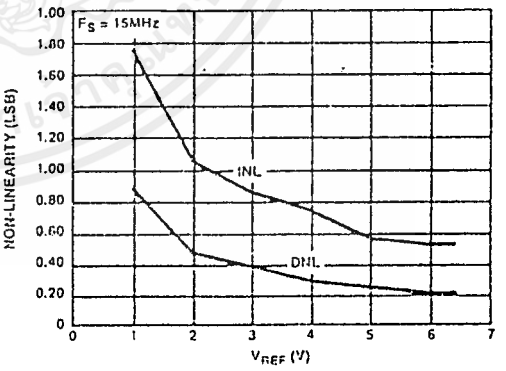


FIGURE 9. NON-LINEARITY vs REFERENCE VOLTAGE

Typical Performance Curves (Continued)

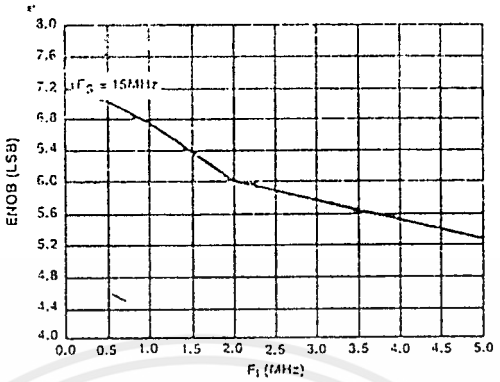


FIGURE 10. ENOB vs INPUT FREQUENCY

Pin Descriptions

PIN	NAME	DESCRIPTION	
1	B1	Bit 1 (LSB)	Output Data Bits (High = True)
2	B2	Bit 2	
3	B3	Bit 3	
4	B4	Bit 4	
5	B5	Bit 5	
6	B6	Bit 6	
7	B7	Bit 7	
8	B8	Bit 8 (MSB)	
9	OF	Overflow	
10	$^1/_4$ R	Reference Ladder $^1/_4$ Point	
11	V_{SS}	Digital Ground	
12	V_{DD}	Digital Power Supply, +5V	
13	CE2	Tri-State Output Enable Input, Active Low, See Truth Table.	
14	CE1	Tri-State Output Enable Input Active High. See Truth Table.	
15	V_{REF}-	Reference Voltage Negative Input	
16	V_{IN}	Analog Signal Input	
17	V_{AA}-	Analog Ground	
18	CLK	Clock Input	
19	PHASE	Sample clock phase control input. When PHASE is low, "Sample Unknown" occurs when the clock is low and "Auto Balance" occurs when the clock is high (see text).	
20	$^1/_2$ R	Reference Ladder Midpoint	
21	V_{IN}	Analog Signal Input	
22	V_{REF}+	Reference Voltage Positive Input	
23	$^3/_4$ R	Reference Ladder $^3/_4$ Point	
24	V_{AA}+	Analog Power Supply, +5V	

CHIP ENABLE TRUTH TABLE

CE1	CE2	B1 - B8	OF
0	1	Valid	Valid
1	1	Tri-State	Valid
X	0	Tri-State	Tri-State

X = Don't Care

Theory of Operation

A sequential parallel technique is used by the CA3318 converter to obtain its high speed operation. The sequence consists of the "Auto-Balance" phase, $\phi 1$, and the "Sample Unknown" phase, $\phi 2$. (Refer to the circuit diagram.) Each conversion takes one clock cycle*. With the phase control (pin 19) high, the "Auto-Balance" ($\phi 1$) occurs during the high period of the clock cycle, and the "Sample Unknown" ($\phi 2$) occurs during the low period of the clock cycle.

* The device requires only a single phase clock The terminology of $\phi 1$ and $\phi 2$ refers to the high and low periods of the same clock.

During the "Auto-Balance" phase, a transmission switch is used to connect each of the first set of 256 commutating capacitors to their associated ladder reference tap. Those tap voltages will be as follows:

$$V_{TAP}(N) = [(N/256)\,V_{REF}] - (1/512)\,V_{REF}]$$
$$= [(2N - 1)/512]\,V_{REF}$$

Where:

V_{TAP} (n) = reference ladder tap voltage at point n.
V_{REF} = voltage across V_{REF}- to V_{REF}+
N = tap number (1 through 256)

The other side of these capacitors are connected to single-stage amplifiers whose outputs are shorted to their inputs by switches. This balances the amplifiers at their intrinsic trip points, which is approximately $(V_{AA}+ - V_{AA}-)/2$. The first set of capacitors now charges to their associated tap voltages.

At the same time a second set of commutating capacitors and amplifiers is also auto-balanced. The balancing of the second-stage amplifier at its intrinsic trip point removes any tracking differences between the first and second amplifier stages. The cascaded auto-balance (CAB) technique, used here, increases comparator sensitivity and temperature tracking.

In the "Sample Unknown" phase, all ladder tap switches and comparator shorting switches are opened. At the same time V_{IN} is switched to the first set of commutating capacitors. Since the other end of the capacitors are now looking into an effectively open circuit, any input voltage that differs from the previous tap voltage will appear as a voltage shift at the comparator amplifiers. All comparators that had tap voltages greater than V_{IN} will go to a "high" state at their outputs. All comparators that had tap voltages lower than V_{IN} will go to a "low" state.

The status of all these comparator amplifiers is AC coupled through the second-stage comparator and stored at the end of this phase (ϕ2) by a latching amplifier stage. The latch feeds a second latching stage, triggered at the end of ϕ1. This delay allows comparators extra settling time. The status of the comparators is decoded by a 256 to 9-bit decoder array, and the results are clocked into a storage register at the end of the next ϕ2.

A 3-stage buffer is used at the output of the 9 storage registers which are controlled by two chip-enable signals. $\overline{CE1}$ will independently disable B1 through B6 when it is in a high state. CE2 will independently disable B1 through B8 and the OF buffers when it is in the low state.

To facilitate usage of this device, a phase control input is provided which can effectively complement the clock as it enters the chip.

Continuous-Clock Operation

One complete conversion cycle can be traced through the CA3318 via the following steps. (Refer to timing diagram.) With the phase control in a "low" state, the rising edge of the clock input will start a "sample" phase. During this entire "high" state of the clock, the comparators will track the input voltage and the first-stage latches will track the comparator outputs. At the falling edge of the clock, all 256 comparator outputs are captured by the 256 latches. This ends the "sample" phase and starts the "auto-balance" phase for the comparators. During this "low" state of the clock, the output of the latches settles and is captured by a second row of latches when the clock returns high. The second-stage latch output propagates through the decode array, and a 9-bit code appears at the D inputs of the output registers. On the next falling edge of the clock, this 9-bit code is shifted into the output registers and appears with time delay t_D as valid data at the output of the tri-state drivers. This also marks the end of the next "sample" phase, thereby repeating the conversion process for this next cycle.

Pulse-Mode Operation

The CA3318 needs two of the same polarity clock edges to complete a conversion cycle: if, for instance, a negative going clock edge ends sample "N", then data "N" will appear after the next negative going edge. Because of this requirement, and because there is a maximum sample time of 500ns (due to capacitor droop), most pulse or intermittent sample applications will require double clock pulsing.

If an indefinite standby state is desired, standby should be in auto-balance, and the operation would be as in Figure 3A.

If the standby state is known to last less than 500ns and lowest average power is desired, then operation could be as in Figure 3B.

Increased Accuracy

In most cases the accuracy of the CA3318 should be sufficient without any adjustments. In applications where accuracy is of utmost importance, five adjustments can be made to obtain better accuracy, i.e., offset trim; gain trim; and $^1/_4$, $^1/_2$ and $^3/_4$ point trim.

Offset Trim

In general, offset correction can be done in the preamp circuitry by introducing a dc shift to V_{IN} or by the offset trim of the op amp. When this is not possible the V_{REF}- input can be adjusted to produce an offset trim. The theoretical input voltage to produce the first transition is $^1/_2$ LSB. The equation is as follows:

$$V_{it1}\text{ (0 to 1 transition)} = {}^1/_2\text{ LSB} = {}^1/_2\,(V_{REF}/256)$$
$$= V_{REF}/512$$

If V_{IN} for the first transition is less than the theoretical, then a single-turn 50Ω pot connected between V_{REF}- and ground will accomplish the adjustment. Set V_{IN} to 1/2 LSB and trim the pot until the 0-to-1 transition occurs.

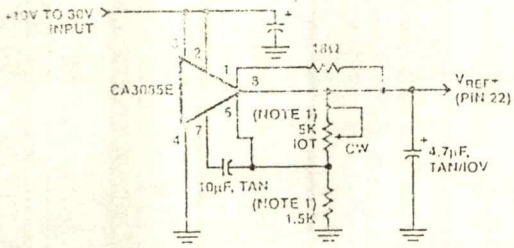
If V_{IN} for the first transition is greater than the theoretical, then the 50Ω pot should be connected between V_{REF}- and a negative voltage of about 2 LSB's. The trim procedure is as stated previously.

Gain Trim

In general, the gain trim can also be done in the preamp circuitry by introducing a gain adjustment for the op amp. When this is not possible, then a gain adjustment circuit should be made to adjust the reference voltage. To perform this trim, V_{IN} should be set to the 255 to overflow transition. That voltage is $^1/_3$ LSB less than V_{REF}+ and is calculated as follows:

$$V_{IN}\text{ (255 to 256 transition)} = V_{REF} - V_{REF}/512$$
$$= V_{REF}(511/512)$$

To perform the gain trim, first do the offset trim and then apply the required V_{IN} for the 255 to overflow transition. Now adjust V_{REF}+ until that transition occurs on the outputs.

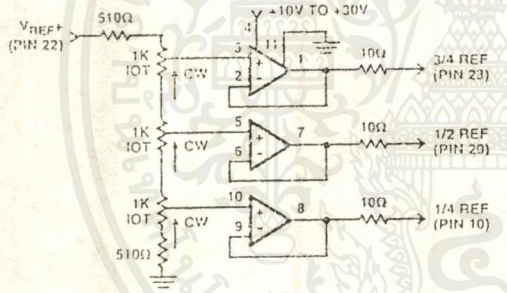


NOTE: Bypass $V_{REF}+$ to analog GND near A/D with 0.1µF ceramic cap. Parts noted should have low temperature drift.

FIGURE 11. TYPICAL VOLTAGE REFERENCE SOURCE FOR DRIVING $V_{REF}+$ INPUT

$^1/_4$ Point Trims

The $^1/_4$, $^1/_2$ and $^3/_4$ points on the reference ladder are brought out for linearity adjusting or if the user wishes to create a nonlinear transfer function. The $^1/_4$ points can be driven by the reference drivers shown (Figure 12) or by 2-K pots connected between $V_{REF}+$ and $V_{REF}-$. The $^1/_2$ (mid-) point should be set first by applying an input of 257/512 x (V_{REF}) and adjusting for an output changing from 128 to 129. Similarly the $^1/_4$ and $^3/_4$ points can be set with inputs of 129/512 and 385/512 x (V_{REF}) and adjusting for counts of 192 to 193 and 64 to 65. (Note that the points are actually $^1/_4$, $^1/_2$ and $^3/_4$ of full scale +1 LSB.)



NOTES:

1. All Op Amps = $^3/_4$ CA324E
2. Bypass all reference points to analog ground near A/D with 0.1µF ceramic caps.
3. Adjust $V_{REF}-$ first, then $^1/_4$, $^3/_4$ and $^1/_4$ points.

FIGURE 12. TYPICAL 1/4 POINT DRIVERS FOR ADJUSTING LINEARITY (USE FOR MAXIMUM LINEARITY)

9-Bit Resolution

To obtain 9-bit resolution, two CA3318's can be wired together. Necessary ingredients include an open-ended ladder network, an overflow indicator, tri-state outputs, and chip-enable controls—all of which are available on the CA3318.

The first step for connecting a 9-bit circuit is to totem-pole the ladder networks, as illustrated in Figure 13. Since the absolute resistance value of each ladder may vary, external trim of the mid-reference voltage may be required.

The overflow output of the lower device now becomes the ninth bit. When it goes high, all counts must come from the upper device. When it goes low, all counts must come from the lower device. This is done simply by connecting the lower overflow signal to the $\overline{CE1}$ control of the lower A/D converter and the CE2 control of the upper A/D converter. The tri-state outputs of the two devices (bits 1 through 8) are now connected in parallel to complete the circuitry. The complete circuit for a 9-bit A/D converter is shown in Figure 14.

Grounding/Bypassing

The analog and digital supply grounds of a system should be kept separate and only connected at the A/D. This keeps digital ground noise out of the analog data to be converted. Reference drivers, input amps, reference taps, and the V_{AA} supply should be bypassed at the A/D to the analog side of the ground. See Figure 15 for a block diagram of this concept. All capacitors shown should be low impedance 0.1µF ceramics and should be mounted as close to the A/D as possible. If $V_{AA}+$ is derived from V_{DD}, a small (10Ω resistor or inductor and additional filtering (4.7µF tantalum) may be used to keep digital noise out of the analog system.

Input Loading

The CA3318 outputs a current pulse to the V_{IN} terminal at the start of every sample period. This is due to capacitor charging and switch feedthrough and varies with input voltage and sampling rate. The signal source must be capable of recovering from the pulse before the end of the sample period to guarantee a valid signal for the A/D to convert. Suitable high speed amplifiers include the HA-5033, HA-2542, and CA3450. Figure 16 is an example of an amplifier which recovers fast enough for sampling at 15MHz.

Output Loading

The CMOS digital output stage, although capable of driving large loads, will reflect these loads into the local ground. It is recommended that a local CMOS buffer such as CD74HC541 E be used to isolate capacitive loads.

Definitions

Dynamic Performance Definitions

Fast Fourier Transform (FFT) techniques are used to evaluate the dynamic performance of the converter. A low distortion sine wave is applied to the input, it is sampled, and the output is stored in RAM. The data is then transformed into the frequency domain with a 4096 point FFT and analyzed to evaluate the dynamic performance of the A/D. The sine wave input to the part is -0.5dB down from fullscale for all these tests.

Signal-to-Noise (SNR)

SNR is the measured RMS signal to RMS noise at a specified input and sampling frequency. The noise is the RMS sum of all of the spectral components except the fundamental and the first five harmonics.

Signal-to-Noise + Distortion Ratio (SINAD)

SINAD is the measured RMS signal to RMS sum of all other spectral components below the Nyquist frequency excluding DC.

Effective Number of Bits (ENOB)

The effective number of bits (ENOB) is derived from the SINAD data. ENOB is calculated from:

$$ENOB = (SINAD - 1.76 + V_{CORR})/6.02$$

where: $V_{CORR} = 0.5dB$

Total Harmonic Distortion (THD)

THD is the ratio of the RMS sum of the first 5 harmonic components to the RMS value of the measured input signal.

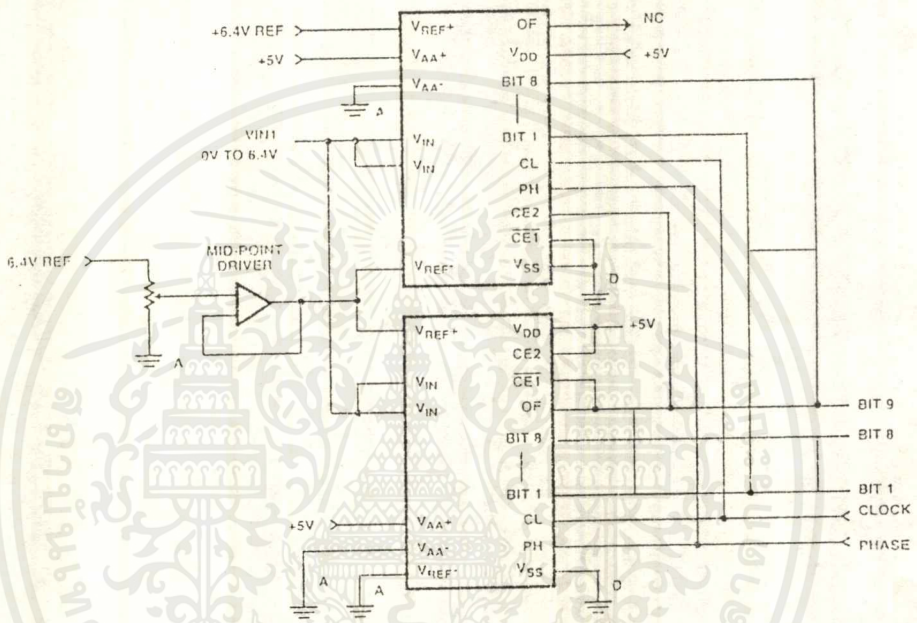


FIGURE 13. USING TWO CA3318s FOR 9-BIT RESOLUTION

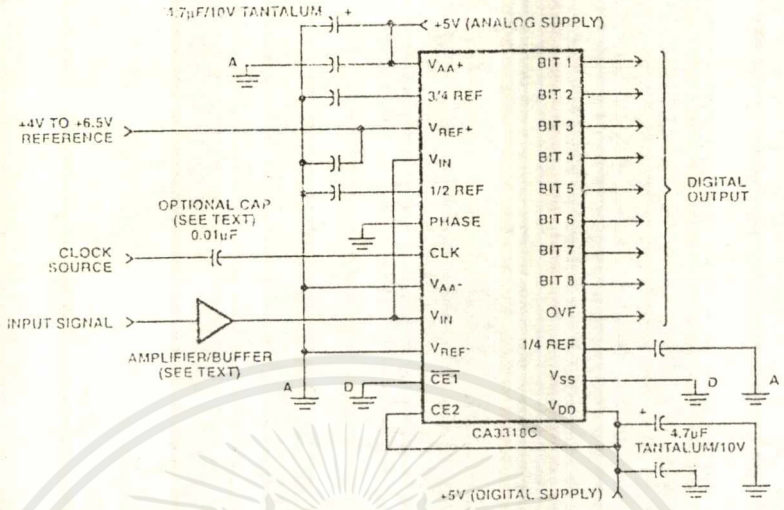


FIGURE 14. TYPICAL CIRCUIT CONFIGURATION FOR THE CA3318 WITH NO LINEARITY ADJUST

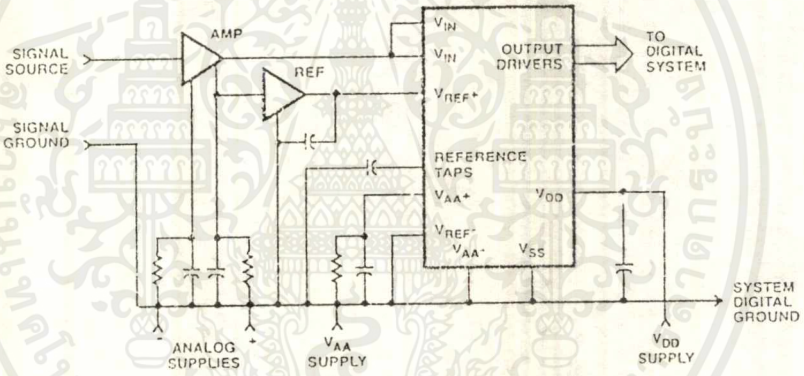
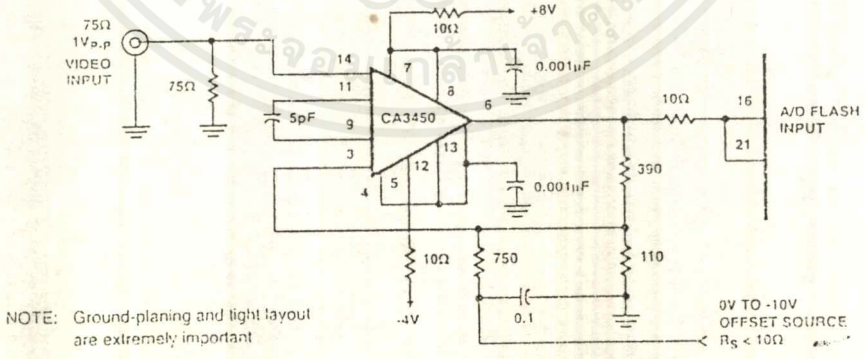


FIGURE 15. TYPICAL SYSTEM GROUNDING/BYPASSING



NOTE: Ground-planing and tight layout are extremely important

FIGURE 16. TYPICAL HIGH BANDWIDTH AMPLIFIER FOR DRIVING THE CA3318

TABLE 1. OUTPUT CODE TABLE

CODE DESCRIPTION	INPUT VOLTAGE (NOTE 1)		BINARY OUTPUT CODE									DECIMAL COUNT
	V_{REF} 5.40V (V)	V_{REF} 5.12V (V)	OF	MSB B8	B7	B6	B5	B4	B3	B2	LSB B1	
Zero	0.00	0.00	0	0	0	0	0	0	0	0	0	0
1 LSB	0.025	0.02	0	0	0	0	0	0	0	0	1	1
2 LSB	0.05	0.04	0	0	0	0	0	0	0	1	0	2
.	.	.					.					.
.	.	.					.					.
.	.	.					.					.
1/4 Full Scale	1.60	1.28	0	0	1	0	0	0	0	0	0	64
.	.	.					.					.
.	.	.					.					.
.	.	.					.					.
1/2 Full Scale - 1 LSB	3.175	2.54	0	0	1	1	1	1	1	1	1	127
1/2 Full Scale	3.20	2.56	0	1	0	0	0	0	0	0	0	128
1/2 Full Scale + 1 LSB	3.225	2.58	0	1	0	0	0	0	0	0	1	129
.	.	.					.					.
.	.	.					.					.
.	.	.					.					.
3/4 Full Scale	4.80	3.84	0	1	1	0	0	0	0	0	0	192
.	.	.					.					.
.	.	.					.					.
.	.	.					.					.
Full Scale - 1 LSB	6.35	5.08	0	1	1	1	1	1	1	1	0	254
Full Scale	6.375	5.10	0	1	1	1	1	1	1	1	1	255
Over Flow	6.40	5.12	1	1	1	1	1	1	1	1	1	511

NOTE: 1. The voltages listed above are the ideal centers of each output code shown as a function of its associated reference voltage.

Reducing Power

Most power is consumed while in the auto-balance state. When operating at lower than 15MHz clock speed, power can be reduced by stretching the sample (ϕ2) time. The constraints are a minimum balance time (ϕ1) of 33ns, and a maximum sample time of 500ns. Longer sample times cause droop in the auto-balance capacitors. Power can also be reduced in the reference string by switching the reference on only during auto-balance.

Clock Input

The Clock and Phase inputs feed buffers referenced to V_{AA}+ and V_{AA}-. Phase should be tied to one of these two potentials, while the clock (if DC coupled) should be driven at least from 0.2 to 0.7 x (V_{AA}+ - V_{AA}-). The clock may also be AC coupled with at least a 1 V_{P-P} swing. This allows TTL drive levels or 5V CMOS levels when V_{AA}+ is greater than 5V.

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายผลิศักดิ์ ตันเจริญ
วันเดือนปีเกิด	7 มกราคม 2519
สถานที่เกิด	จังหวัดสระบุรี
ภูมิลำเนาเดิม	81/1 ม.7 ต.ทับกวาง อ.แก่งคอย จ.สระบุรี 18260
ที่อยู่ปัจจุบัน	81/1 ม. 7 ต.ทับกวาง อ.แก่งคอย จ.สระบุรี 18260
โทรศัพท์	(036)329056
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนแสงวิทยาคม
มัธยมศึกษาตอนต้น	โรงเรียนสระบุรีวิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคสระบุรี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคลวิทยาเขต ภาคตะวันออกเฉียงเหนือ นครราชสีมา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	ร่วมจิตน้อมเกล้าเฉลิมพระเกียรติ
คติพจน์	-

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายศักดิศร วัชราคม
วันเดือนปีเกิด	31 มกราคม 2517
สถานที่เกิด	จังหวัดน่าน
ภูมิลำเนาเดิม	60/1 ถ.สุมนเทวราช ต.ในเวียง
	อ.เมือง จ.น่าน 55000
ที่อยู่ปัจจุบัน	11/115 หมู่ 4 ถ.ช่างอากาศอุทิศ
	เขตดอนเมือง กรุงเทพ ฯ 10210
โทรศัพท์	(02) 9281596,5652784
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนน่านคริสต์เตียนศึกษา
มัธยมศึกษาตอนต้น	โรงเรียนศรีสวัสดิ์วิทยาคาร
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคน่าน
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคสิงห์บุรี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
	ภาควิชาครุศาสตร์วิศวกรรม
	คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ทำวันนี้ให้ดีที่สุด

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายสนธยา วันชัย
วันเดือนปีเกิด	18 มิถุนายน พ.ศ. 2518
สถานที่เกิด	จังหวัดเพชรบูรณ์
ภูมิลำเนาเดิม	110 หมู่ 13 ต.บุ่งคล้า อ.หล่มสัก จ.เพชรบูรณ์ 67110
ที่อยู่ปัจจุบัน	110 หมู่ 13 ต.บุ่งคล้า อ.หล่มสัก จ.เพชรบูรณ์ 67110
โทรศัพท์	-
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนบ้านบุ่งคล้า
มัธยมศึกษาตอนต้น	โรงเรียนท่าพลพิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคเพชรบูรณ์
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขต ภาคตะวันออกเฉียงเหนือ นครราชสีมา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายณัฐสมทัย บุญทองโท
วันเดือนปีเกิด	14 เมษายน 2518
สถานที่เกิด	จังหวัดกำแพงเพชร
ภูมิลำเนาเดิม	109 หมู่ 8 ต.คลองน้ำไหล อ.คลองลาน จ.กำแพงเพชร
ที่อยู่ปัจจุบัน	109 หมู่ 8 ต.คลองน้ำไหล อ คลองลาน จ. กำแพงเพชร
โทรศัพท์	-
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนบ้านสุขสำราญ
มัธยมศึกษาตอนต้น	โรงเรียนคลองลานวิทยา
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคกำแพงเพชร
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคกำแพงเพชร
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ผู้แพ้ในวันนี้ ใช่ว่าวันหน้าจะแพ้เสมอไป

บรรณานุกรม

[1] บรรจง ปิยธำรง และ สุรเชษฐ์ ศรีพลกรัง การออกแบบระบบดิจิตอลโดยใช้ VHDL : การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18.

[2] Jean-Michael Berge and Rolan Airiau. Circuit Synthesis whit VHDL : Kluwer Acadamic Publishers. 1994

[3] Richard Larry Ukeiley. Field Programmable Gate Arrays (FPGAs) The 3000 series : Practice-Hall Inc. 1993

[4] Xilinx Inc. The programmable Logic Data Book : Xilinx Inc. 1994

[5] Zainalabedin Navabi. VHDL Analysis and Modeling of Digital System : McGRAW-HILL.1993

[6] ดร.ธวัช เมฆสวรรค์, นายฟุมิโอะ มิคุมะ, เทคนิคการซ่อมเครื่องรับโทรทัศน์, องค์การค้าคุรุสภา,2519

[7] ดร.ธวัช เมฆสวรรค์, นายฟุมิโอะ มิคุมะ, เทคนิคการซ่อมเครื่องรับโทรทัศน์, บริษัทสำนักพิมพ์ดวงกมล, 2534