เครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลสัญญาณเชิงเลข

Design of Equalizer using Digital Signal Processing

โดย

นายเพิ่มพล กฤชสินชัย

นายอนุวัต พุ่มมะปราง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลสัญญาณเชิงเลข

Design of Equalizer using Digital Signal Processing

โดย

นายเพิ่มพล กฤชสินชัย รหัส 42010244

นายอนุวัต พุ่มมะปราง รหัส 42010431

อาจารย์ที่ปรึกษา

รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษาที่ 2545

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลเชิงเลข

Design of Equalizer using Digital Signal Processing

ผู้จัดทำ นายเพิ่มพล กฤชสินชัย 42010244

นายอนุวัต พุ่มมะปราง 42010431

.....อาจารย์ที่ปรึกษา

(รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์)

เครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลสัญญาณเชิงเลข

นายเพิ่มพล กฤชสินชัย รหัส 42010244
นายอนุวัต พุ่มมะปราง รหัส 42010431
รศ.ดร. สุรพันธุ์ เอื้อไพบูลย์ (อาจารย์ที่ปรึกษา)
ภาคการศึกษาที่ 2 ปีการศึกษา 2544

บทคัดย่อ

โครงงานเครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลสัญญาณเชิงเลข เป็นการศึกษาการสร้างและออกแบบอุปกรณ์ปรับความถี่เสียงโดยใช้วิธีการประมวลผลสัญญาณเชิงเลข มีการแบ่งเป็นย่านความถี่ต่างๆ ซึ่งแต่ละย่านสามารถควบคุมค่าอัตราขยาย(Gain) และตัวประกอบคุณภาพ(Quality factor) ซึ่งพารามิเตอร์เหล่านี้ควบคุมได้โดยวงจรที่ออกแบบด้วย FPGA โดยการเขียนภาษาอธิบายการทำงานของฮาร์ดแวร์(VHDL)

Design of Equalizer using Digital Signal Processing

Mr. Purmpol Krichsinchai (42010244)
Mr. Anuwat Poommaprang (42010431)
Dr. Surapan Airphiboon (Advisor)
2nd Semester, Academic Year 2002

Abstract

This project, "Design of Equalizer using Digital Signal Processing", is the education of building and design audio equalizer by Digital Signal Processing (DSP) method that can be adjusted gain and quality factor individually by circuit designed from FPGA with hardware description language (VHDL).

กิตติกรรมประกาศ

โครงงานการสร้างและออกแบบ เครื่องปรับความถี่เสียงเฉพาะย่านโดยการประมวลผลสัญญาณเชิงเลข แม้ว่าแต่ละขั้นตอนจะพบปัญหาและอุปสรรคมากมาย แต่ก็สำเร็จลุล่วงไปได้ด้วยดี ทั้งนี้ก็ด้วย อาจารย์ที่ปรึกษาคือ รศ.ดร. สุรพันธุ์ เอื้อไพบูลย์ และอาจารย์ทุกท่านในห้องโปรเจค ได้ให้ความรู้ และคำแนะนำต่าง ๆ ซึ่งมีประโยชน์อย่างมากมายสำหรับการแก้ปัญหาที่เกิดขึ้น รวมทั้งเป็นประโยชน์อย่างยิ่งต่อการทำงานในอนาคต

ผู้จัดทำขอขอบคุณคณาจารย์ทุกท่านในภาควิชาอิเล็กทรอนิกส์ที่อุทิศกายและใจเพื่อให้นักศึกษาได้รับความรู้ และเพื่อนๆ ทุกคนที่ให้คำแนะนำ ตลอดจนความช่วยเหลือต่างๆ

.....

(นายเพิ่มพล กฤชสินชัย)

.....

(นายอนุวัต พุ่มมะปราง)

สารบัญ

หน้า

สารบัญรูป

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงาน

การจัดแถบความถี่เสียง เป็นสิ่งที่สำคัญมากในการควบคุมสัญญาณเสียงในช่วงความถี่ที่มนุษย์สามารถได้ยินและมักจะใช้อุปกรณ์ที่เรียกว่า เครื่องปรับความถี่เสียงเฉพาะย่าน หรือ Equalizer ซึ่งเราสามารถพบทั่วไปในห้องบันทึกเสียง,เครื่องเสียงรถยนต์หรือในระบบเครื่องเสียงทั่วไป ฯลฯ Equalizer จะประกอบด้วยตัวกรองชนิดต่างๆ ซึ่งเนื้อหาในตอนนี้จะอธิบายถึงการใช้ตัวกรองแบบชนิดป้อนกลับ(recursive filter)เป็นเครื่องมือในการสร้าง Equalizers

หลักการพื้นฐานของ Equalizer คือตัวกรองที่สามารถปรับการตอบสนองทางความถี่ เพื่อที่จะจัดรูปร่าง (Shaping) สเปคตรัมของสัญญาณเสียง เพื่อให้ได้คุณสมบัติตามที่ต้องการ สำหรับ equalizer แบบดั้งเดิมนั้นจะใช้ตัวกรองแบบอนาลอก แต่ในปัจจุบัน มีแนวโน้มที่จะเป็นแบบดิจิตอลเกือบทั้งหมด เนื่องจากสามารถที่จะให้คุณภาพเสียงที่ดี และลดค่าใช้จ่ายในการผลิตได้มากในอนาคต ซึ่งทั้งหมดนี้พัฒนาและสร้างขึ้นได้ด้วยหลักการของการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing, DSP)

1.2 ขีดความสามารถของโครงงาน

Equalizer ในโครงงานนี้จะเป็นแบบกราฟิก Equalizer ซึ่งแบ่งความถี่ในช่วงความถี่เสียงออกเป็นช่วงต่าง ๆ เพื่อทำการออกแบบตัวกรองให้สามารถปรับแต่งสัญญาณโดยเพิ่ม (Boost) หรือ (Cut) ลดระดับสัญญาณในช่วงความถี่นั้น ๆ และตัวกรองแต่ละตัวสามารถปรับค่าตัวประกอบคุณภาพ (Quality factor, Q) โดยจะนำFPGA มาทำหน้าที่ควบคุมและส่งค่าให้แก่บอร์ด DSP

ในการสร้างตัวกรองแบบดิจิตอล ได้ใช้บอร์ดประมวลผลสัญญาณดิจิตอล TMS320C31 DSK จำนวนแถบความถี่ (Band) จึงถูกจำกัดที่ความเร็วในการประมวลผลและหน่วยความจำภายใน ของไมโครโพรเซสเซอร์ เพื่อที่จะสามารถทำงานได้ในแบบเวลาจริง (Real-time)

1.3 เนื้อหาโดยสังเขป

บทที่ 2 เนื้อหาและทฤษฎี

เนื้อหาในบทนี้จะกล่าวถึงทฤษฎีการออกแบบตัวกรองสัญญาณเสียงชนิดป้อนกลับ (recursive audio filter) ,การใช้งานบอร์ด TMS320C31 DSK ,ทฤษฎีของอุปกรณ์ FPGA และบอร์ด PLD – A01

บทที่ 3 การออกแบบและการทดลอง

ในบทนี้จะกล่าวถึงรายละเอียดในการออกแบบตัวกรองที่ความถี่ต่าง ๆ รวมทั้งการจำลองผลลัพธ์ด้วยโปรแกรม MATLAB และส่วนในการเขียนโปรแกรมภาษาซี เพื่อสั่งงานบอร์ด TMS320C31 DSK ลักษณะโครงสร้างของอุปกรณ์ควบคุมที่ออกแบบบนบอร์ด FPGA และผลลัพธ์ที่ได้ซึ่งก็คือการตอบสนองทางความถี่ของตัวกรอง

บทที่ 4 สรุปผลการทดลองปัญหาและแนวทางแก้ไข

จะเป็นการสรุปผลการจากผลลัพธ์ที่ได้จากการวัด ปัญหาที่เกิดขึ้น สาเหตุของปัญหา รวมทั้งการแก้ไข และแนวทางในการพัฒนาโครงงานต่อไปในอนาคต

บทที่ 2
เนื้อหาและทฤษฎี

2.1 ตัวกรองชนิดป้อนกลับ (Recursive Audio Filter)

Peak Filter

เป็นตัวกรองที่ใช้สำหรับการเพิ่ม (boosting) หรือลด(cutting) ความถี่ที่ต้องการ โดยใช้สมการส่งผ่าน (Transfer function) อันดับ 2 ของตัวกรองแบบความถี่ผ่าน เข้ามาช่วย

$$H_{BP}(s) = \frac{(H0/Q_\infty)s}{s^2+s/Q_\infty+1} \quad (2.1)$$

จะได้ Transfer function ของ peak filter เป็น

$$H(s) = 1+H_{BP}(s) \quad (2.2)$$

$$= \frac{s^2+(1+H0)s/Q_\infty+1}{s^2+s/Q_\infty+1} \quad (2.3)$$

$$= \frac{s^2+(V0)s/Q_\infty+1}{s^2+s/Q_\infty+1} \quad (2.4)$$

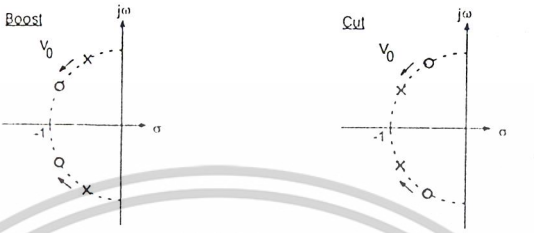
Transfer function ข้างบนนี้เป็นการนำ ตัวกรองชนิดผ่านตลอดมาต่อขนานกับตัวกรองแบบความถี่ผ่านอันดับ 2 โดย

H0 คือ ขนาดของผลตองสนองความถี่ที่ความถี่ = 0

Q_∞ คือ ค่า Q-factor ของตัวกรอง ณ ความถี่ที่ต้องการ

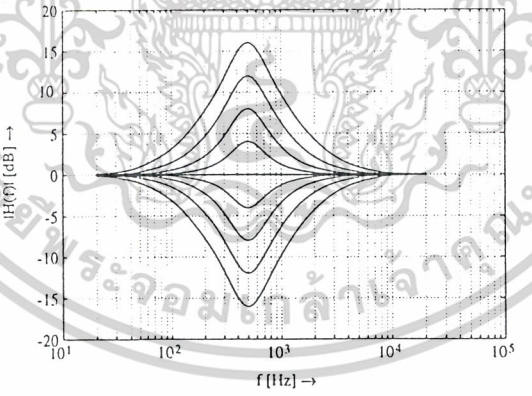
V0 คือ ขนาดของผลตองสนองความถี่ ณ ความถี่ที่ต้องการ

Pole และ zero ของตัวกรองจะเคลื่อนที่อยู่บนวงกลมหนึ่งหน่วย โดยการปรับค่าV0 โดยการเคลื่อนที่ของ complex pole จะสัมพันธ์กับ complex zero รูปที่ 2.1 จะแสดงให้เห็นถึงกรณี boost และ cut และเมื่อเพิ่มค่า Q-factor complex pole จะเคลื่อนที่เข้าหาแกน $j\omega$ บนวงกลมหนึ่งหน่วย

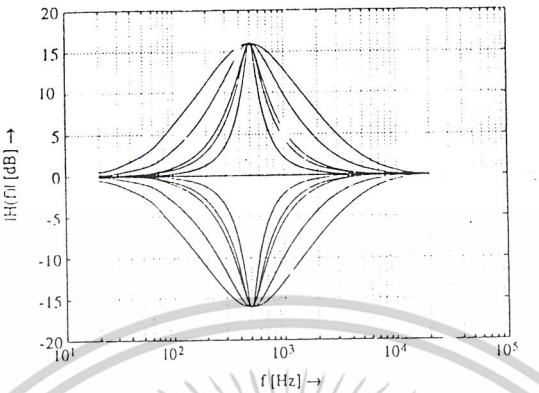


รูปที่ 2.1 ตำแหน่งของ pole และ zero ของpeak filter

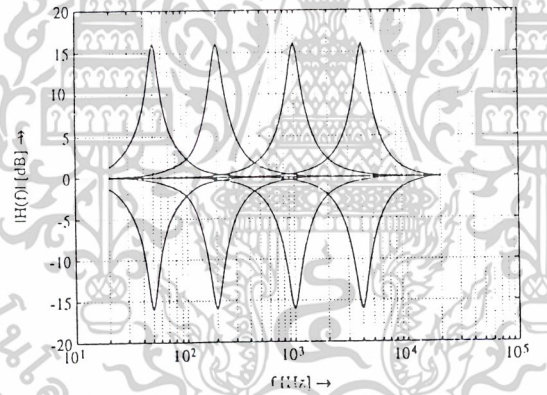
รูปที่ 2.2 แสดงขนาดของผลตอบสนองความถี่ โดยการปรับเปลี่ยนค่า V0 ที่ความถี่กลาง 500Hz และ Q-factor 1.25 รูปที่ 2.3 แสดงขนาดของผลตอบสนองความถี่โดยการปรับเปลี่ยนค่า Q-factor และ รูปที่ 2.4 แสดงขนาดของผลตอบสนองความถี่ โดยการปรับเปลี่ยนค่า ความถี่กลาง(fc)



รูปที่2.2 แสดงขนาดของผลตอบสนองความถี่ของ peak filter fc=500Hz, Q_∞= 1.25



รูปที่ 2.3 แสดงขนาดของผลตอบสนองความถี่ ของpeak filter boost / cut $\pm$16 dB, fc =500 Hz, Q_∞= 0.707, 1.25, 2.5, 3, 5.



รูปที่2.4 แสดงขนาดของผลตอบสนองความถี่ ของ Peak filter boost / cut $\pm$20 dB Q_∞ = 1.25, fc = 20, 200, 1000, 4000 Hz.

2.2 การแปลงกลุ่มตัวแปร (Mapping to Z-domain)

การทำให้ตัวกรองที่ออกแบบเป็นตัวกรองเชิงเลขนั้นเราต้องนำตัวกรองข้างต้นที่มี Transfer function อยู่ใน S-domain มาเปลี่ยนให้อยู่ใน Z-domain โดยเราจะใช้วิธี bilinear transformation โดยจัดให้

$$s = \frac{2}{T}\frac{z-1}{z+1} \qquad (2.5)$$

ตารางที่1 ประกอบด้วยสูตรการหาค่าสัมประสิทธิ์ Transfer function ของตัวกรองอันดับ 2

$$H(z) = \frac{a0+a1z^{-1}+a2z^{-2}}{1+b1z^{-1}+b2z^{-2}} \qquad (2.6)$$

ซึ่งใช้วิธีการแปลงเชิงเส้นคู่ (Bilinear transformation และ) ค่า K = tan(ΩcT/2) สำหรับตัวกรองที่ความถี่ต่างกันไป

ตารางที่ 2.1 Peak filter design

Peak (boost V0 = $10^{G/20}$)				
a0	a1	a2	b1	b2
$\frac{1+V0K/Q_\infty+K^2}{1+K/Q_\infty+K^2}$	$\frac{2(K^2-1)}{1+K/Q_\infty+K^2}$	$\frac{1-V0K/Q_\infty+K^2}{1+K/Q_\infty+K^2}$	$\frac{2(K^2-1)}{1+K/Q_\infty+K^2}$	$\frac{1-K/Q_\infty+K^2}{1+K/Q_\infty+K^2}$

Peak (cut V0 = $10^{-G/20}$)				
a0	a1	a2	b1	b2
$\frac{1+K/Q_\infty+K^2}{1+V0K/Q_\infty+K^2}$	$\frac{2(K^2-1)}{1+K/Q_\infty+K^2}$	$\frac{1-K/Q_\infty+K^2}{1+V0K/Q_\infty+K^2}$	$\frac{2(K^2-1)}{1+V0K/Q_\infty+K^2}$	$\frac{1-V0K/Q_\infty+K^2}{1+V0K/Q_\infty+K^2}$

2.3 สมการผลต่างสืบเนื่อง (Difference Equation)

สมการผลต่างสืบเนื่อง เป็นวิธีการอธิบายคุณสมบัติของระบบเชิงเลขชนิดหนึ่ง ซึ่งจะอธิบายคุณสมบัติของระบบในโดเมนของเวลา รูปแบบของสมการผลต่างสืบเนื่อง N อันดับ เขียนได้ดังนี้

$$y(n) = \sum_{k=1}^{k=N} a_k y(n-k) + \sum_{K=0}^{K=L} b_k x(n-k) \qquad (2.7)$$

โดยที่ x(n) เป็นลำดับสัญญาณเข้าเชิงเลข

y(n) เป็นลำดับสัญญาณออกเชิงเลข

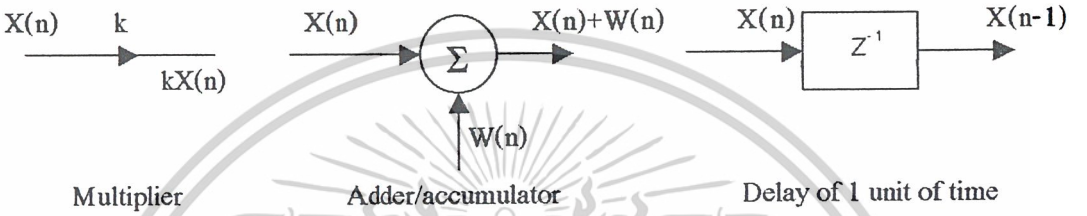
a_k และ b_k เป็นค่าสัมประสิทธิ์ของสมการ

ทฤษฎีโครงสร้างตัวกรองเชิงเลข

ตัวกรองเชิงเลขจะประกอบด้วยองค์ประกอบพื้นฐาน ดังต่อไปนี้

- **ตัวบวก (Adder)**
- **ตัวคูณ (Multiplier)**
- **ตัวหน่วง (Delay)**

สัญลักษณ์ขององค์ประกอบพื้นฐานทั้ง 3ตัวสามารถแสดงได้ดังรูปที่ 2.5

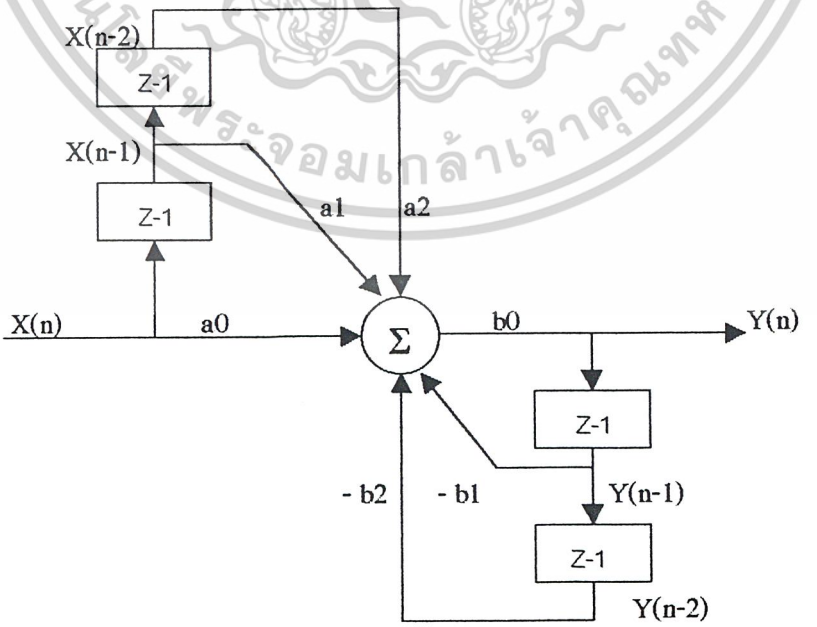


รูปที่ 2.5 แสดงองค์ประกอบพื้นฐานทั้ง 3 ตัว ที่ใช้ใน โครงสร้างของตัวกรองเชิงเลข

ตัวกรองเชิงเลขแบบIIRสามารถอธิบายได้ด้วยสมการผลต่างสืบเนื่องดังนี้

$$y(n) = \sum_{i=0}^{M} a_i x(n-1) + \sum_{i=1}^{N} b_i y(n-1) \qquad (2.8)$$

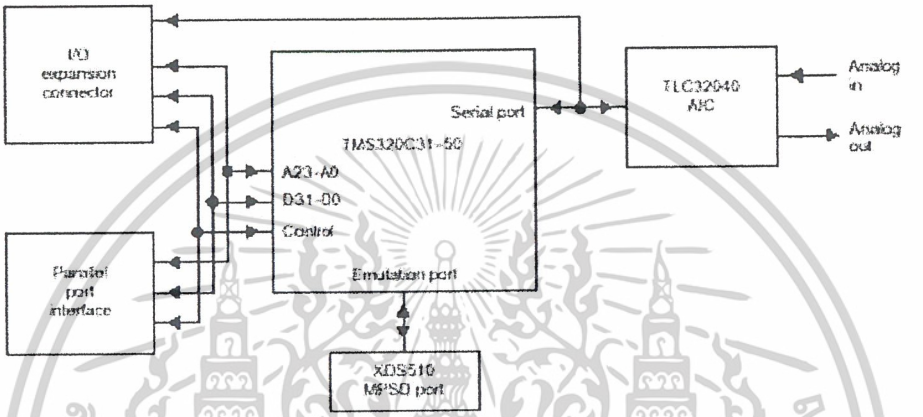
จากสมการผลต่างสืบเนื่องเราสามารถนำมาเขียนเป็น โครงสร้างตัวกรอง ได้หลายรูปแบบ เช่น โครงสร้างแบบโดยตรง 1(Direct Form I)ดังในรูปที่ 2.6



รูปที่ 2.6 โครงสร้างตัวกรองอันดับ2แบบ Direct form I

2.4 บอร์ดประมวลผลสัญญาณดิจิตอล TMS320C31 DSK

บอร์ด TMS320C31 DSK เป็นบอร์ดเอนกประสงค์ในการประมวลผลสัญญาณดิจิตอล ของ บริษัท Texus Instruments ซึ่งสามารถนำไปสร้างและพัฒนาระบบ ได้โดยการโปรแกรมชุดคำสั่งเพื่อกำหนดให้บอร์ดทำงานในลักษณะต่าง ๆ และยังสามารถเพิ่มในส่วนของฮาร์แวร์ทั้งในส่วนของการทำงานของบอร์ดเองและการเชื่อมต่อกับระบบต่างๆภายนอกได้อีกด้วย



รูปที่ 2.7 บล็อคไดอะแกรมของบอร์ด TMS320C31 DSK

2.4.1 ส่วนประกอบต่าง ๆ ที่สำคัญของบอร์ด TMS320C31 DSK มีดังนี้

1. ตัวประมวลผลสัญญาณดิจิตอล TMS320C31
   ทำหน้าที่เป็นตัวประมวลผลสัญญาณหลักของระบบซึ่งทำงานแบบทศนิยมลอย (Floating Point) ขนาด 32 บิต
2. วงจรเชื่อมต่อกับสัญญาณอนาลอก TLC 32040 (AIC)
   ทำหน้าที่แปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลและส่งให้กับตัวประมวลผลสัญญาณดิจิตอลแล้ว และรับสัญญาณดิจิตอลจากตัวประมวลผลสัญญาณดิจิตอลแล้วแปลงกลับเป็นสัญญาณอนาลอก โดยสามารถทำการเลือกกำหนดความถี่ในการสุ่มตัวอย่างได้สูงสุด 20 KHz
3. ออสซิเลเตอร์หลัก (Main Oscillator)
   ทำหน้าที่กำเนิดความถี่ 50MHz จ่ายให้แก่ตัวประมวลผลสัญญาณ
4. ตัวกำหนดแรงดันหลัก (Voltage Regulator)
   ทำหน้าที่กำหนดแรงดันจ่ายให้แก่อุปกรณ์ต่าง ๆ บนบอร์ด

5. พอร์ตขนาน (Parallel Port)

ทำหน้าที่เป็นตัวเชื่อมต่อบอร์ดพัฒนาระบบประมวลผลสัญญาณดิจิตอล TMS320C31 DSK เข้ากับคอมพิวเตอร์เพื่อใช้ในการโหลดโปรแกรม การทดสอบ ใช้คอนเนคเตอร์แบบ DB-25

6. วงจรเชื่อมต่อกับโฮสต์ (Host Interface Logic)

ทำหน้าที่เชื่อมต่อตัวประมวลผลสัญญาณดิจิตอล TMS320C31 เข้ากับพอร์ตขนาน

7. คอนเนคเตอร์สำหรับการขยาย (Expansion Connector)

ทำหน้าที่เป็นช่องทางส่งข้อมูล ระหว่างบอร์ดกับอุปกรณ์ภายนอกที่ทำการเชื่อมต่อ

8. คอนเนคเตอร์แบบ RCA

ทำหน้าที่เชื่อมต่อสัญญาณอนาลอกเข้ากับวงจรเชื่อมต่อกับสัญญาณอนาลอก (AIC)

9. ฟิวส์แบบรีเซ็ตได้ (Resettable Fuse)

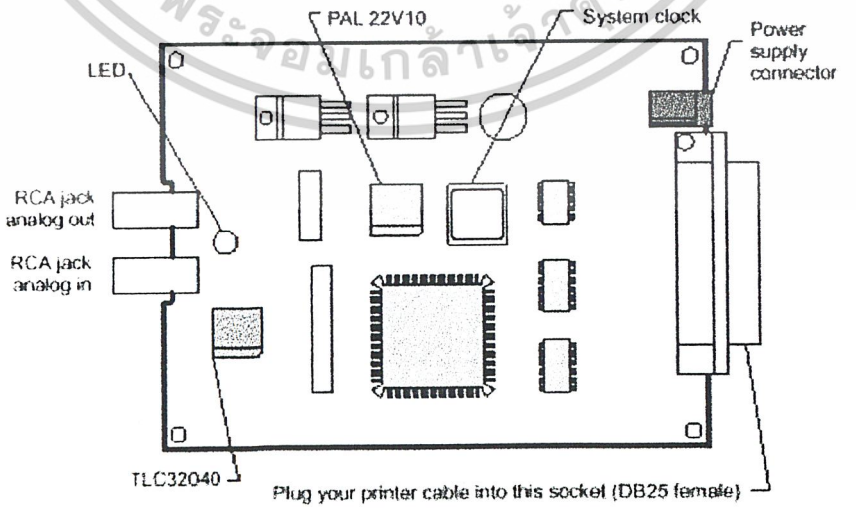
ทำหน้าที่กันกระแสเกินและสามารถรีเซ็ตได้ในภายหลัง โดยไม่ต้องเปลี่ยน

10. จัมเปอร์ (Jumper Block Header)

ทำหน้าที่เชื่อมวงจรสัญญาณอนาลอก(AIC) เข้ากับพอร์ตอนุกรมของตัวประมวลผลสัญญาณดิจิตอล TMS320C31 สามารถถอดออกเพื่อจ่ายสัญญาณจากอุปกรณ์ที่พัฒนาขึ้นให้กับพอร์ตอนุกรมของตัวประมวลผลสัญญาณดิจิตอล TMS320C31 แทนได้

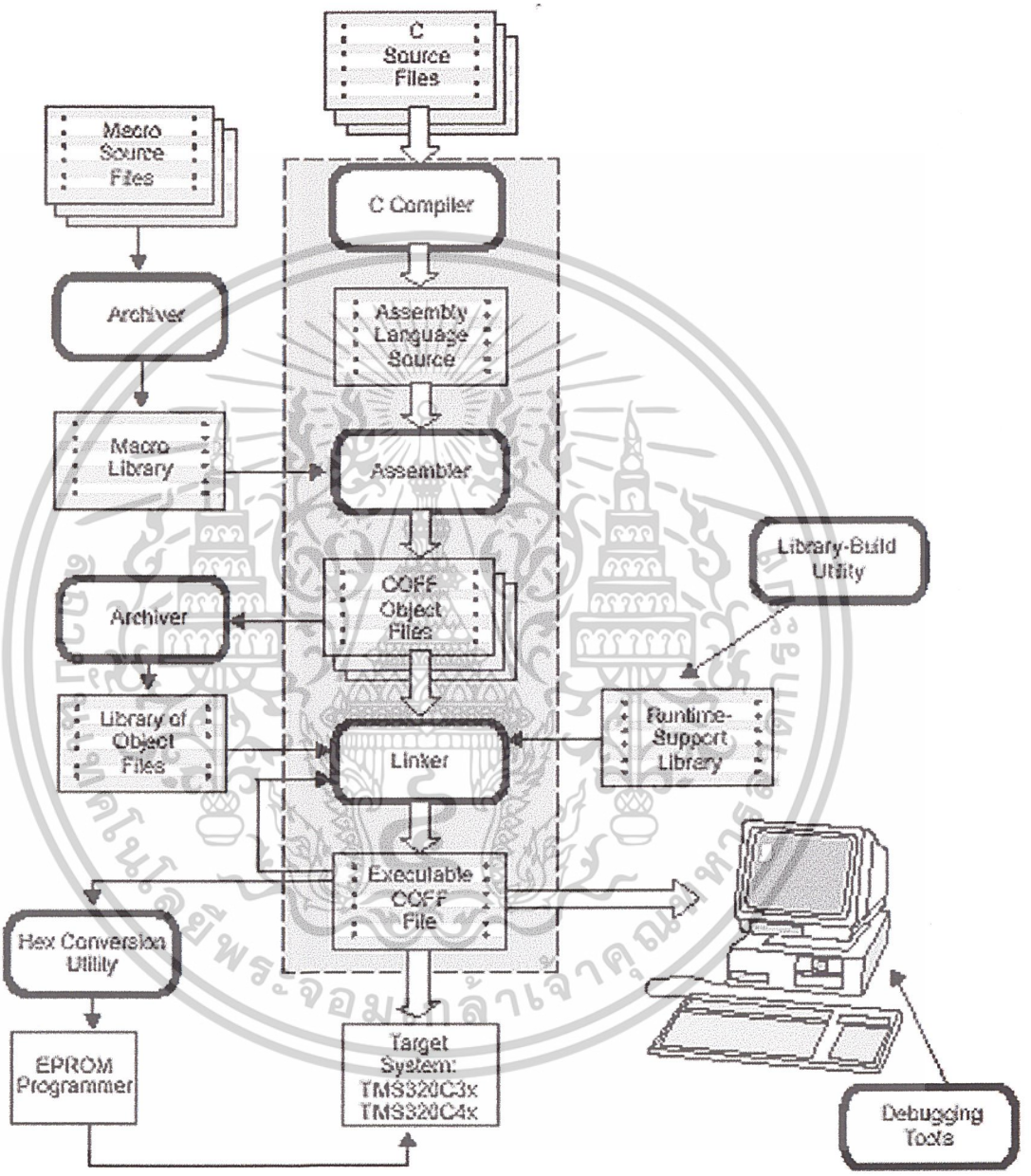
11. พอร์ตอีมูเลเตอร์แบบ XDS

สามารถใช้เชื่อมต่อเพื่ออัพเกรดบอร์ดประมวลผลสัญญาณดิจิตอล TMS320C31 DSK



รูปที่ 2.8 ส่วนประกอบและอุปกรณ์ต่าง ๆ ในบอร์ด TMS320C31 DSK

2.4.2 ขั้นตอนในการพัฒนากระบวนการทำงาน (Algorithm) สำหรับตัวประมวลผลสัญญาณดิจิตอลด้วยโปรแกรมภาษาซี



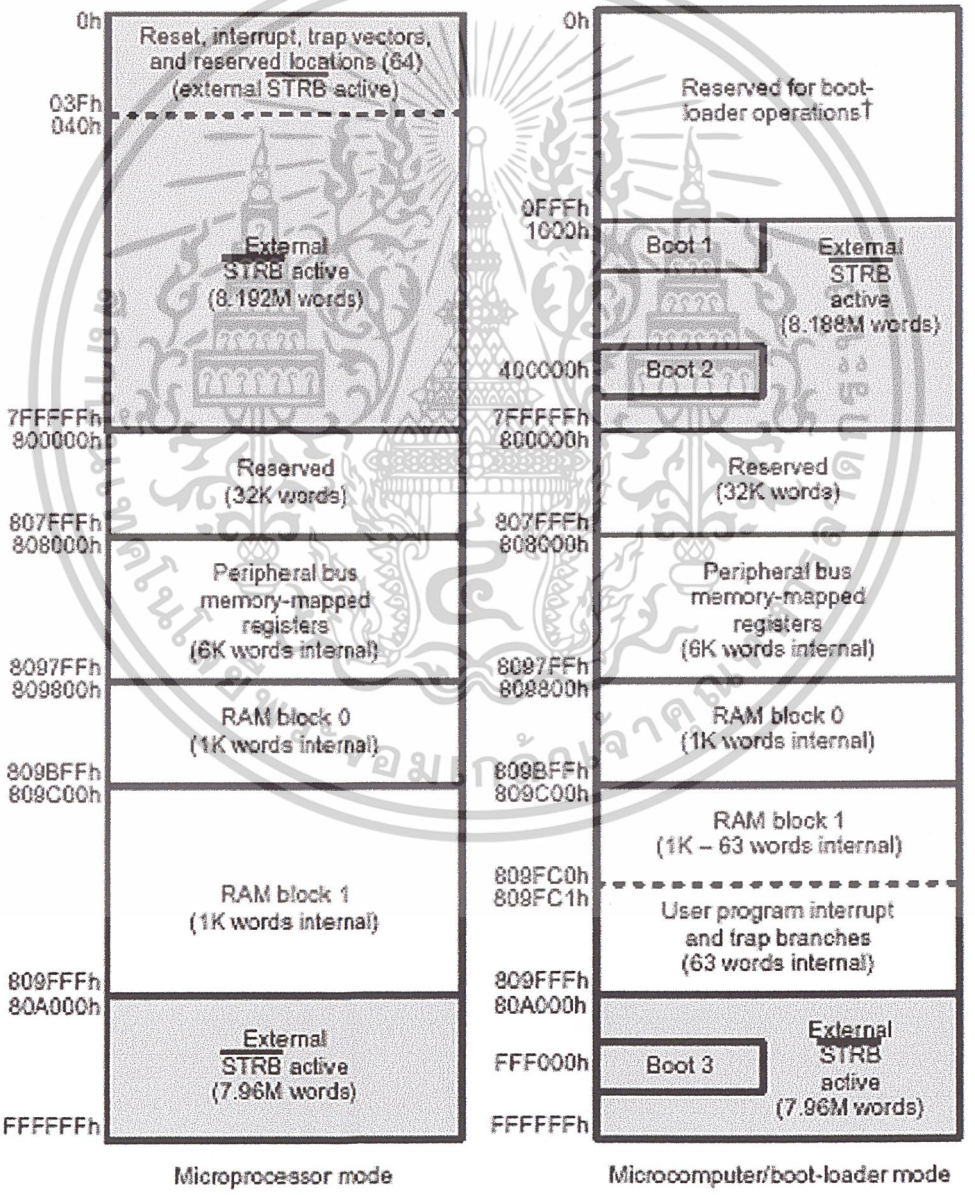
รูปที่ 2.9 ขั้นตอนการพัฒนาโปรแกรมด้วยภาษาซีสำหรับ TMS320C3x/4x

จากรูปที่ 2.9 โปรแกรมกระบวนการทำงานที่เป็นภาษาซี จะถูกแปลงเป็นภาษาแอสแซมบลีด้วย คอมไพลเลอร์ภาษาซี และถูกแปลงเป็นแอสแซมบลีด้วย แอสแซมเบอร์ ต่อจากนั้นจะถูกเชื่อมเข้ากับไลบรารีด้วย Linker เพื่อเป็นไฟล์ COFF ที่ตัวประมวลผลสัญญาณดิจิตอลสามารถเข้าใจได้

2.4.3 การจัดการหน่วยความจำของ TMS320C31

ในรูปที่ 2.10 แสดงการจัดการหน่วยความจำของ TMS320C31 ซึ่งจะเห็นว่า RAM Block 0และ RAM Block 1 แต่ละอันมีขนาด 1 K word (32-bit) ของหน่วยความจำบนชิป อย่างไรก็ตาม ตำแหน่งหน่วยภายในความจำ 256 สุดท้าย จะถูกใช้สำหรับการสื่อสาร Kernel และ Vector

จุดเริ่มต้น RAM Block 0 อยู่ที่ 809800 ในเลขฐานสิบหก ซึ่งเป็นครึ่งหนึ่งของที่ว่างในหน่วยความจำที่อ้างอิงได้ขนาด 2^{24} หรือ 16 million words



รูปที่ 2.10 การจัดการหน่วยความจำของ TMS320C31

2.4.4 การควบคุม AIC

secondary DX serial communication protocol

x x | ← to TA register → | x x | ← to RA register → | 0 0	d13 and d6 are MSBs (unsigned binary)
x | ← to TA' register → | x | ← to RA' register → | 0 1	d14 and d7 are 2's complement sign bits
x | ← to TB register → | x | ← to RB register → | 1 0	d14 and d7 are MSBs (unsigned binary)
x x x x x x x x d7 d6 d5 d4 d3 d2 1 1 Control register	d2 = 0/1 deletes/inserts the bandpass filter d3 = 0/1 disables/enables the loopback function d4 = 0/1 disables/enables the AUX IN+ and AUX IN− terminals d5 = 0/1 asynchronous/synchronous transmit receive sections d6 = 0/1 gain control bits (see gain control section) d7 = 0/1 gain control bits (see gain control section)

รูปที่ 2.11 AIC secondary communication protocol

การส่งผ่านข้อมูลเกิดขึ้นโดยผ่านทาง รีจิสเตอร์รับข้อมูล (Data Receive Register, DR) และรีจิสเตอร์ส่งข้อมูล (Data Transmit Register, DX) ซึ่ง AIC จะถูกควบคุมโดย DX รีจิสเตอร์ และ เมื่อ 2 LSB มีค่าเท่ากับ 1 เกิด Secondary Communication ซึ่งจะควบคุมฟังก์ชันการทำงานดังรูปที่2.11

2.4.5 การคำนวณค่าอัตราความถี่สุ่ม(Sampling Rate) และ แบนวิดธ์(BW) ของตัวกรอง

เราสามารถกำหนดแบนวิดธ์ได้โดยรีจิสเตอร์ TA จากสมการ

$$BW = 3600(New\ SCF/Set\ SCF) \quad (2.9)$$

และ

$$TA = MCLK/(2\times SCF) \quad (2.10)$$

และกำหนดอัตราความถี่สุ่ม (F_s) ได้โดยสมการ

$$TB = MCLK/(2\times TA\times F_s) \quad (2.11)$$

โดย BW = แบนวิดธ์(BW) ของตัวกรอง

SCF = ความถี่ Switching Capacitor

MCLK = ความถี่สัญญาณนาฬิกาหลัก (Master Clock)

TA = รีจิสเตอร์ TA

TB = รีจิสเตอร์ TB

2.5 อินเทอร์รัพท์ (Interrupt)

ตัวประมวลผล 'C3x มีการสนับสนุนอินเทอร์รัพท์หลายตัว ซึ่งนำไปใช้งานในหลายรูปแบบ อินเทอร์รัพท์ภายในถูกสร้างโดย ตัวควบคุม DMA, timer และ พอร์ตอนุกรม อินเทอร์รัพท์ภายนอกได้แก่ INT0-INT3 ซึ่งอินเทอร์รัพท์แต่ละรูปแบบนั้นมีการจัดการแบบอัตโนมัติ ยอมให้เกิดขึ้นพร้อมกันได้

2.5.1 ตาราง อินเทอร์รัพท์ เวคเตอร์ ของ TMS320C31

รูปที่ 2.12 และ รูปที่ 2.13 แสดง อินเทอร์รัพท์ เวคเตอร์ ใน Microprocessor mode และ Microcomputer mode ตามลำดับ ซึ่งอินเทอร์รัพท์ เวคเตอร์ จะเก็บค่าแอดเดรส ของ Interrupt service routine ซึ่งจะเริ่มทำงานเมื่อเกิดอินเทอร์รัพท์ขึ้น ในทางกลับกัน ใน Microcomputer/ Boot-loader mode อินเทอร์รัพท์ เวคเตอร์จะเก็บค่า Branch Instruction ไว้เพื่อเริ่ม Interrupt service routine

Address	Name	Function
00h	RESET	External reset signal input
01h	INT0	External interrupt on the $\overline{INT0}$ pin
02h	INT1	External interrupt on the $\overline{INT1}$ pin
03h	INT2	External interrupt on the $\overline{INT2}$ pin
04h	INT3	External interrupt on the $\overline{INT3}$ pin
05h	XINT0	Internal interrupt generated when serial port 0 transmit buffer is empty
06h	RINT0	Internal interrupt generated when serial port 0 transmit buffer is full
07h	XINT1†	Internal interrupt generated when serial port 1 transmit buffer is empty
08h	RINT1†	Internal interrupt generated when serial port 1 transmit buffer is full
09h	TINT0	Internal interrupt generated by timer 0
0Ah	TINT1	Internal interrupt generated by timer 1
0Bh	DINT	Internal interrupt generated by DMA controller
0Ch	Reserved	
•	•	
•	•	
•	•	
1Fh	Reserved	
20h	TRAP 0	Internal interrupt generated by TRAP 0 instruction
	•	
	•	
	•	
3Bh	TRAP 27	Internal interrupt generated by TRAP 27 instruction
3Ch	TRAP 28 (reserved)	
3Dh	TRAP 29 (reserved)	
3Eh	TRAP 30 (reserved)	
3Fh	TRAP 31 (reserved)	

† Reserved on 'C31

รูปที่ 2.12 ตำแหน่ง Reset, Interrupt และ Trap-Vector สำหรับ TMS320C31 ใน Microprocessor mode

Address	Name	Function
809FC1	INT0	External reset signal input
809FC2	INT1	External interrupt on the $\overline{\text{INT0}}$ pin
809FC3	INT2	External interrupt on the $\overline{\text{INT1}}$ pin
809FC4	INT3	External interrupt on the $\overline{\text{INT2}}$ pin
809FC5	XINT0	External interrupt on the $\overline{\text{INT3}}$ pin
809FC6	RINT0	Internal interrupt generated when serial port 0 transmit buffer is empty
809FC7	XINT1 (Reserved)	
809FC8	RINT1 (Reserved)	
809FC9	TINT0	Internal interrupt generated by timer0
809FCA	TINT1	Internal interrupt generated by timer1
809FCB	DINT	Internal interrupt generated by DMA controller
809FCC–809FDF	Reserved	
809FE0	TRAP0	Internal interrupt generated by TRAP 0 instruction
809FE1	TRAP1	Internal interrupt generated by TRAP 1 instruction
•	•	
•	•	
•	•	
809FFB	TRAP27	Internal interrupt generated by TRAP 27 instruction
809FFC–809FFF	Reserved	

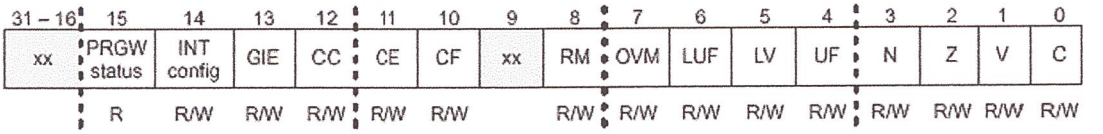
รูปที่ 2.13 ตำแหน่ง Reset, Interrupt และ Trap-Vector สำหรับ TMS320C31 ใน Microcomputer Boot mode

2.5.2 บิตควบคุมอินเทอร์รัพท์ในซีพียู (CPU Interrupt control Bits)

รีจิสเตอร์ 3 ตัว ซึ่งบรรจุบิตควบคุมอินเทอร์รัพท์ ได้แก่

รีจิสเตอร์ สถานะ (ST)

CPU global interrupt-enable bit (GIE) ถูกวางตำแหน่งใน CPU Status รีจิสเตอร์ (ST) ซึ่งควบคุม maskable CPU อินเทอร์รัพท์ทั้งหมด เมื่อบิตนี้ถูกเคลียร์เป็น 0 ก็จะเป็นการปิด(Disable) CPU อินเทอร์รัพท์ทั้งหมด



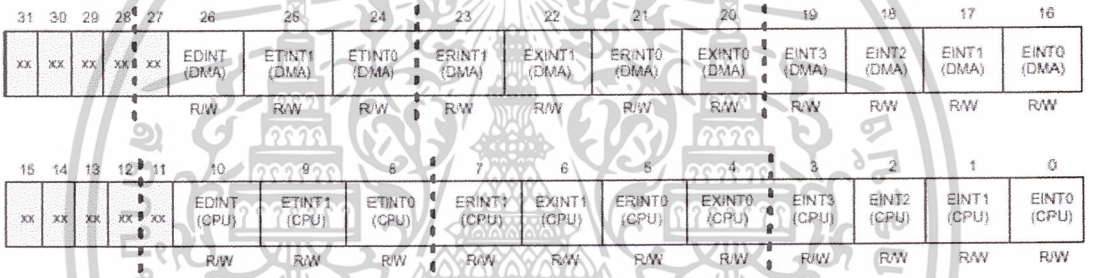
31 – 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xx	PRGW status	INT config	GIE	CC	CE	CF	xx	RM	OVM	LUF	LV	UF	N	Z	V	C
	R	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Notes: 1) xx = reserved bit, read as 0
2) R = read, W = write

รูปที่ 2.14 รีจิสเตอร์สถานะ(Status Register, ST)

รีจิสเตอร์เปิด CPU/DMA อินเทอร์รัพท์(Interrupt-Enable Register,IE)

รีจิสเตอร์นี้ จะแยก เปิด/ปิด CPU, ภายนอก DMA, พอร์ตอนุกรม และ Timer อินเทอร์รัพท์



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
xx	xx	xx	xx	xx	EDINT (DMA)	ETINT1 (DMA)	ETINT0 (DMA)	ERINT1 (DMA)	EXINT1 (DMA)	ERINT0 (DMA)	EXINT0 (DMA)	EINT3 (DMA)	EINT2 (DMA)	EINT1 (DMA)	EINT0 (DMA)
					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

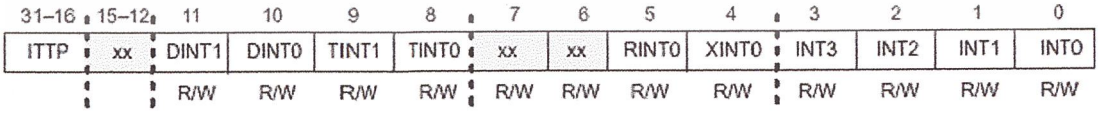
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xx	xx	xx	xx	xx	EDINT (CPU)	ETINT1 (CPU)	ETINT0 (CPU)	ERINT1 (CPU)	EXINT1 (CPU)	ERINT0 (CPU)	EXINT0 (CPU)	EINT3 (CPU)	EINT2 (CPU)	EINT1 (CPU)	EINT0 (CPU)
					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Notes: 1) xx = reserved bit, read as 0
2) R = read, W = write

รูปที่ 2.15 รีจิสเตอร์เปิด CPU/DMA อินเทอร์รัพท์ (Interrupt-Enable Register,IE)

รีจิสเตอร์ CPU อินเทอร์รัพท์แฟลก (CPU interrupt flag register, IF)

รีจิสเตอร์นี้จะเก็บค่าอินเทอร์รัพท์แฟลกบิต ซึ่งชี้ว่าอินเทอร์รัพท์ที่เกี่ยวของได้ถูกเซ็ต



31–16	15–12	11	10	9	8	7	6	5	4	3	2	1	0
ITTP	xx	DINT1	DINT0	TINT1	TINT0	xx	xx	RINT0	XINT0	INT3	INT2	INT1	INT0
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Notes: 1) xx = reserved bit, read as 0
2) R = read, W = write

รูปที่ 2.16 รีจิสเตอร์ CPU อินเทอร์รัพท์แฟลก (CPU interrupt flag register, IF)

2.5.3 การจัดการอินเทอร์รัพท์ ในภาษาซี ของ TMS320C3x

อินเทอร์รัพท์ซีสามารถจัดการได้โดยตรงโดยฟังก์ชันในภาษาซี โดยใช้การกำหนดชื่อดังนี้

c_int*nn*

โดย *nn* เป็นเลขอินเทอร์รัพท์สองหลัก

เช่น c_int04 คือ External Interrupt 3, INT3

c_int10 คือ Timer Interrupt 0, TINT0

2.6 ทางเดินข้อมูลหลัก (Primary Bus)

ใน TMS320C31 ได้ให้การเชื่อมต่อข้อมูลจากภายนอก คือ ทางเดินข้อมูลหลัก (Primary Bus)ซึ่งเป็นแบบ 32 บิตข้อมูล และ 24 บิตแอสเดรส เอาไว้เชื่อมต่อกับ หน่วยความจำภายนอก และอุปกรณ์อื่นๆ ซึ่งจะมีสัญญาณอื่นๆ ที่เกี่ยวข้อง ซึ่งมีสถานะดัง รูปที่ 2.17

Signal	Type†	Description	Value After Reset	Idle Status
$\overline{\text{STRB}}$	O/Z	Primary interface access strobe	1	1
R/$\overline{\text{W}}$	O/Z	Specifies memory read (active high) or write (active low) mode	1	1
$\overline{\text{HOLD}}$	I	Hold external memory interface	NA‡	Ignored
$\overline{\text{HOLDA}}$	O/Z	Hold acknowledge for external memory interface	1	1
$\overline{\text{RDY}}$	I	Indicates external primary interface is ready to be accessed	NA‡	Ignored
A (23–0)	O/Z	Primary address bus. When the primary bus address lines are not in high-impedance state due to $\overline{\text{HOLD}}$ signal, they keep in the last external primary bus access.	HI	Address of last external bus access
D (31–0)	I/O/Z	Primary data bus. These signals go to high-impedance between write accesses.	HIZ	HIZ

† I Input
O Output
Z High impedance

‡ NA means not affected.

รูปที่ 2.17 สัญญาณที่เกี่ยวข้องในการเชื่อมต่อกับ Primary Bus

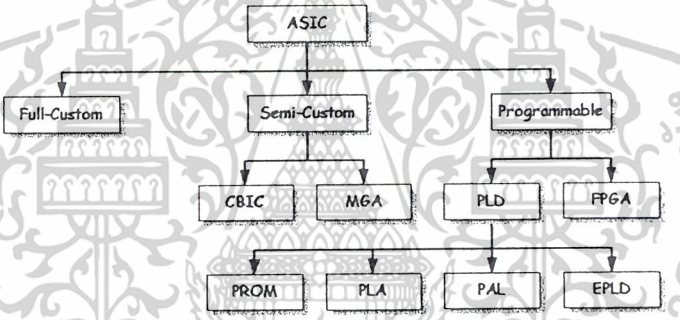
2.7 ทฤษฎีบอร์ด FPGA

ปัจจุบันการออกแบบวงจรดิจิตอลที่มีความซับซ้อนและมีพฤติกรรมเจาะจงเฉพาะงานนั้นๆแล้ว เรานิยมใช้อุปกรณ์ที่สามารถโปรแกรมพฤติกรรมของวงจรลงไปได้เพื่อลดขนาดและความยุ่งยากของการต่อวงจรและหลีกเลี่ยงสัญญาณรบกวนอาจจะที่เกิดขึ้นด้วย ดังนั้นจึงมีการออกแบบวงจรที่ใช้ภาษา

อธิบายพฤติกรรมของวงจร (Hardware Description Language) โดยใช้แผนภาพวงจร (Schematic) ผสมกับภาษาคอมพิวเตอร์แล้วใช้ซอฟต์แวร์สังเคราะห์ลอจิก (Logic Synthesis Tools) แปลงให้เป็นข้อมูลที่สามารถนำไปโปรแกรมลงในชิพไอซีที่สามารถโปรแกรมได้ เพื่อสร้างเป็นวงจรที่ต้องการ โดยไอซีที่สามมารถโปรแกรมได้เหล่านี้มีชื่อเรียกว่า ASIC : Application-Specific Integrated Circuit (ออกเสียงว่า เอ-ซิก) ซึ่งตัวอย่างของ ASIC ได้แก่ชิพไอซีที่ใช้สำหรับตุ๊กตาของเล่นพูดได้ ดาวเทียม และชิพที่ภายในบรรจุด้วยไมโครโปรเซสเซอร์กับอุปกรณ์ทางลอจิกอื่นๆ

2.7.1 ประเภทของ ASIC

ASIC แบ่งเป็น 3 ประเภทใหญ่ๆ คือ Full-custom, Semi-custom และ Programmable ดังรูปที่ 2.18



รูปที่ 2.18 ประเภทของ ASIC

1. Full-custom

ASIC ประเภทนี้ลูกค้าจะเป็นผู้ออกแบบเซลล์ลอจิก (เช่น แอนด์เกท ออร์เกท มัลติเพล็กเซอร์ และฟลิบฟล็อบ) และลักษณะการจัดวางอุปกรณ์บนตัวไอซีรวมถึงหน้ากากสำหรับควบคุมการเจือและสร้างชั้นสาร (Mask) ต่างๆ ที่ใช้ในการทำไอซีเอง ดังนั้นค่าใช้จ่ายในการออกแบบและการผลิตจะสูงมาก

2. Semi-custom

ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบเอาไว้ก่อนแล้วในรูปแบบของไลบรารีและลูกค้าจะเป็นผู้ออกแบบ Mask ต่างๆ เอง ตัวอย่างของไอซีประเภทนี้ได้แก่ Standard-Cell–Based ASIC และ Masked Gate-Array–Based ASIC

3. Programmable

ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบไว้ก่อนเช่นเดียวกับ Semi-Custom แต่ชั้นของ Mask จะไม่สามารถเปลี่ยนแปลงได้ตามความต้องการของผู้ออกแบบ ไอซีประเภทนี้ยังแบ่งออกเป็น 2 ชนิดคือ Programmable Logic Device (PLD) และ Field-Programmable Gate Array (FPGA)

3.1 Programmable Logic Device (PLD)

มีโครงสร้างภายในเป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่เป็นกลุ่มซึ่งมีทั้งวงจรคอมบิเนชัน (Combination) และซีเควนเชียล (Sequential) สำหรับเทคโนโลยีของวงจรที่ใช้สร้าง PLD จะมีทั้ง TTL, ECL และ CMOS ตามความเหมาะสมของแต่ละระบบ ไอซี PLD ทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่เหมือนกัน โดยมีวงจรคอมบิเนชันที่เป็นผลคูณร่วมบวก (Sum of product) ประกอบไปด้วยชุดของแอนด์เกทต่อร่วมกับออร์เกท และในการโปรแกรมจะเป็นการเลือกว่าอินพุทภายในของแอนด์เกทกับสัญญาณอินพุทใดบ้างที่จะต้องต่อถึงกัน ซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับจากเอาท์พุทภายในเอง เช่น การติดต่ออินพุทของออร์เกทกับเอาท์พุทของแอนด์เกทตัวต่างๆ สำหรับการโปรแกรมทางกายภาพนั้นอินพุทต่างๆ ของอุปกรณ์ ทุกตัวจะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดก็จะตัดฟิวส์ตัวนั้นทิ้ง ทำให้สามารถโปรแกรมได้เพียงครั้งเดียว ไอซี PLD บางชนิดใช้มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดยใช้กระแสไฟฟ้า และสามารถลบแล้วโปรแกรมเข้าไปใหม่ได้อีก สำหรับไอซีในตระกูล PLD ได้แก่ PROM, PAL, PLA และ EPLD

3.2 Field-Programmable Gate Array (FPGA)

เป็นอุปกรณ์ที่มีความซับซ้อนมากกว่า PLD ไปอีกระดับหนึ่ง ซึ่งในความเป็นจริงแล้ว PLD และ FPGA แตกต่างกันน้อยมาก สำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่างรวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL

โครงสร้างภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกทต่างๆมากมาย ซึ่งในปัจจุบันความจุเกทภายในตัวชิพ FPGA ได้เพิ่มขึ้นจากระดับไม่กี่พันตัวจนถึงระดับล้านตัว ซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่าย ซึ่งในปัจจุบันการออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อยๆ

2.7.2 การโปรแกรมบนบอร์ด FPGA

ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างกันไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรมซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรมโดยการใช้หน่วยความจำ

1. การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

1.1 Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

1.2 Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรมแล้ว จุดเชื่อมต่อจะเชื่อมถึงกัน

2. การโปรแกรมโดยใช้หน่วยความจำ

2.1 EEPROM Based FPGA

FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจุของเกทต่ำ โดยทั่วไปจะน้อยกว่า 20,000 เกท แต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในการโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิท ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

2.2 SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM) ทำให้สามารถโปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 – 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการโปรแกรมน้อย (ระดับ nsec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

2.7.3 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ ในกรณีของการออกแบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา

แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี สำหรับภาษาที่ใช้สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น

ในขั้นตอนการออกแบบวงจรด้วย FPGA จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรมของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆ จึงทำให้ผลที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดลแต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำการสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไป

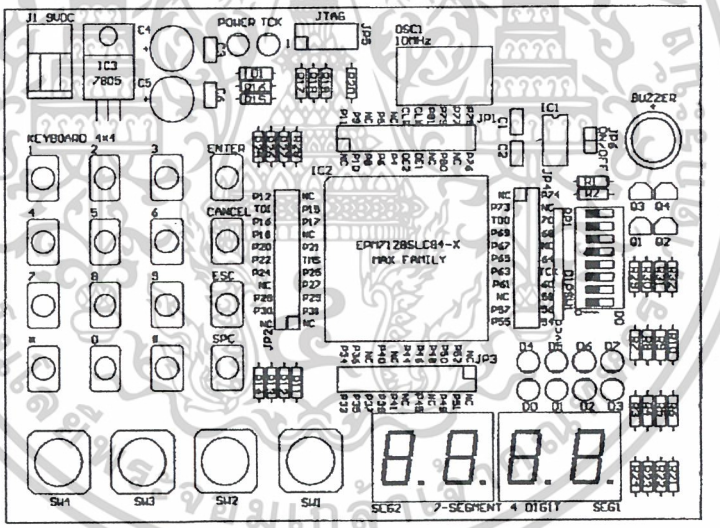
นั้นเป็นอย่างไร เช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้นตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตามที่กำหนด

2.8 บอร์ดทดลอง PLD – A01

PLD-A01 เป็นบอร์ดทดลองที่ถูกออกแบบมาเพื่ออำนวยความสะดวกในการเรียนรู้และใช้งาน FPGA ของบริษัท ALTERA ในตระกูล MAX7000S อนุกรม EPM7128SLC84 โดยใช้งานร่วมกับโปรแกรม MAX+PLUS II ในการออกแบบและพัฒนาระบบทางดิจิตอลระดับสูง

2.8.1 โครงสร้างและองค์ประกอบของบอร์ด PLD - A01

ส่วนประกอบต่างๆของบอร์ด PLD - A01 เป็นดังรูปที่ 2.19



รูปที่ 2.19 บอร์ด PLD - A01/1

อุปกรณ์ต่างๆบนบอร์ด PLD - A01 มีดังต่อไปนี้

- วงจรรวมตระกูล MAX7000S ในอนุกรม EPM7128SLC84
- JTAG CONNECTOR
- พอร์ตขยายช่องสัญญาณ
- สวิตช์ กดติด - ปล่อยดับ 4 ตัว
- สวิตช์เมตริกซ์ขนาด 4 x 4
- ดิพสวิตช์ 8 หลัก 1 ชุด

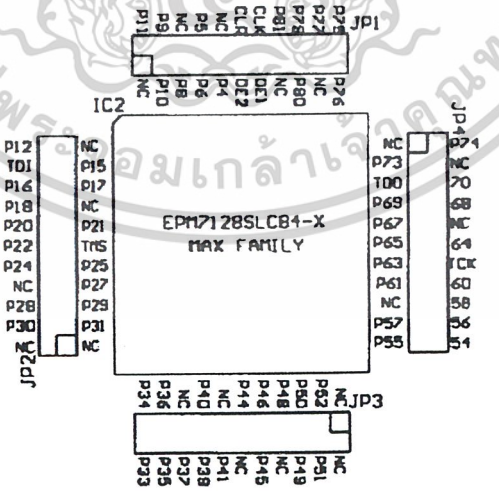
- ไดโอดเปล่งแสง 8 ดวง
- 7-Segment 4 หลักชนิด Common Anode แบบ Multiplex
- BUZZER 1 ชุด
- OSCILLATOR ความถี่ 10.000 MHz
- DC INPUT แรงดัน 7-10 Volt

2.8.2 วงจรรวมตระกูล MAX7000S ในอนุกรม EPM7128SLC84

เป็นวงจรรวมที่มีความจุปานกลาง ประมาณ 2,500 เกต โดยมีโครงสร้างภายในเป็นแบบ EEPROM BASE FPGA นอกจากนี้สามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง (Non-Volatile Configuration) ซึ่งจะใช้ AND-OR Plane ในการทำลอจิกฟังก์ชัน และมักมีการจัดสถาปัตยกรรมในรูปแบบอะเรย์ (Array Structure) ส่วนการโปรแกรมสามารถทำซ้ำได้ประมาน 10,000 ครั้ง ภายใน EPM7128SLC84 จะประกอบด้วย Macrocell จำนวน 128 ตัว ในแต่ละ Macrocell จะมี Programmable-AND/Fixed-OR Array, Configurable Register with Independently Programmable clock, Clock Enable, Clear และ Preset Functions

1. พอร์ตขยายช่องสัญญาณ

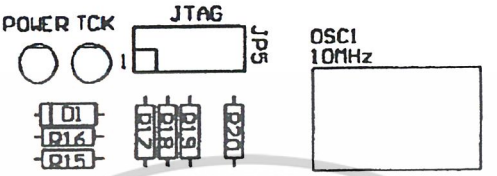
เป็น Header คู่ตัวเมียสำหรับขยายการต่อช่องสัญญาณ ตำแหน่งขาของชิพ ไอซี EPM7128SLC84 กับ Hole จะมีความสัมพันธ์ดังรูปที่ 2.20



รูปที่ 2.20 การจัดวางตำแหน่งของชิพไอซี กับ Header

2. JTAG Connector

ใช้สำหรับต่อสาย Byte Blaster เพื่อ Download ข้อมูลของวงจรลอจิกจาก Computer ลงในชิพไอซี ซึ่งจะมีตำแหน่งขาและการจัดวางดังรูปที่ 2.21



รูปที่ 2.21 ตำแหน่งขาและการจัดวาง JTAG Connection

3. สวิตช์กดติด -ปล่อยดับ 4 ตัว

สวิตช์ SW1-SW4 เป็นสวิตช์แบบกดติด - ปล่อยดับ และมีตัวต้านทาน PULL-UP 10k ต่ออยู่ เมื่อกดสวิตช์จะได้สัญญาณลอจิกศูนย์ โดยที่

SW1 ต่อกับขา 49 ของ EPM7128SLC84

SW2 ต่อกับขา 50 ของ EPM7128SLC84

SW3 ต่อกับขา 51 ของ EPM7128SLC84

SW4 ต่อกับขา 52 ของ EPM7128SLC84

4. สวิตช์เมตริกช์ขนาด 4x4

เป็นสวิตช์เมตริกช์ ขนาด 16 ปุ่ม ประกอบด้วย 4 Row กับ 4 Column ในหลักของ Column จะมีตัวต้านทาน PULL UP 10k ต่ออยู่และมีลักษณะการจัดวางวงจรดังรูปที่ 2.22 โดยที่

Column ที่ 1 มีตัวต้านทาน pull up 10 k ต่อกับขา 28 ของ EPM7128SLC84

Column ที่ 2 มีตัวต้านทาน pull up 10 k ต่อกับขา 29 ของ EPM7128SLC84

Column ที่ 3 มีตัวต้านทาน pull up 10 k ต่อกับขา 30 ของ EPM7128SLC84

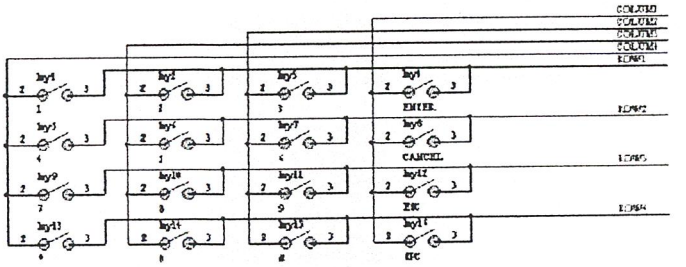
Column ที่ 4 มีตัวต้านทาน pull up 10 k ต่อกับขา 31 ของ EPM7128SLC84

Row ที่ 1 ต่อกับขา 22 ของ EPM7128SLC84

Row ที่ 2 ต่อกับขา 24 ของ EPM7128SLC84

Row ที่ 3 ต่อกับขา 25 ของ EPM7128SLC84

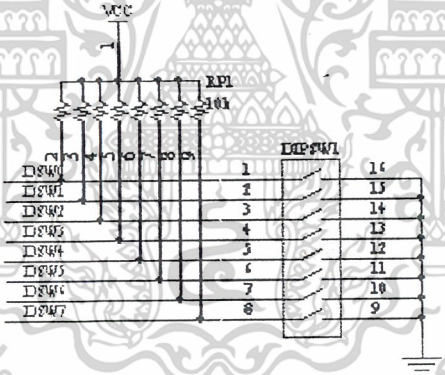
Row ที่ 4 ต่อกับขา 27 ของ EPM7128SLC84



รูปที่ 2.22 วงจรสวิตช์เมตริกช์ขนาด 4*4

5. ดิพสวิตช์ 8 หลัก

เป็นดิพสวิตช์ขนาด 8 ช่อง (Octal Dip Switches) ต่ออยู่กับตัวต้านทาน PULL UP 10k เมื่อเลื่อนสวิตช์ไปที่ตำแหน่ง ON จะได้สัญญาณลอจิกศูนย์ และเมื่อเลื่อนสวิตช์กลับมาที่ตำแหน่ง OFF จะได้สัญญาณลอจิกหนึ่งมีการต่อวงจรดังรูปที่ 2.23



รูปที่ 2.23 การต่อวงจรของดิพสวิตช์ 8 ช่อง

โดยที่

DSW0 ต่อกับขา 54 ของ EPM7128SLC84

DSW1 ต่อกับขา 55 ของ EPM7128SLC84

DSW2 ต่อกับขา 56 ของ EPM7128SLC84

DSW3 ต่อกับขา 57 ของ EPM7128SLC84

DSW4 ต่อกับขา 58 ของ EPM7128SLC84

DSW5 ต่อกับขา 60 ของ EPM7128SLC84

DSW6 ต่อกับขา 61 ของ EPM7128SLC84

DSW7 ต่อกับขา 63 ของ EPM7128SLC84

6. ไดโอดเปล่งแสง 8 ตัว

ในบอร์ดทดลองชุดนี้มีไดโอดเปล่งแสง 8 ดวง โดยแต่ละดวงจะมีตัวต้านทาน 1k เป็นตัวจำกัดกระแส เมื่อให้ลอจิกหนึ่งจะทำให้ LED สว่างและเมื่อให้ลอจิกศูนย์จะทำให้ LED ดับ ลักษณะการต่อวงจรเป็นดังรูปที่ 2.24 โดยที่

LED0 ต่อกับขา 21 ของ EPM7128SLC84

LED1 ต่อกับขา 20 ของ EPM7128SLC84

LED2 ต่อกับขา 18 ของ EPM7128SLC84

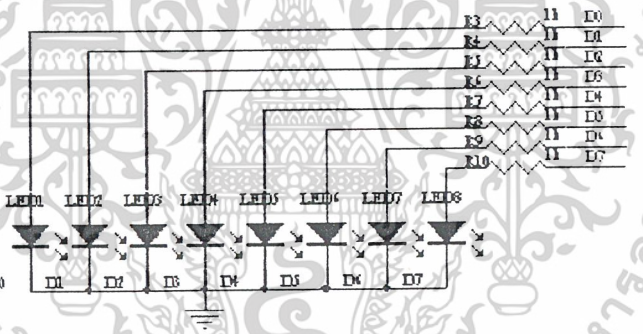
LED3 ต่อกับขา 17 ของ EPM7128SLC84

LED4 ต่อกับขา 16 ของ EPM7128SLC84

LED5 ต่อกับขา 15 ของ EPM7128SLC84

LED6 ต่อกับขา 12 ของ EPM7128SLC84

LED7 ต่อกับขา 11 ของ EPM7128SLC84



รูปที่ 2.24 การต่อวงจรสำหรับ LED 8 ดวง

7. 7-Segment 4 หลักชนิด Common Anode แบบ Multiplex

SEG1 และ SEG2 เป็น LED 7-Segment ตัวละ 2 หลัก แบบคอมมอนแอโนด แต่ละหลักจะต่อเป็นแบบ Multiplexor สำหรับขาคอมมอนจะมีตัวต้านทาน 330 โอห์มต่ออนุกรมอยู่เพื่อจำกัดกระแสให้กับ LED หากต้องการให้ Segment a ของหลักที่หนึ่งสว่าง จะต้องให้ a เป็นลอจิกศูนย์ และเนื่องจากขาคอมมอนต่อกับทรานซิสเตอร์ชนิด PNP เพื่อที่จะทำให้ทรานซิสเตอร์นำกระแสจะต้องให้ c1 เป็นลอจิกศูนย์ จะทำให้ Segment A หลักที่หนึ่งสว่างขึ้นและจะมีตำแหน่งขาดังนี้

Segment a ต่อกับขา 33 ของ EPM7128SLC84

Segment b ต่อกับขา 34 ของ EPM7128SLC84

Segment c ต่อกับขา 35 ของ EPM7128SLC84

Segment d ต่อกับขา 36 ของ EPM7128SLC84

Segment e ต่อกับขา 37 ของ EPM7128SLC84

Segment f ต่อกับขา 39 ของ EPM7128SLC84

Segment g ต่อกับขา 40 ของ EPM7128SLC84

Segment dot ต่อกับขา 41 ของ EPM7128SLC84

C1 ต่อกับขา 44 ของ EPM7128SLC84

C2 ต่อกับขา 45 ของ EPM7128SLC84

C3 ต่อกับขา 46 ของ EPM7128SLC84

C4 ต่อกับขา 48 ของ EPM7128SLC84

8. BUZZER

ใช้ไอซี NE555 เป็นตัวกำเนิดความถี่เสียงให้แก่ BUZZER โดยใช้จุดต่อ SP ซึ่งต่ออยู่กับขา 10 ของ EPM7128SLC84 เมื่อ SP ได้รับลอจิกหนึ่งจะทำให้ BUZZER มีเสียงดังขึ้นมา และเมื่อให้ SP ได้รับลอจิกศูนย์จะทำให้ BUZZER ไม่ทำงาน ส่วน JP6 ไว้ สำหรับตัด BUZZER ออกจาก NE555 ซึ่งจะทำให้ไม่มีสัญญาณส่งให้ BUZZER เลย หากไม่ต้องการให้ BUZZER มีเสียงดังก็ให้ดึง JP6 ออก

- SP ต่อกับขา 10 ของ EPM7128SLC84

9. OSCILLATOR

บอร์ดทดลองชุดนี้ใช้ OSILLATOR แบบโมดูลออสซิเลเตอร์ สำหรับผลิตความถี่ 10.000 MHz ป้อนให้กับขา 83 (GCLK) ของ EPM7128SLC84

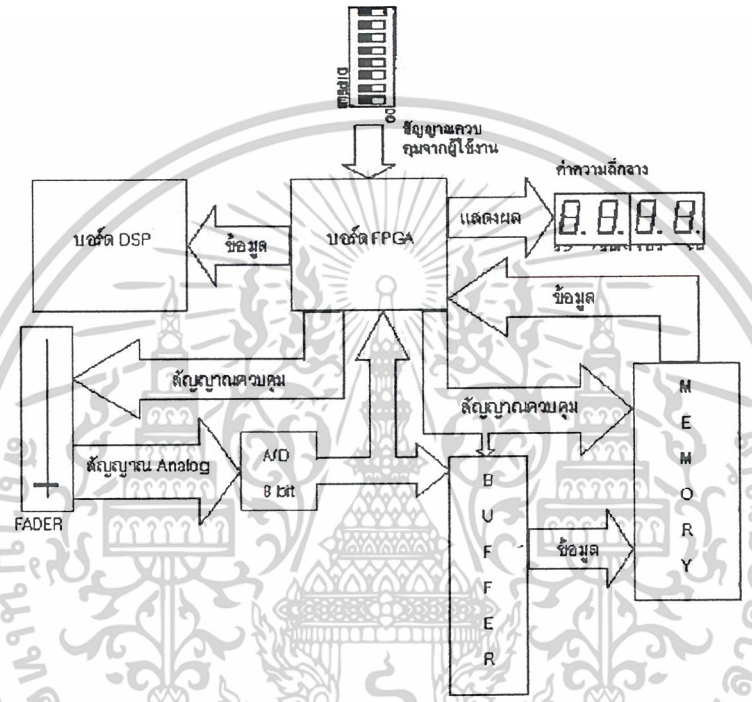
- GCLK ต่ออยู่กับขา 83 ของ EPM7128SLC84

10. DC INPUT แรงดัน 7-10 Volt

เป็นแจ็คอแด็ปเตอร์ตัวเมียแกนเล็กสำหรับรับแรงดันไฟตรง 7 ถึง 10 โวลท์ ซึ่งภายในบอร์ดทดลองจะมีไอซี 7805 เร็คกูเรเตอร์แรงดันคงที่ 5 โวลท์ โดยที่แกนในจะเป็นขั้วบวก และแกนนอกเป็นขั้วลบ

บทที่ 3
การออกแบบ

3.1 การออกแบบ



รูปที่ 3.1 แสดงโครงสร้างของวงจรทั้งหมด

สำหรับโครงงานนี้จะออกแบบ Equalizer ที่มีตัวกรองpeak filter 6ตัว ทำการปรับขาดของผลตอบสนองความถี่ได้ 6ความถี่ โดยจะทำที่ความถี่ 80, 160, 320, 640, 1280, และ2560 Hz โดยเราจะนำค่าความถี่ต่างๆแทนลงใน fc แล้วนะค่าสัมประสิทธิ์ที่ได้ไปแทนลงในสมการผลต่างสืบเนื่อง ดังตัวอย่างที่ 1

<u>ตัวอย่างที่ 1</u> ค่า fc = 1280 Hz, Q_∞= 5, V0=10 และค่าความถี่สุ่มเท่ากับ 7891.41 Hz

นำค่าต่างๆแทนลงในสมการ

$K= \tan(2\pi \times fc)/(2 \times Fs))$

$a0=(1+(V0 \times K/Q)+K^2)/(1+(K/Q)+K^2)$

$a1=2 \times (K^2-1)/(1+(K/Q)+K^2)$

$a2=(1-(V0 \times K/Q)+K^2)/(1+(K/Q)+K^2)$

b1=a1

$b2=(1-(K/Q)+K^2)/(1+(K/Q)+K^2)$

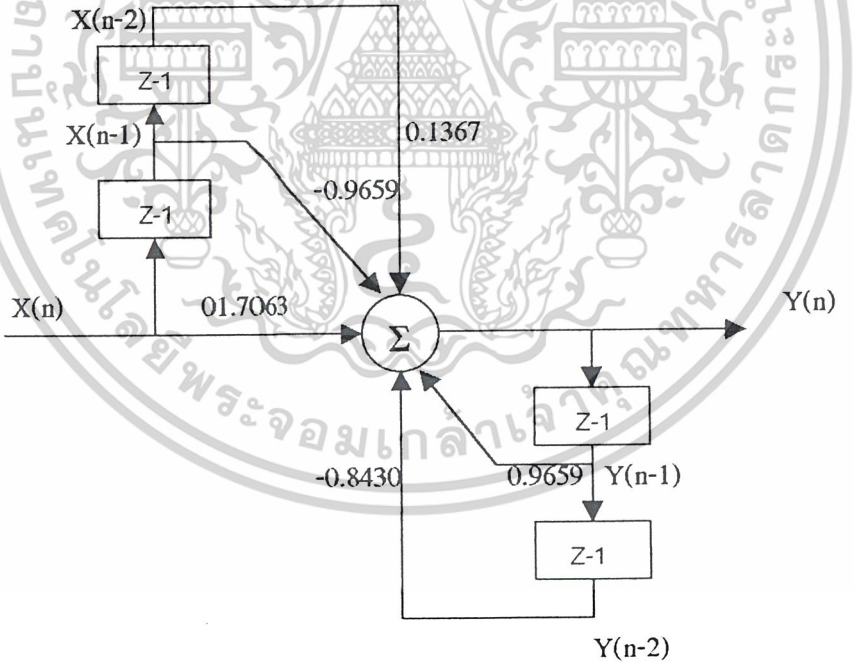
เราจะได้สัมประสิทธิ์ตัวกรอง

a0=1.7063, a1= -0.9659, a2=0.1367, b1= -0.9659, b2=0.8430 .

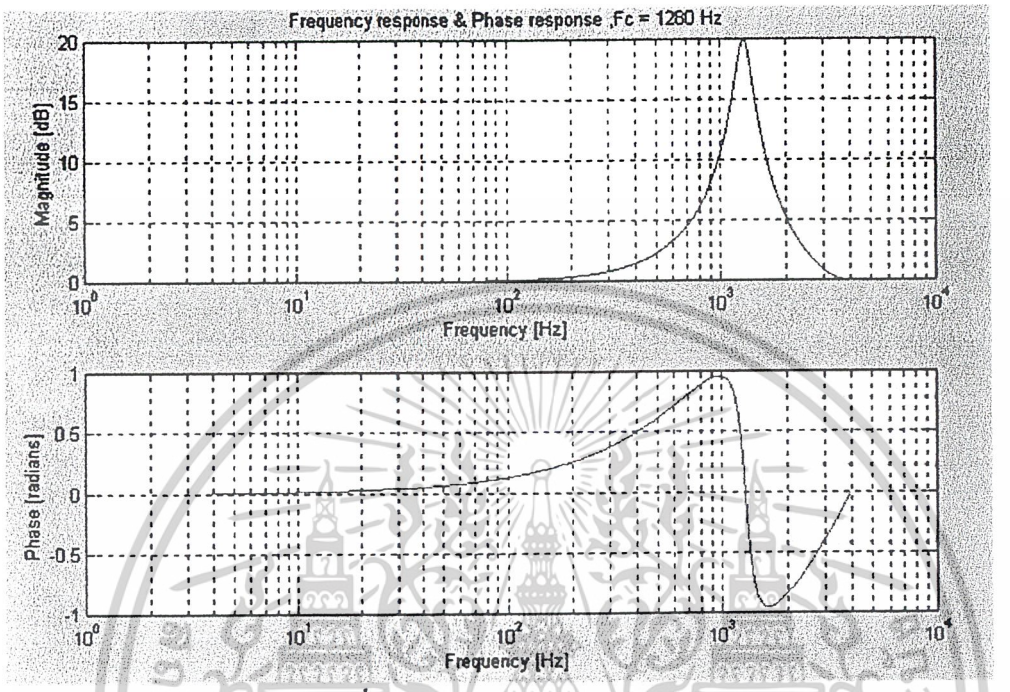
นำมาสร้างเป็นสมการผลต่างสืบเนื่อง

$$Y(n) = a0X(n) + a1X(n)Z^{-1} + a2X(n)Z^{-2} - b1Y(n)Z^{-1} - b2Y(n)Z^{-2}$$

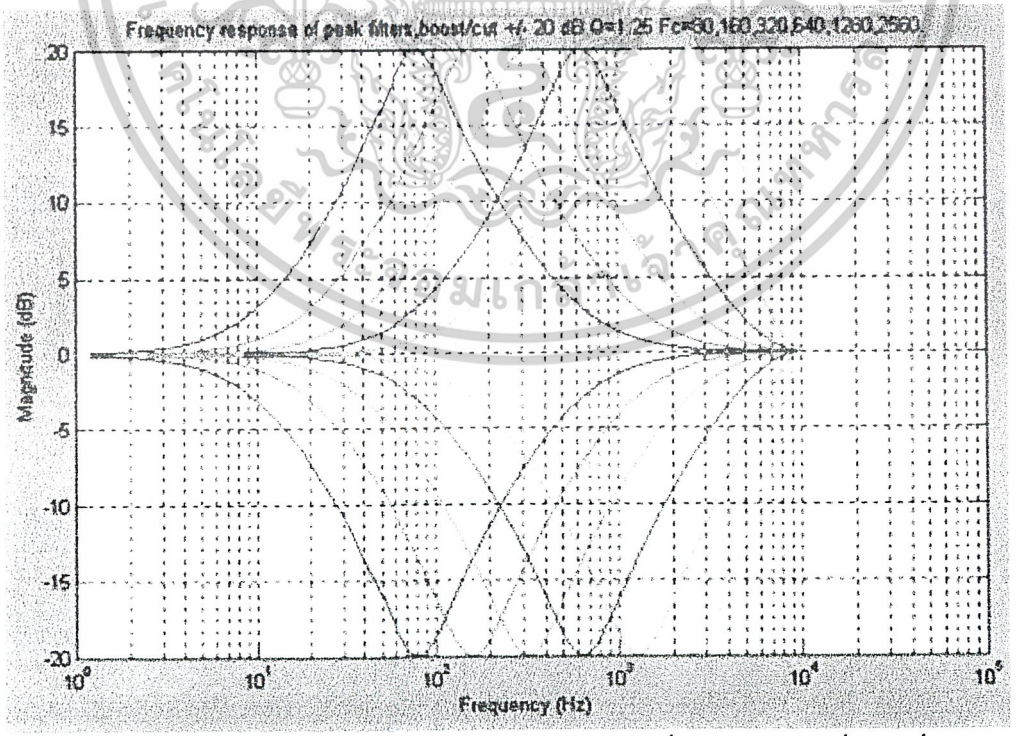
เขียนเป็นโครงสร้างตัวกรองโดยตรง 1 ดังรูปที่ 3.1 และนำมาประมวลผลโดยใช้โปรแกรม Matlab ได้ดังรูปที่ 3.2 รูปที่ 3.3, 3.4 และรูปที่3.5 จะแสดงการประมวลผลด้วยโปรแกรมMatlabโดยที่ รูปที่3.3 แสดงEqualizer ในกรณี เพิ่มและ ลดทอนที่ความถี่ต่างๆกันไป รูปที่3.4 คือจุดของ Pole และZeroของตัวกรองในระนาบแกนZที่ความถี่ต่างๆในกรณีเพิ่ม รูปที่3.5 คือจุดของ Poleและ Zero ของตัวกรองในระนาบแกนZที่ความถี่ต่างๆในกรณีลดทอน



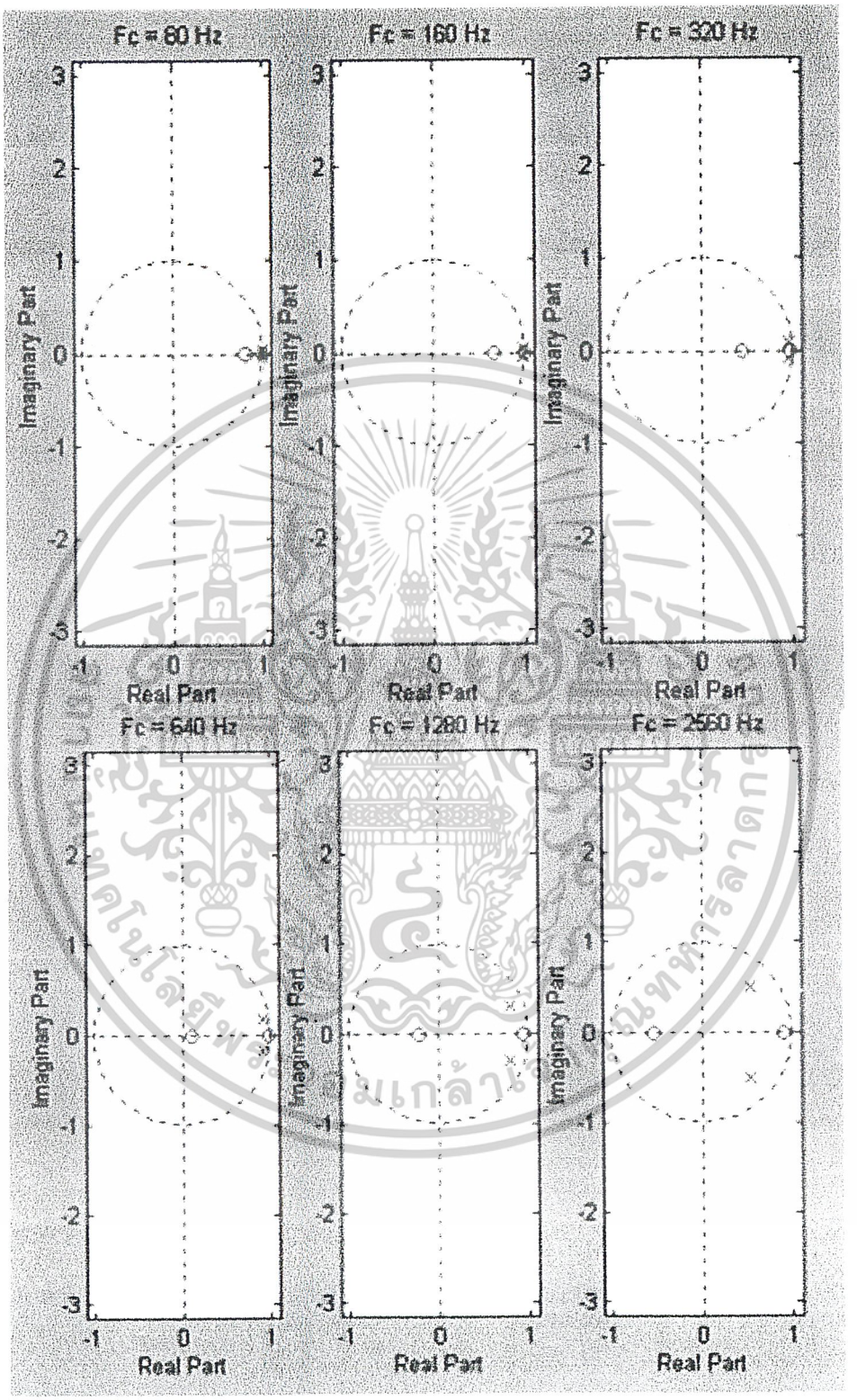
รูปที่ 3.2 โครงสร้างตัวกรองแบบตรง 1 จากตัวอย่างที่ 1



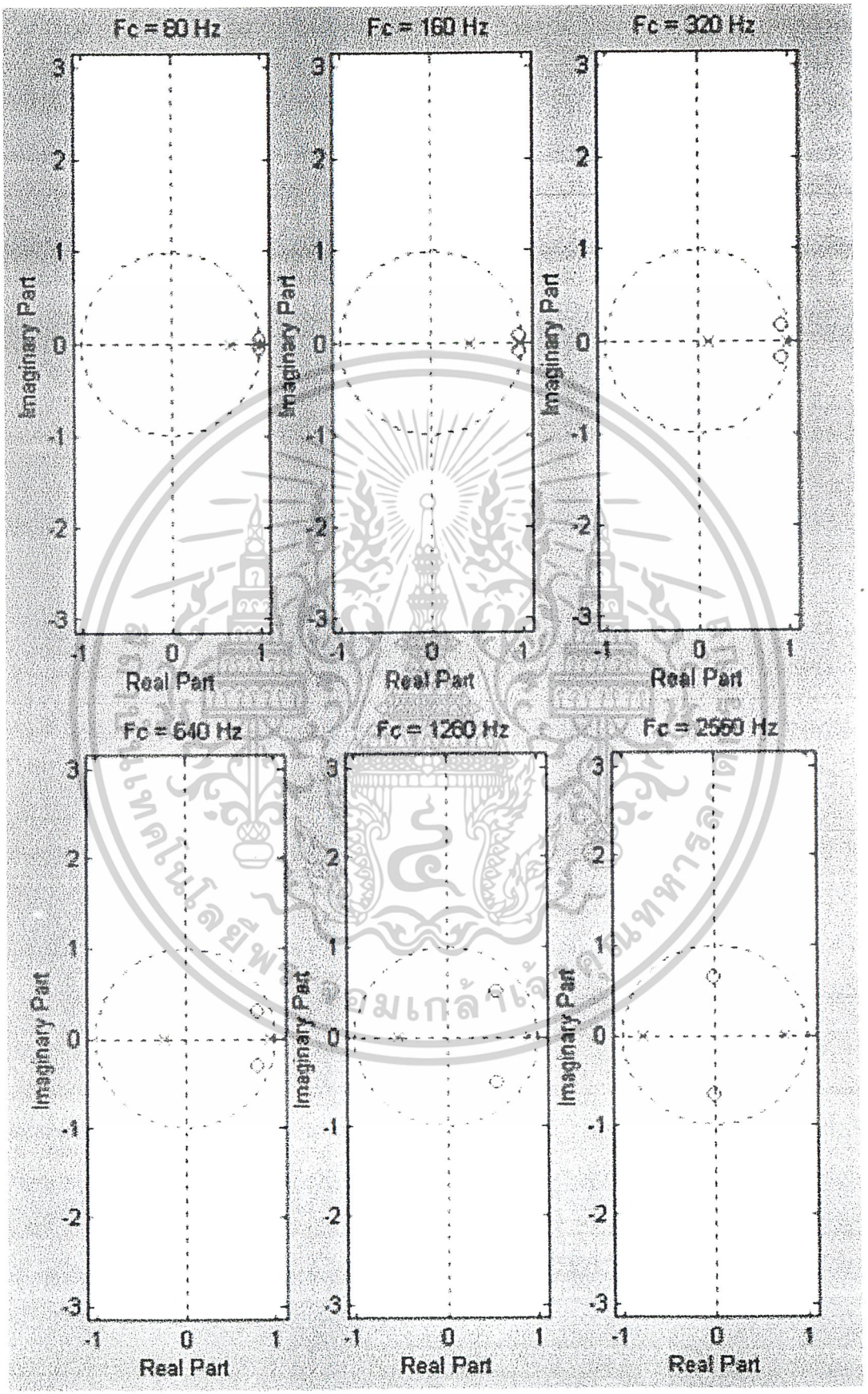
รูปที่ 3.3 รูปการประมวลผลโดยใช้ Matlab



รูปที่ 3.4 แสดงผลตอบสนองของEqualizer ในกรณี เพิ่มและลดทอนที่ความถี่ต่าง ๆ



รูปที่3.5 จุดของ PoleและZeroในกรณีเพิ่ม 20 dB fc = 80, 160, 320, 640, 1280, 2560Hz Q = 5



รูปที่3.6 จุดของ Poleและ Zeroในกรณีลดทอน 20 dB fc = 80, 160, 320, 640, 1280, 2560Hz,Q = 5

3.2 การตั้งค่าอัตราความถี่สุ่มการตัวอย่างและแบนวิดธ์ของ AIC

เนื่องจากเราต้องการค่าอัตราความถี่สุ่มตัวอย่างที่ 8 kHz ดังนั้นเพื่อป้องกันความถี่เทียม (Aliasing frequency) และมีการเผื่อค่าไว้เล็กน้อย จึงกำหนดค่าแบนวิดธ์ที่ 3.6 kHz ได้จากสมการที่ 2.9 และ 2.10

SCF = 3600(288K)/3600

= 288 kHz

TA = 6.25 MHz/ (2×288K)

= 10.85 ≈ 11

คำนวณรีจิสเตอร์ TB จากค่าอัตราความถี่สุ่ม(F_s) ได้จากสมการที่ 2.11

TB = 6.25 MHz/ (2×11×8,000)

= 35.51 ≈ 36

กลับมาคำนวณค่าที่แท้จริงของ SCF

SCF = 6.25 MHz/(2×TA)

= 284.09 kHz

คำนวณค่าที่แบนวิดธ์แท้จริง

BW = 3600(289.09 kHz/288 kHz)

= 3551.14 Hz

และคำนวณค่าอัตราความถี่สุ่มที่แท้จริง

Fs = 6.25 MHz/(2×11×36)

= 7891.41 Hz

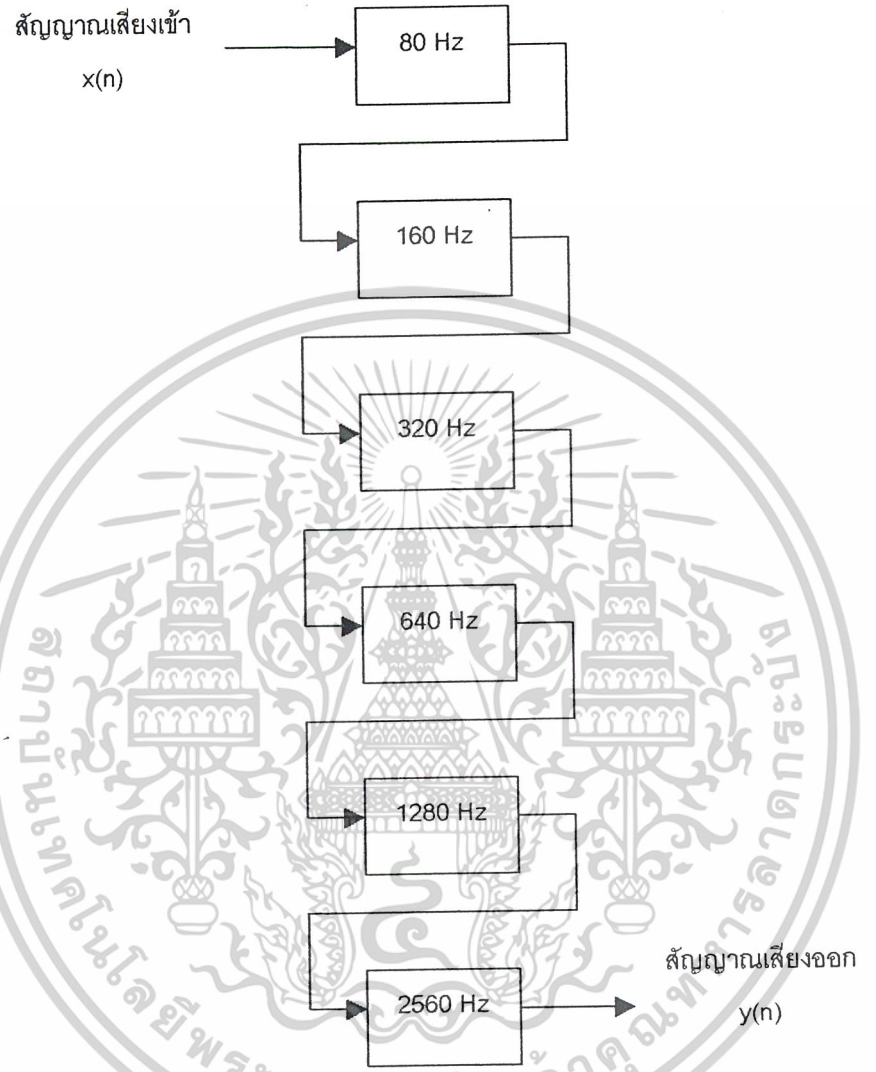
3.3 การทำให้เป็นผลสำเร็จในด้านซอฟต์แวร์ (Software Implementation) จากการออกแบบข้างต้นของตัวกรองแต่ละตัว นำมารวมกันเขียนเป็นไดอะแกรมอย่างง่ายได้ดังรูปที่ 3.7 และจากสมการสืบเนื่องของตัวกรองแต่ละตัวนำมาเขียนเป็นโปรแกรมภาษาซี โดยที่

ไฟล์ EQ.C เป็นไฟล์ที่รับอินพุทเข้ามาคำนวณและส่งเอาท์พุทออกไป

AICCOMC.C เป็นไฟล์ที่ทำการติดต่อสื่อสารกับAIC และการกำหนดค่าเริ่มต้นต่าง ๆ (Initialization)

EQ.CMD เป็นไฟล์ที่เก็บคำสั่งเพื่อทำการเชื่อมโยง (Linking)

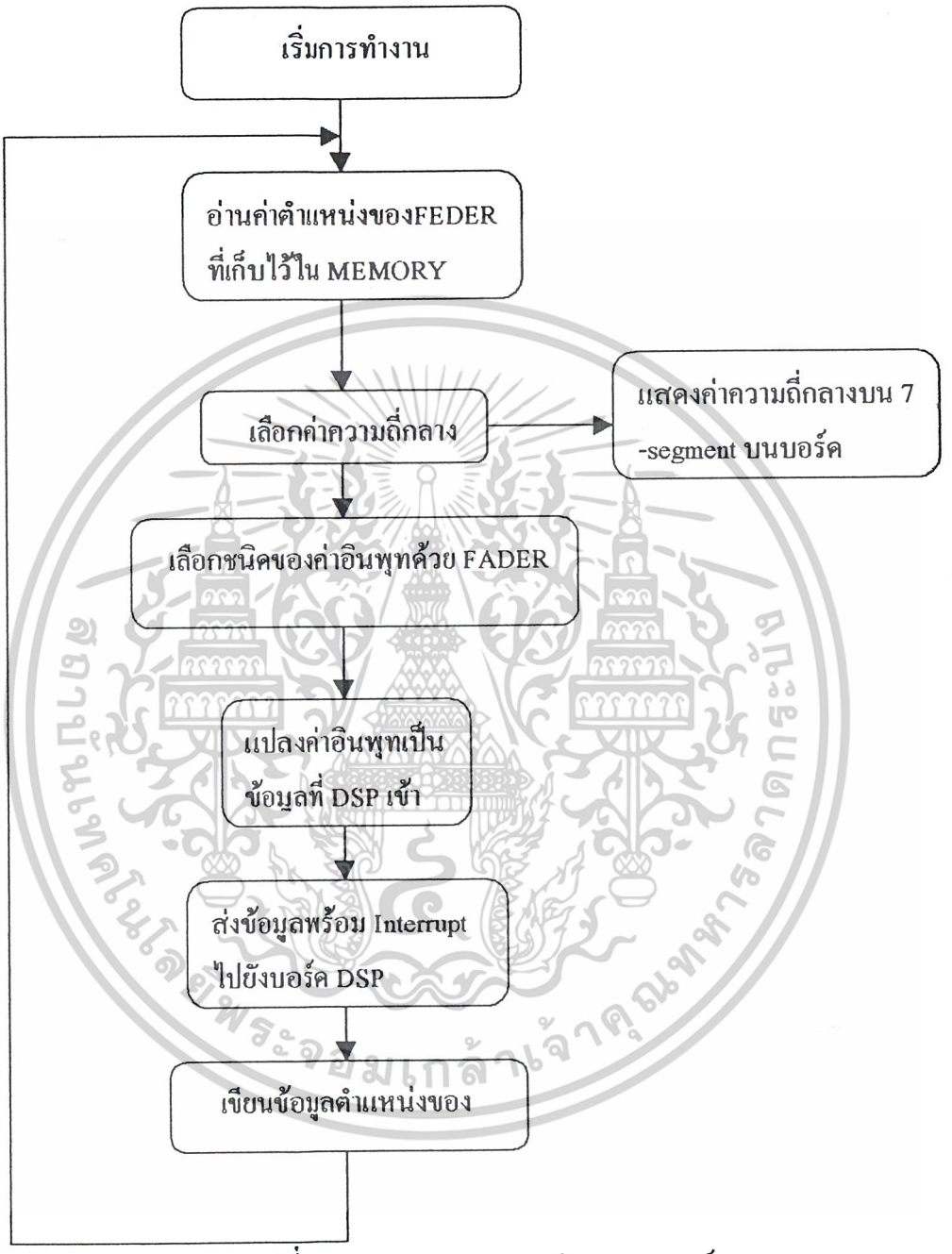
ซึ่งรายละเอียดของไฟล์ข้างต้นมีดังในภาคผนวก



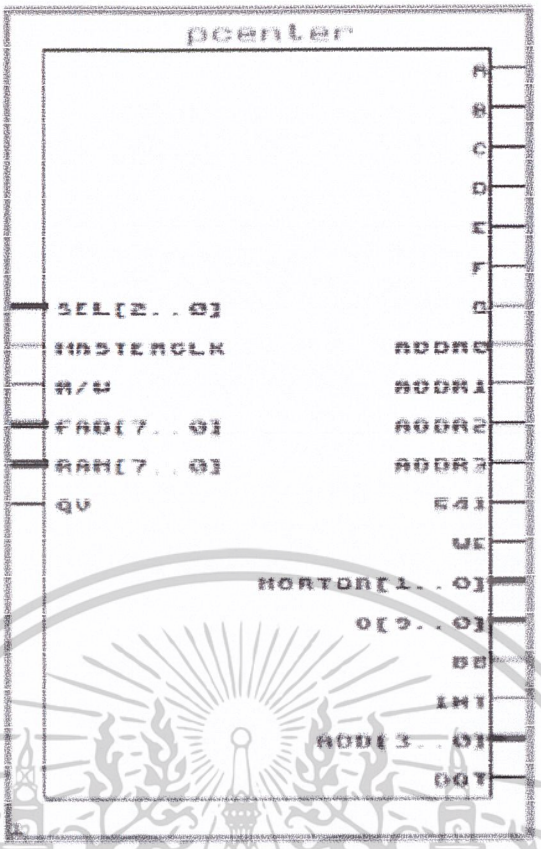
รูปที่ 3.7 ไดอะแกรมอย่างง่ายของ Equalizer

3.4 การออกแบบอุปกรณ์ควบคุม

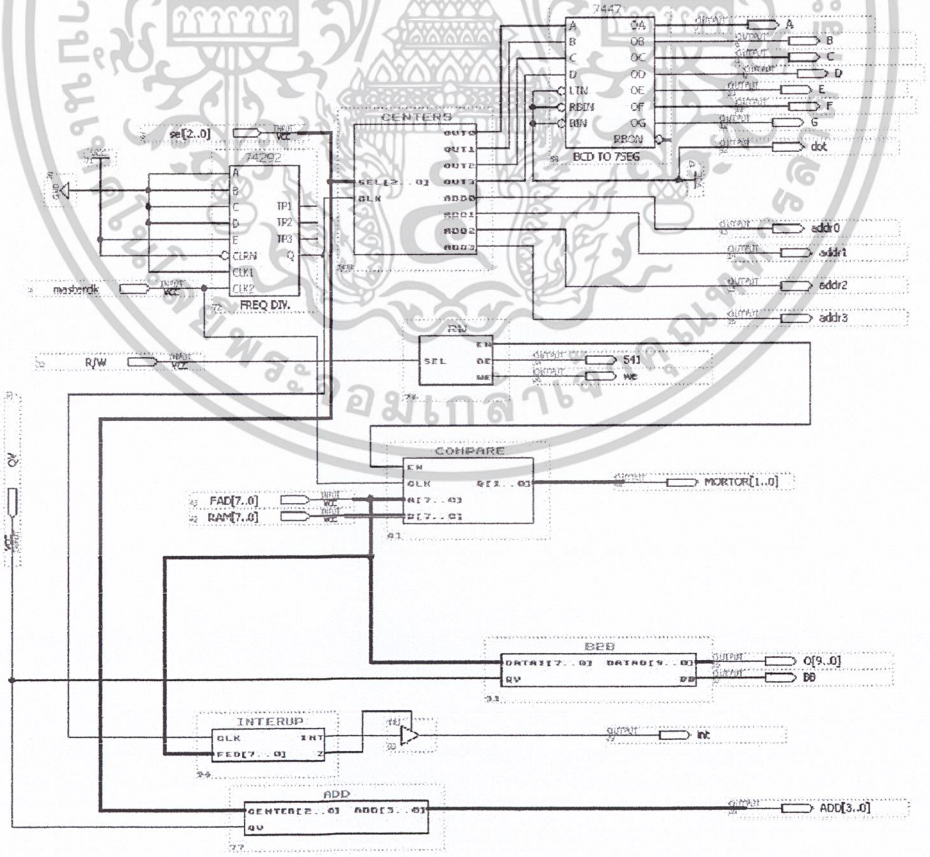
ในส่วนของอุปกรณ์ควบคุมนั้นเราจะใช้ FPGA ควบคุมโดยออกแบบวงจรที่นำมาใช้ร่วมกับบอร์ด DSP โดย FPGA ทำหน้าที่ควบคุมจอแสดงผลค่าความถี่กลาง ,ควบคุมการอ่านและเขียน Memory ,แปลงค่าที่ได้จากอินพุทที่ได้และส่งไปยังบอร์ด DSP พร้อมสัญญาณ Interrupt และควบคุมตำแหน่งของ Fader ให้อยู่ในตำแหน่งที่เก็บค่าไว้ใน Memory โดยมีลำดับการทำงานดังในรูป 3.8



รูปที่ 3.8 Block diagram การทำงานของบอร์ด FPGA



รูปที่ 3.9 โครงสร้างภายนอกของอุปกรณ์ควบคุม



รูปที่ 3.10 โครงสร้างภายในของอุปกรณ์ควบคุม

การเขียนโปรแกรมลงบนบอร์ด FPGA เรานั้นใช้โปรแกรม MAX+plus II เวอร์ชั่น 10.0 BASELINE โปรแกรมลงบนชิพ ALTERA เบอร์ EPM7128SLC84-10

ตำแหน่งขาของอุปกรณ์มีดังนี้

1. **ชุดข้อมูลจาก** Fader เป็นชุดที่ต่อตรงอยู่กับ ADC เพื่อนำข้อมูลมาประมวลผลและความคุมตำแหน่ง Fader

   ข้อมูลบิทที่1 (FAD0)ต่ออยู่กับขา 64

   ข้อมูลบิทที่2 (FAD1)ต่ออยู่กับขา 65

   ข้อมูลบิทที่3 (FAD2)ต่ออยู่กับขา 67

   ข้อมูลบิทที่4 (FAD3)ต่ออยู่กับขา 68

   ข้อมูลบิทที่5 (FAD4)ต่ออยู่กับขา 69

   ข้อมูลบิทที่6 (FAD5)ต่ออยู่กับขา 70

   ข้อมูลบิทที่7 (FAD6)ต่ออยู่กับขา 73

   ข้อมูลบิทที่8 (FAD7)ต่ออยู่กับขา 74

2. **ชุดข้อมูลจาก** Memory ข้อมูลที่ได้จากชุดนี้จะนำไปควบคุมตำแหน่ง Fader

   ข้อมูลบิทที่1 (RAM0)ต่ออยู่กับขา 75

   ข้อมูลบิทที่2 (RAM1)ต่ออยู่กับขา 76

   ข้อมูลบิทที่3 (RAM2)ต่ออยู่กับขา 77

   ข้อมูลบิทที่4 (RAM3)ต่ออยู่กับขา 79

   ข้อมูลบิทที่5 (RAM4)ต่ออยู่กับขา 80

   ข้อมูลบิทที่6 (RAM5)ต่ออยู่กับขา 81

   ข้อมูลบิทที่7 (RAM6)ต่ออยู่กับขา 11

   ข้อมูลบิทที่8 (RAM7)ต่ออยู่กับขา 12

3. **ชุดข้อมูล** DSP เป็นชุดข้อมูลที่ส่งให้กับบอร์ด DSP เพื่อนำไปประมวลผล

   ข้อมูลบิทที่1 (O0)ต่ออยู่กับขา 28

   ข้อมูลบิทที่2 (O1)ต่ออยู่กับขา 29

   ข้อมูลบิทที่3 (O2)ต่ออยู่กับขา 30

   ข้อมูลบิทที่4 (O3)ต่ออยู่กับขา 22

   ข้อมูลบิทที่5 (O4)ต่ออยู่กับขา 24

   ข้อมูลบิทที่6 (O5)ต่ออยู่กับขา 25

   ข้อมูลบิทที่7 (O6)ต่ออยู่กับขา 27

   ข้อมูลบิทที่8 (O7)ต่ออยู่กับขา 49

   ข้อมูลบิทที่9 (O8)ต่ออยู่กับขา 50

ข้อมูลบิทที่10 (O9)ต่ออยู่กับขา 51

ข้อมูลบิทที่11 ()ต่ออยู่กับขา 63

ข้อมูลบิทที่12 ต่ออยู่กับขา 61

ข้อมูลบิทที่13 ต่ออยู่กับขา 60

ข้อมูลบิทที่14 ต่ออยู่กับขา 55

ข้อมูลบิทที่15 ต่ออยู่กับขา 17

4. **ชุดควบคุม** Memory ข้อมูลชุดนี้เป็นเอาท์พุทที่ใช้ควบคุมการทำงานของ Memory และบัฟเฟอร์

- ควบคุมการอ่าน ,เขียน Memory "WE" ติออยู่กับขา 4
- ควบคุมบัฟเฟอร์ "OE" ต่ออยู่กับขา 5
- แอดเดรส Memory บิทที่ 1 (ADD0)ต่ออยู่กับขา 6
  บิทที่ 2 (ADD1)ต่ออยู่กับขา 18
  บิทที่ 3 (ADD2)ต่ออยู่กับขา 9
  บิทที่ 4 (ADD3)ต่ออยู่กับขา 10

5. **ชุดแสดงผลความถี่กลาง** ชุดนี้จะต่อกับ 7 segment ที่อยู่บนบอร์ด FPGA ทำหน้าที่แสดงค่าความถี่กลาง โดยจะทำงานแบบ Dynamic

ไดโอด A ต่ออยู่กับ ขา 33

ไดโอด B ต่ออยู่กับ ขา 34

ไดโอด C ต่ออยู่กับ ขา 35

ไดโอด D ต่ออยู่กับ ขา 36

ไดโอด E ต่ออยู่กับ ขา 37

ไดโอด F ต่ออยู่กับ ขา 39

ไดโอด G ต่ออยู่กับ ขา 40

ขา Common ของ 7 segment หลักที่ 1 (ADDR0)ต่ออยู่กับขา44

ขา Common ของ 7 segment หลักที่ 2 (ADDR1)ต่ออยู่กับขา45

ขา Common ของ 7 segment หลักที่ 3 (ADDR2)ต่ออยู่กับขา46

ขา Common ของ 7 segment หลักที่ 4 (ADDR3)ต่ออยู่กับขา48

6. **ชุดสวิชซ์เลือกความถี่กลาง** ใช้ดิฟสวิชซ์บนบอร์ด F4GA เป็นตัวควบคุมโดย

D5 – D7 ใช้ควบคุมความถี่กลาง(SEL0-SEL2) ต่ออยู่กับขา 60,61 และ 63 ตามลำดับ

D0 ใช้เลือกการอ่าน ,เขียน Memory (R/W)ต่ออยู่กับขา 54

D1 ใช้เลือกค่า Q-factor และ ขนาด Amplitude (QV)ต่ออยู่กับขา 55

การทำงานของโปรแกรมสามารถบรรยายได้คือ เมื่อเริ่มการทำงานจอ 7 segment จะแสดงค่าความถี่กลางตามสวิชซ์ D5 – D7 สวิชซ์ D1 จะถูกใช้เป็นตัวเลือกว่าค่าที่ FPGA ส่งไปยังบอร์ด

DSP จะเป็นค่า Q-factor หรือค่าขนาดการ boost หรือ cut โดยจะสอดคล้องกับตำแหน่งของ Fader สวิชช์ D0 เป็นตัวกำหนดว่า memory กำลังทำงานอย่างไร หากกำหนดให้เขียนข้อมูลบอร์ด FPGA จะส่งสัญญาณไห้ memory รับข้อมูลที่ออกมาจากบัฟเฟอร์ตามแอดเดรสต่างๆกันซึ่งแปลไปตามค่าความถี่กลางและค่า Q-factor หรือค่าขนาดการ boost, cut แต่ถ้าหาก memory ถูกกำหนดให้อ่านข้อมูล FPGAจะนำค่าที่ได้จากfader และค่าที่ได้จากmemoryมาเปรียบเทียบกันและทำการสั่งfaderให้ไปยังตำแหน่งนั้น

บทที่ 4
การทดลองและผลการทดลอง

**การทดลองแยกออกเป็น 2 ส่วนดังนี้

4.1 ผลการทดลองอุปกรณ์ควบคุม

4.1.1 การจดจำตำแหน่งของFader

การทดลองหัวข้อนี้จะทดสอบความสามารถในการจดจำตำแหน่งของFader ที่ค่าQ-factor ,Magnitude ของค่าความถี่กลางต่างๆกันโดยจะสุ่มเอาตำแหน่งของFader ของแต่หัวข้อเพียงค่าเดียว

ตารางที่ 4.1 แสดงความสามารถในการจดจำตำแหน่งของFader

ค่าความถี่กลาง	ตำแหน่งFader ของค่าQ-factor	ค่าที่เรียกได้	ตำแหน่งFader ของค่าMagnitude	ค่าที่เรียกได้
80 Hz	00011001	00011X1X	00000111	0000X111
160 Hz	00011101	0001111X	00010010	00010011
320 Hz	00111111	00111111	00100110	00100110
640 Hz	011111010	01111011	01010101	010101XX
1280 Hz	11011001	110110XX	10010011	10010011
2560 Hz	11111001	11111XXX	11111111	11111111

* หมายเหตุ X คือแรงดันที่ไม่สามารถระบุเป็นลอจิกได้มีค่าประมาณ 2 Volt

**หมายเหตุ เนื่องจากบอร์ดประมวลผลสัญญาณดิจิตอล เกิดอุบัติเหตุเสียหายระหว่างขั้นตอนการสร้าง ทำให้ไม่สามารถทำการเชื่อมต่อ(Interface) กับ FPGAได้ จึงแบ่งการทดลองโดย ทดลองแยกแต่ละส่วน

4.1.2 ความสามารถในการแปลงค่าข้อมูลเพื่อส่งให้แก่บอร์ด DSP

บอร์ด FPGA จะต้องทำหน้าที่แปลงข้อมูลที่ได้จากFader เพื่อบอกแก่ DSP ว่าค่า Magnitude และค่า Q-factor ที่ผู้ใช้ต้องการคือค่าเท่าใด โดยจะส่งไปเป็นเลขฐาน 2 จำนวน15 บิท เป็นข้อมูล10 บิท และสัญญาณบอกชนิดข้อมูล 4 บิทและบอกชนิดของตัวกรอง 1 บิท เลข Magnitude ที่ส่งไปจะมีค่า 100 ถึง1,000 DSPต้องนำค่าที่ได้ไปคูณกับ 0.01 เพื่อให้ได้ค่า 1 ถึง 10 และค่า Q-factor ที่ส่งไปจะมีค่า1 ถึง 200 DSPต้องนำค่าที่ได้ไปคูณ0.1 ก็จะได้ค่า 0.1 ถึง 20 การทดลองจะสุ่มค่าแต่ละชนิดเพียง 2 ค่า

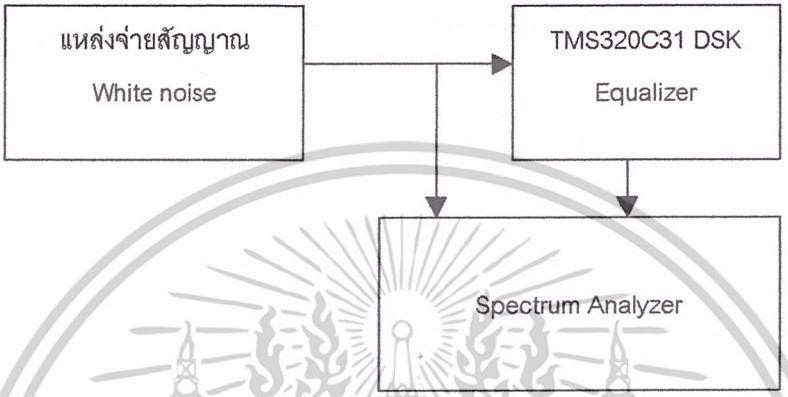
ตารางที่ 4.2 แสดงความสามารถในการแปลงข้อมูล

ชนิดของข้อมูล	ค่าที่ได้จาก Fader	ค่าที่ส่งให้ DSP	ค่าที่ได้จาก Fader	ค่าที่ส่งให้ DSP
Magnitude Fc = 80 Hz	00000000	0000111111101000 Cut 1,000	11100111	0000111100111000 Cut 824
Q – Factor Fc = 80 Hz	00000000	0001000000001010 (10)	00110110	0001000000001110 (14)
Magnitude Fc=160 Hz	00110101	0010110010110000 Cut 600	01101100	0010100101000000 Cut 160
Q – Factor Fc=160 Hz	00011101	0011000000111101 (29)	00101010	0011000001010100 (42)

ชนิดของข้อมูล	ค่าที่ได้จาก Fader	ค่าที่ส่งให้ DSP	ค่าที่ได้จาก Fader	ค่าที่ส่งให้ DSP
Magnitude Fc=320 Hz	01111001	010010001100100 Cut 100	10000101	010000001100100 Boost 100
Q – Factor Fc=320 Hz	00111011	010100000111011 (59)	01001011	10100001001111 (79)
Magnitude Fc=640 Hz	10001101	011000001101100 Cut 128	10011011	011000011011000 Boost 216
Q – Factor Fc=640 Hz	01101011	0111000011101011 (107)	10010111	0111000010010111 (151)
Magnitude Fc=1280Hz	10100111	0111000011101011 Boost 107	10010111	0111000010010111 Boost 151
Q – Factor Fc=1280Hz	10101010	1001000101010 (170)	11000010	1001000011000010 (194)
Magnitude Fc=2560Hz	11101110	1010011011110000 Boost 880	11111111	1010011111101000 Boost 1,000
Q – Factor Fc=2560Hz	11011110	1011000011001000 (200)	11111111	1011000011001000 (200)

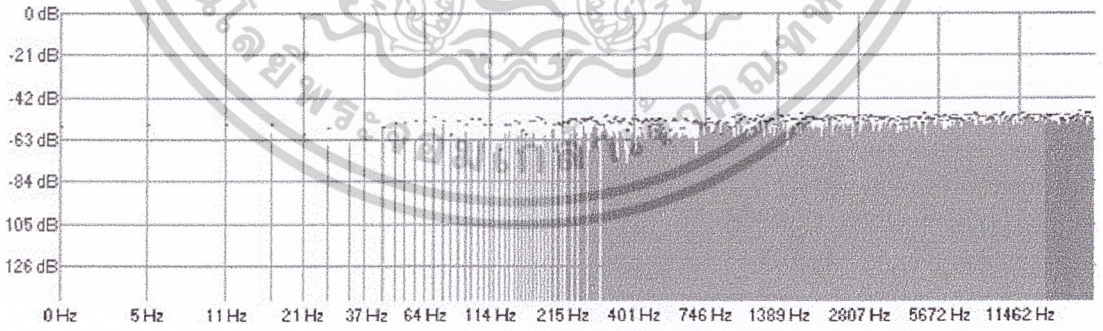
4.2 การทดลองส่วนประมวลผลสัญญาณดิจิตอล

ในการสร้างแหล่งจ่ายสัญญาณ White noise ซึ่งมีระดับสัญญาณเท่า ๆ กันในทุกช่วงความถี่ และ วัดการตอบสนองทางความถี่ (Frequency response) ใช้โปรแกรม Wave Lab V3.0

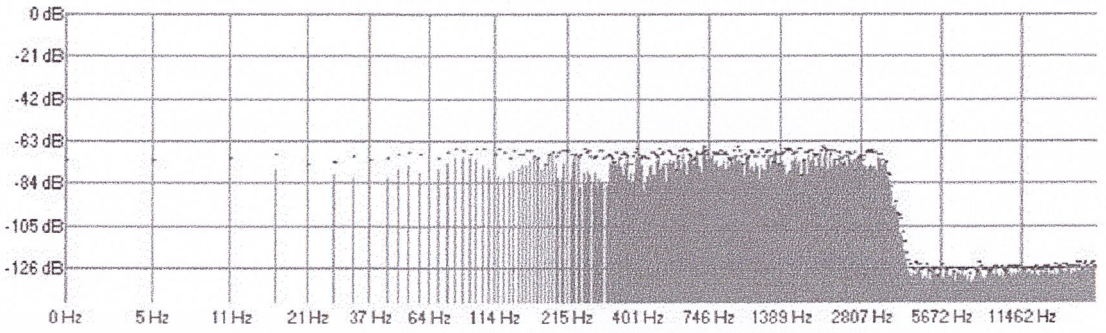


รูปที่ 4.1 ระบบการทดลอง

แล้วทำการทดสอบว่าระบบการทดลองนี้เป็นเชิงเส้น โดยการจ่ายสัญญาณ White noise ดังรูปที่ 4.2 และวัดที่เอาท์พุท โดยตั้งค่าของอีควอไลเซอร์ให้มีผลการตอบสนองความถี่เรียบ(flat) โดยมีค่าอินพุทได้ดังรูปที่ 4.3



รูปที่ 4.2 ผลตอบสนองความถี่ของแหล่งจ่ายสัญญาณ White noise



รูปที่ 4.3 ผลตอบสนองความถี่ของอินพุทเมื่อปรับให้อีคลอไลเซอร์มีการตอบสนองความถี่เรียบ

แล้วทำการทดลองเปลี่ยนค่าพารามิเตอร์ต่าง ๆใน อีควอไลเซอร์ ดังต่อไปนี้

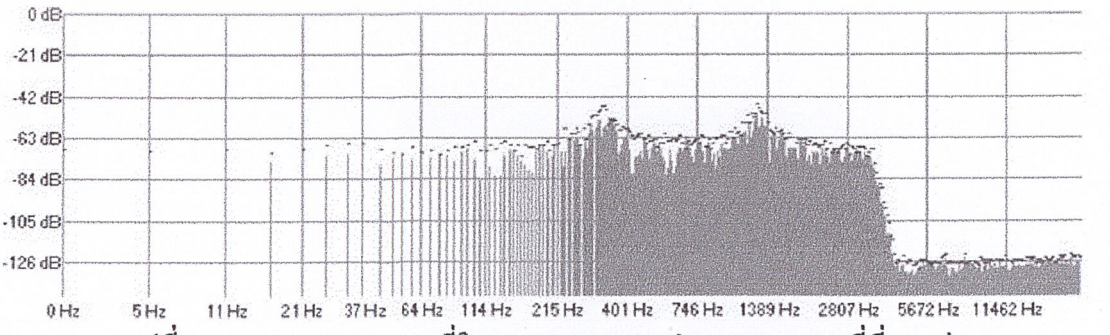
4.2.1 ทดสอบการทำงานของความถี่ที่แตกต่าง

กำหนดค่าพารามิเตอร์ของตัวกรองแต่ละตัวดังนี้

ตารางที่ 4.3 การทดสอบการทำงานของความถี่ที่แตกต่าง

ความถี่(fc) (Hz)	อัตราขยาย(V0) (เท่า)	ตัวประกอบคุณภาพ(Q)	ลักษณะของตัวกรอง(Ch) Boost = 0, Cut = 1
80	1	10	0
160	1	10	0
320	10	10	0
640	1	10	0
1280	10	10	0
2560	1	10	0

ได้ผลดังนี้



รูปที่ 4.4 ผลตอบสนองความถี่ในการทดสอบการทำงานของความถี่ที่แตกต่าง

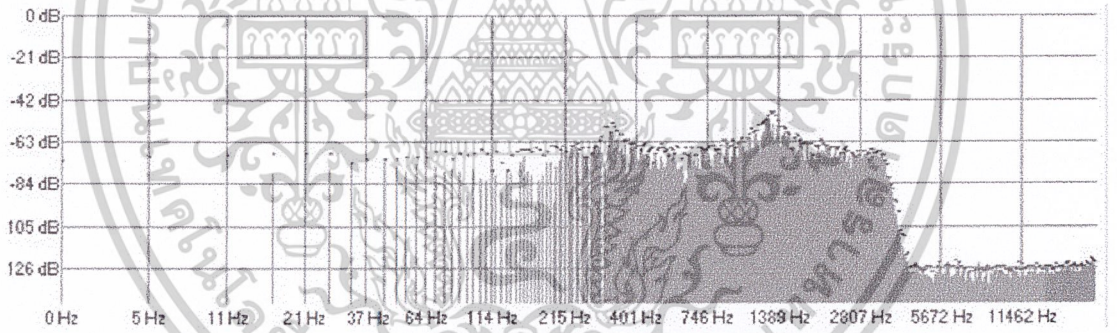
4.2.2 ทดสอบการเปลี่ยนค่าอัตราขยาย

กำหนดค่าพารามิเตอร์ของตัวกรองแต่ละตัวดังนี้

ตารางที่ 4.4 การทดสอบการเปลี่ยนค่าอัตราขยาย

ความถี่(fc) (Hz)	อัตราขยาย(V0) (เท่า)	ตัวประกอบคุณภาพ(Q)	ลักษณะของตัวกรอง(Ch) Boost = 0, Cut = 1
80	1	10	0
160	1	10	0
320	5	10	0
640	1	10	0
1280	10	10	0
2560	1	10	0

ได้ผลดังนี้



รูปที่ 4.5 ผลตอบสนองความถี่ในการทดสอบการเปลี่ยนอัตราขยาย

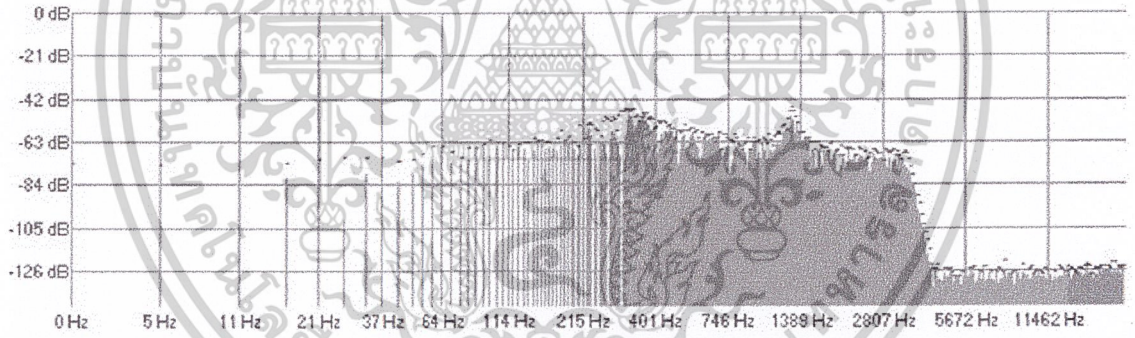
4.2.3 ทดสอบการเปลี่ยนค่าตัวประกอบคุณภาพ

กำหนดค่าพารามิเตอร์ของตัวกรองแต่ละตัวดังนี้

ตารางที่ 4.5 การทดสอบการเปลี่ยนค่าตัวประกอบคุณภาพ

ความถี่(fc) (Hz)	อัตราขยาย(V0) (เท่า)	ตัวประกอบคุณภาพ(Q)	ลักษณะของตัวกรอง(Ch) Boost = 0, Cut = 1
80	1	10	0
160	1	10	0
320	10	3	0
640	1	10	0
1280	10	20	0
2560	1	10	0

ได้ผลดังนี้



รูปที่ 4.6 ผลตอบสนองความถี่ในการทดสอบการเปลี่ยนค่าตัวประกอบคุณภาพ

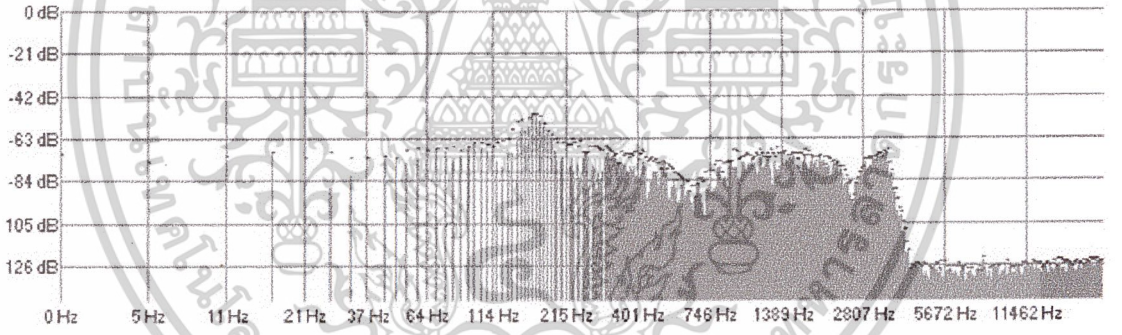
4.2.4 ทดสอบการเปลี่ยนค่าลักษณะของตัวกรอง

กำหนดค่าพารามิเตอร์ของตัวกรองแต่ละตัวดังนี้

ตารางที่ 4.6 การทดสอบการเปลี่ยนค่าลักษณะของตัวกรอง

ความถี่(fc) (Hz)	อัตราขยาย(V0) (เท่า)	ตัวประกอบคุณภาพ(Q)	ลักษณะของตัวกรอง(Ch) Boost = 0, Cut = 1
80	1	10	0
160	10	10	0
320	1	10	0
640	10	10	1
1280	1	10	0
2560	10	10	1

ได้ผลดังนี้



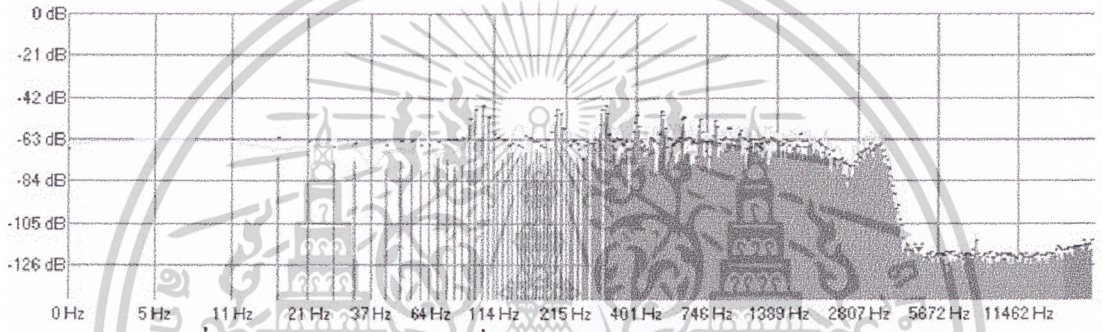
รูปที่ 4.7 ผลตอบสนองความถี่ในการทดสอบการเปลี่ยนค่าลักษณะของตัวกรอง

4.2.5 ทดสอบการจำลองการถอดรหัสจาก FPGA

ทำการทดสอบโดยเพิ่มในโปรแกรมส่วนของการถอดรหัส และใส่ค่ารหัสต่างๆ ลงบนแอสเดรสในหน่วยความจำที่กำหนด (0x809E00)

ตารางที่ 4.7 รหัสที่ทำการถอดรหัส(ก)

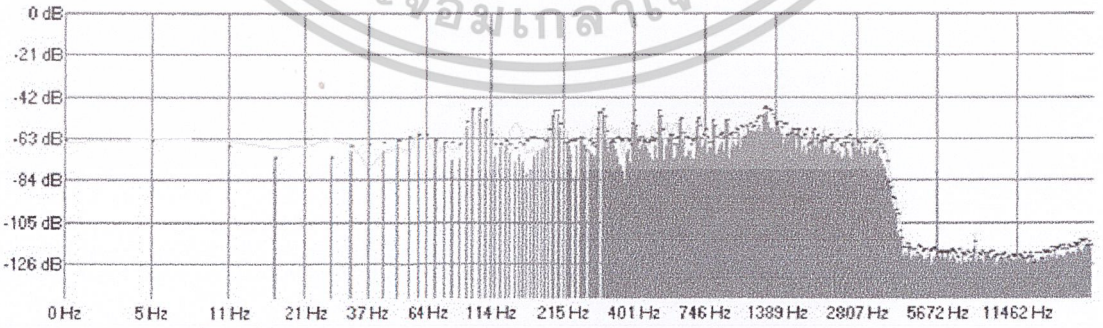
ความถี่	พารามิเตอร์ที่ต้องการเปลี่ยน	ค่าที่ต้องการเปลี่ยน	รหัส
2560	อัตราขยาย V0	-10 (Cut)	101 0110 0111 1000B (0x57E8)



รูปที่ 4.8 ผลตอบสนองความถี่จากการจำลองการถอดรหัสค่าจาก FPGA (ก)

ตารางที่ 4.8 รหัสที่ทำการถอดรหัส(ข)

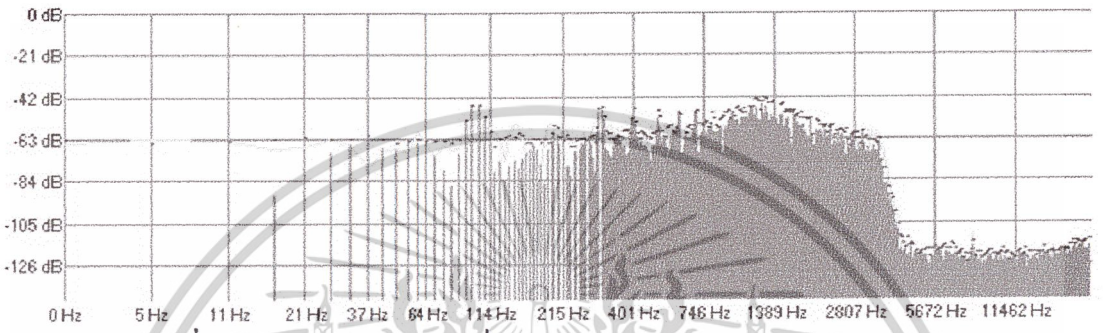
ความถี่	พารามิเตอร์ที่ต้องการเปลี่ยน	ค่าที่ต้องการเปลี่ยน	รหัส
1280	อัตราขยาย V0	+10 (Boost)	100 0010 0111 1000 B (0x4278)



รูปที่ 4.9 ผลตอบสนองความถี่จากการจำลองการถอดรหัสค่าจาก FPGA (ข)

ตารางที่ 4.9 รหัสที่ทำการถอดรหัส (ค)

ความถี่	พารามิเตอร์ที่ต้องการเปลี่ยน	ค่าที่ต้องการเปลี่ยน	รหัส
1280	ตัวประกอบคุณภาพ Q	3	100 1000 0001 1110 B (0x481E)



รูปที่ 4.10 ผลตอบสนองความถี่จากการจำลองการถอดรหัสค่าจาก FPGA (ค)

บทที่ 5
สรุปและวิจารณ์ผลโครงงาน

5.1 สรุปและวิจารณ์ผลการทดลอง

ในการทดลองซึ่งทำการพิสูจน์ว่าระบบการทดลองและอุปกรณ์ต่าง ๆ มีความเป็นเชิงเส้นหรือไม่ พบว่ามีความเป็นเชิงเส้นดี ในย่านความถี่ที่ต้องการ สังเกตได้จากมีผลการตอบสนองความถี่ระหว่างอินพุท (White noise) และเอาท์พุทที่คล้ายคลึงกัน แต่ก็อาจมีการลดทอนของสัญญาณลงบ้าง

จากการทดลองในส่วนประมวลผลสัญญาณดิจิตอล พบว่า เมื่อทดลองเปลี่ยนค่าพารามิเตอร์ต่าง ๆ ของ อีควอไลเซอร์ (หรือตัวกรองแต่ละตัว) ซึ่งได้แก่ค่า อัตราขยาย (V0), ตัวประกอบคุณภาพ (Q) และลักษณะของตัวกรองที่ความถี่ต่าง ๆ พบว่า ผลการตอบสนองความถี่ที่ได้มีความถูกต้อง แม่นยำ ซึ่งเป็นข้อดีของการประมวลผลสัญญาณดิจิตอล ซึ่งถ้าหากมีการควบคุมการเปลี่ยนแปลงค่าพารามิเตอร์ให้เหมาะสม (ไม่มากหรือน้อยเกินไป) ก็จะได้คุณภาพของการปรับแต่งเสียงที่ดี

ส่วนต่อมาเป็นการทดสอบการถอดรหัสข้อมูลจาก FPGA ซึ่งพบว่าการถอดรหัสเป็นไปอย่างถูกต้อง สามารถเปลี่ยนพารามิเตอร์ของอีควอไลเซอร์ได้ตามข้อมูลที่เข้ามา แต่ก็มีปัญหาที่สัญญาณเอาท์พุทที่ออกมา ความถี่ที่ผิดเพี้ยนคล้ายกับมี Aliasing Frequency เข้ามาปะปน ซึ่งปัญหานี้เกิดขึ้นหลังจากที่บอร์ดประมวลผลสัญญาณดิจิตอลได้รับความเสียหาย ทำให้การทำงานผิดเพี้ยนไป และยังเป็นปัญหาสำคัญที่ทำให้ไม่สามารถทำการเชื่อมต่อ(Interface) กับ FPGA ได้

ในส่วนของFPGA พบว่าการทำงานมีความถูกต้องแม่นยำดี ซึ่งเกิดจากการทำงานของแต่ละส่วนที่เป็นอิสระจากกันซึ่งเป็นข้อดีของFPGA จะพบความผิดพลาดบ้างซึ่งมีสาเหตุมาจากข้อมูลอินพุทที่ผิดเพี้ยน เป็นเพราะผลจากNoiseในระบบที่มีค่าต่อข้อมูลถึง 3บิทในโครงงานนี้ ด้านการเลือกอุปกรณ์และตัวโปรแกรมอธิบายฮาร์ดแวร์ค่อนข้างจะพอเหมาะกับความต้องการในระดับหนึ่ง ปัญหาด้านโปรแกรมที่เกิดขึ้นก็สามารถแก้ไขโดยใช้วิธีอื่นเข้ามาทดแทนได้

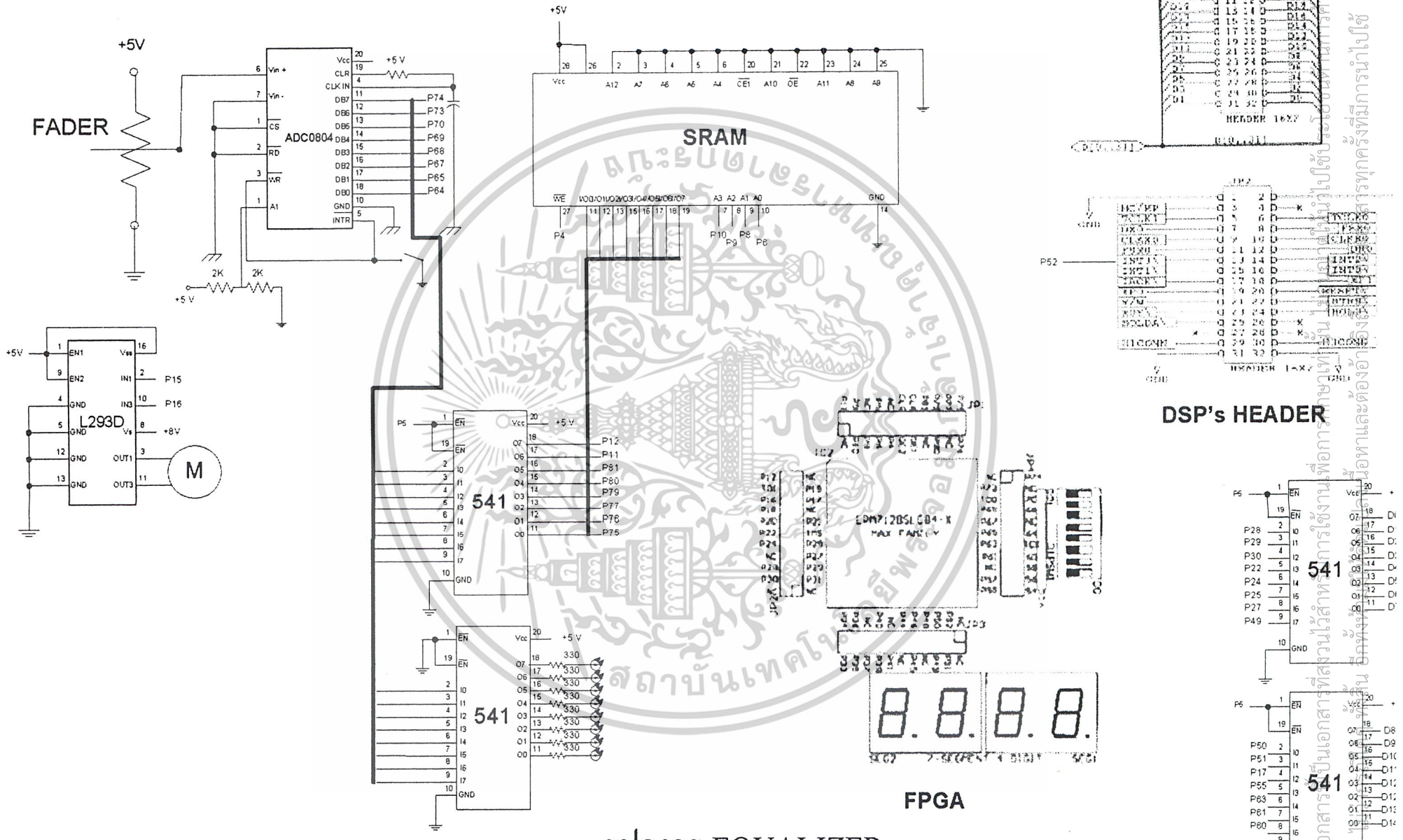
5.2 สิ่งที่ได้รับจากการศึกษาและปฏิบัติโครงงาน

1) ได้นำความรู้ที่ได้จากการเรียนในห้องเรียนเกี่ยวกับไมโครโพรเซสเซอร์ขั้นพื้นฐาน (เช่น MCS51, Z80) นำมาประยุกต์ใช้งานกับไมโครโพรเซสเซอร์ที่มีความซับซ้อนมากขึ้น (DSP และ FPGA)
2) ได้รับความรู้ในการออกแบบและการใช้งานเทคโนโลยีใหม่ ๆ ที่เป็นส่วนสำคัญยิ่งในการพัฒนาสิ่งประดิษฐ์ต่างๆ ในอนาคต
3) ฝึกฝนความรับผิดชอบ และการแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้น รวมทั้งสอนให้เกิดความระมัดระวัง รอบคอบ ในการปฏิบัติงาน

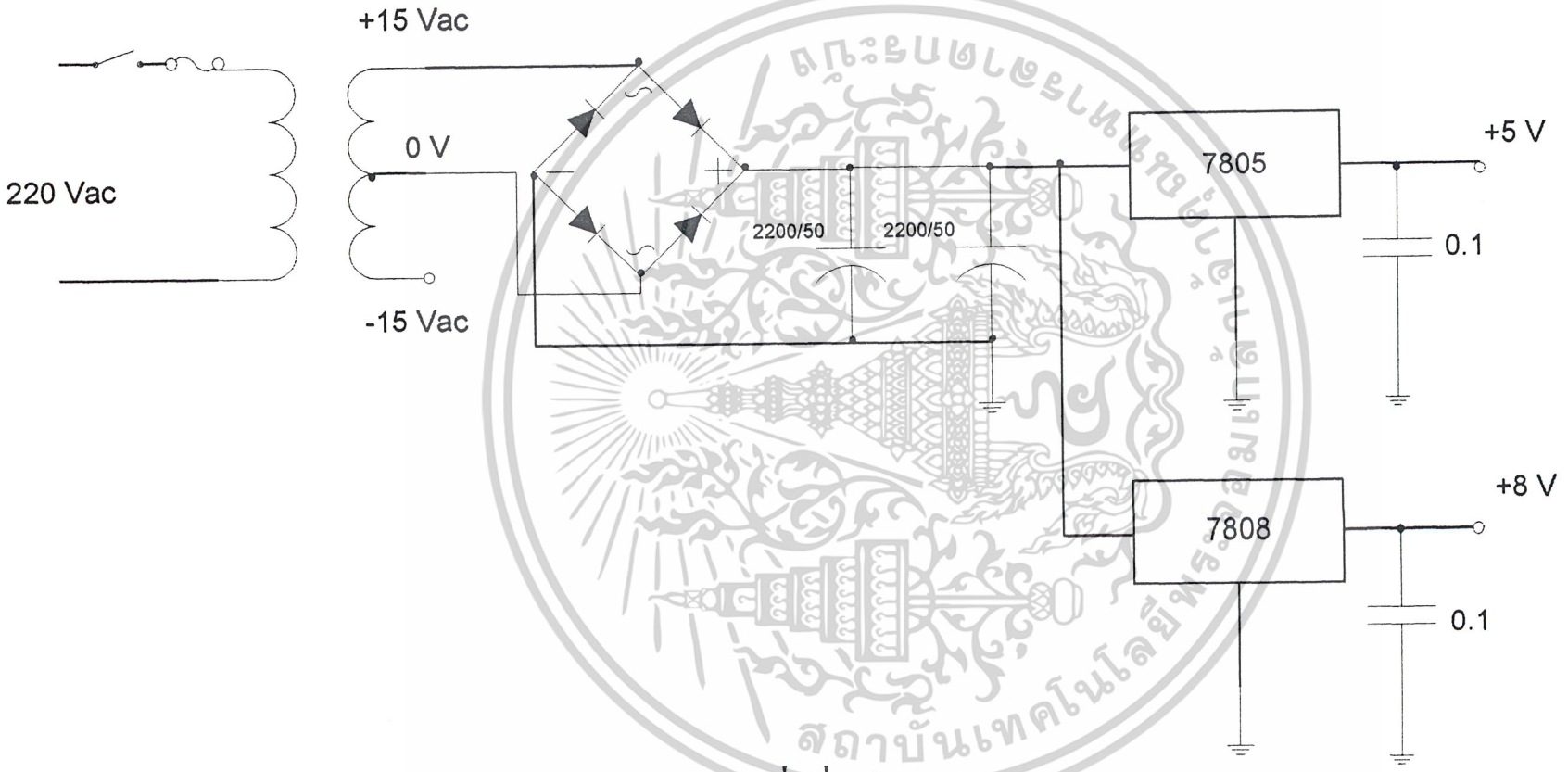
5.3 แนวทางในการพัฒนาโครงงานในอนาคต

ควรเลือกใช้ชิปประมวลผลสัญญาณดิจิตอลให้มีประสิทธิภาพที่ดีขึ้นและเหมาะสม เพื่อที่จะสามารถขยายช่องความถี่ให้มีจำนวนมากขึ้น และเลือกใช้ชิป FPGA ที่มีความเร็วและจำนวนเกทมากขึ้น เพื่อที่จะสามารถเพิ่มเติมการทำงานให้มีความหลากหลาย และเลือกใช้ CODEC ที่มีความอัตราความถี่สุ่ม(Sampling Rate)สูง(44.1kHz ขึ้นไป) และมีจำนวนบิตมากขึ้น(16 บิต ขึ้นไป) เพื่อคุณภาพเสียงที่ดี รวมทั้งการปรับปรุงลักษณะโครงสร้างของตัวกรองให้มีการลดทอนสัญญาณรบกวน(Noise Reduction)

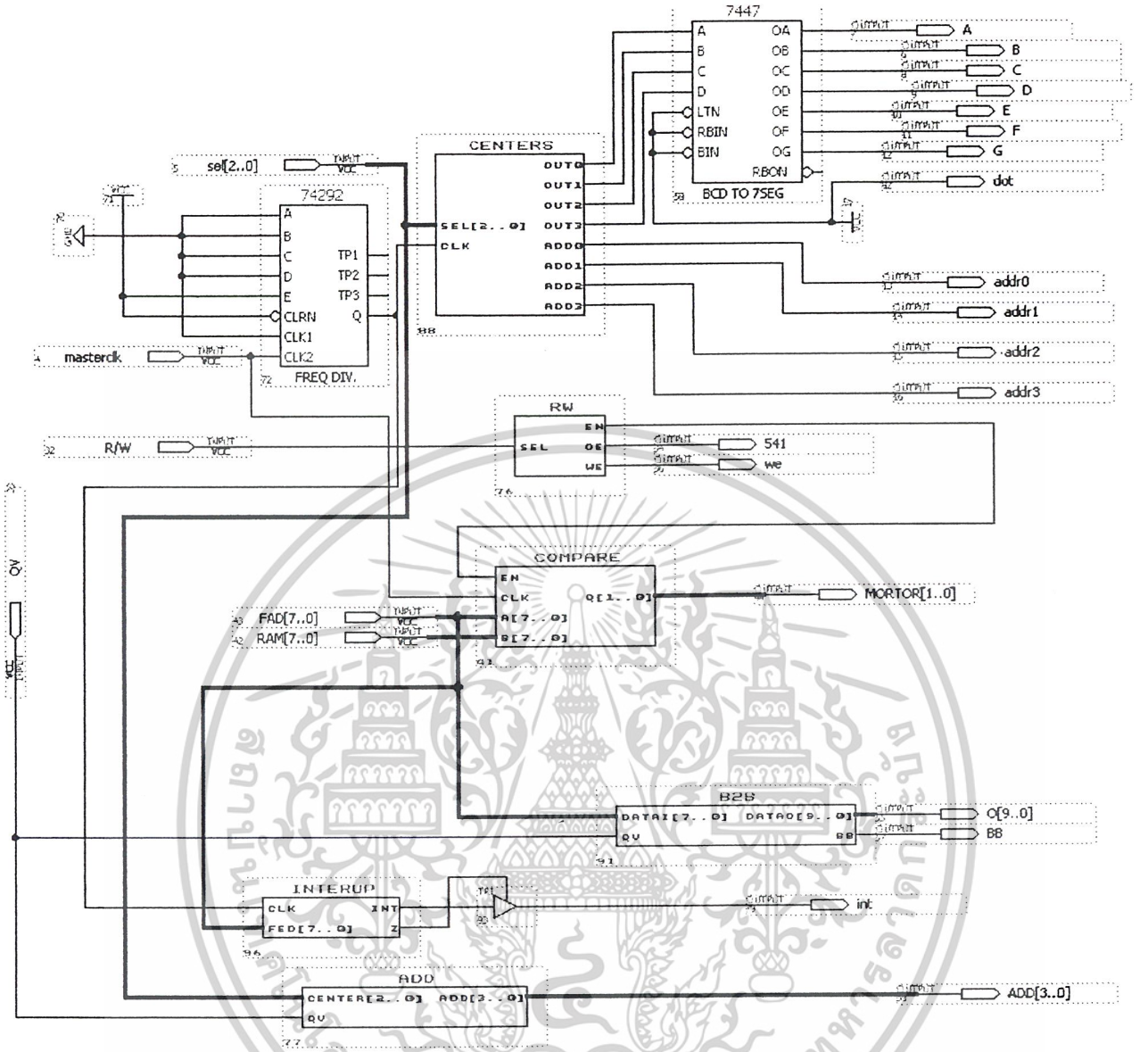
ภาคผนวก



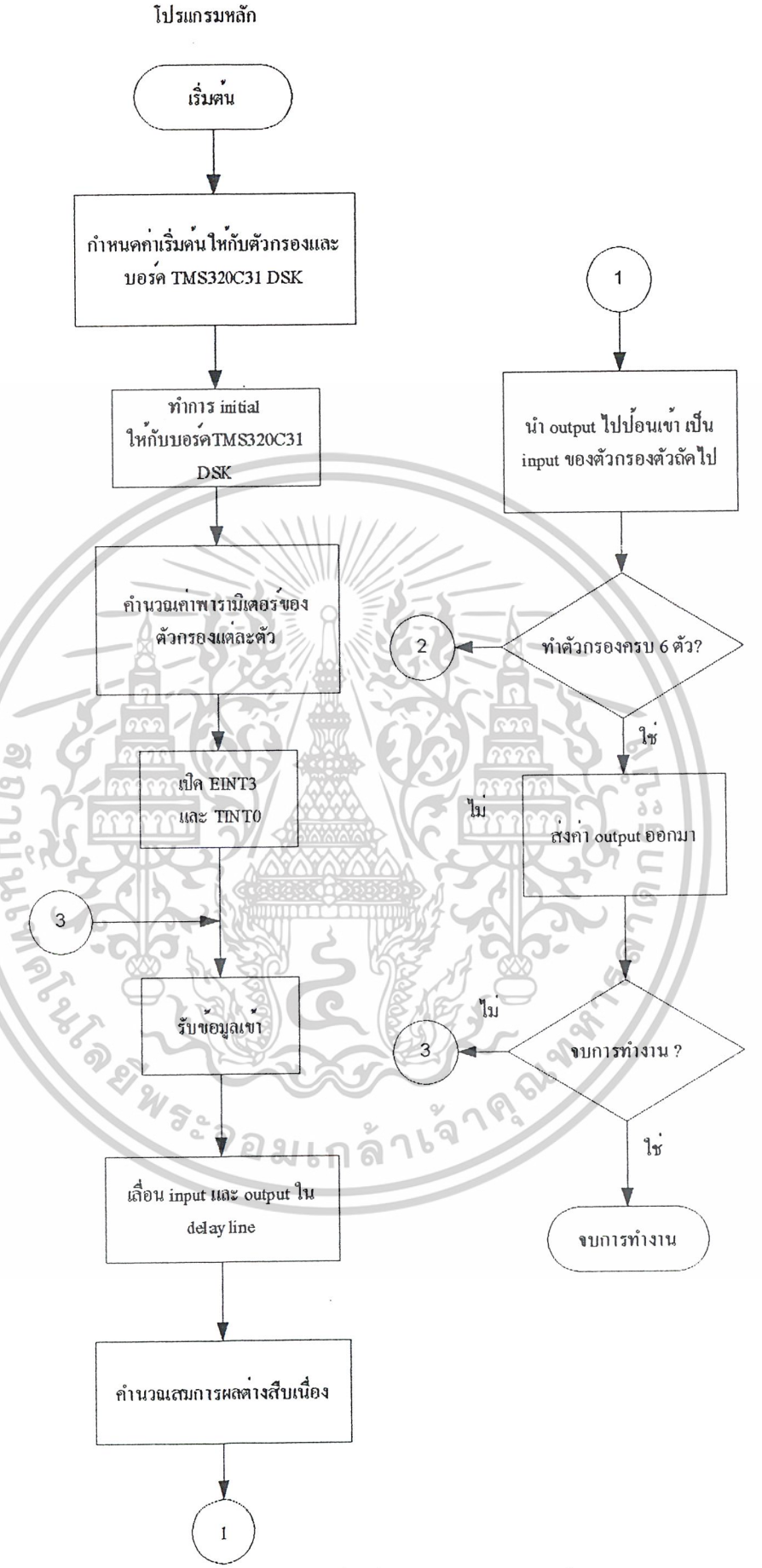
รูปวงจร EQUALIZER



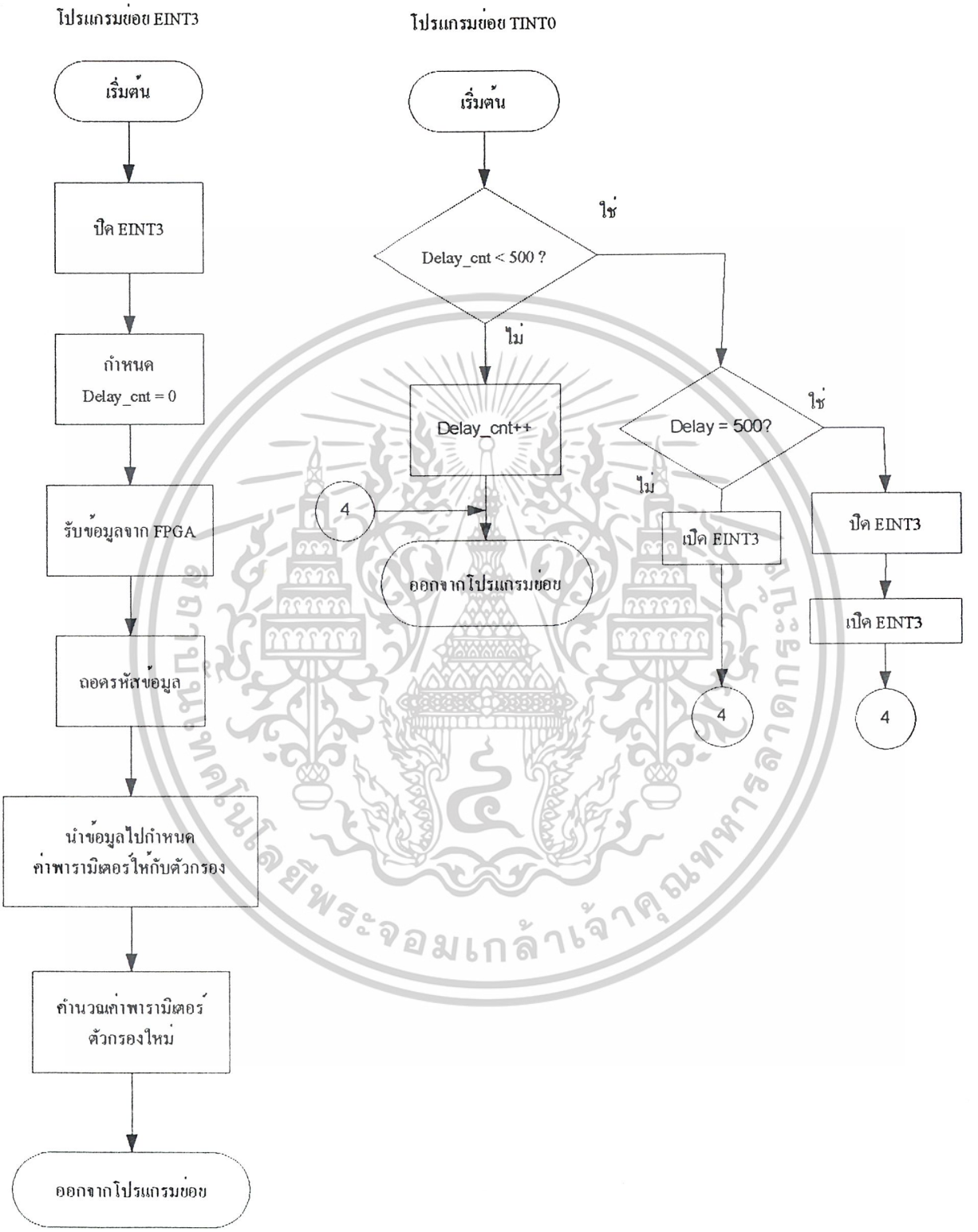
รูปวงจรแหล่งจ่ายไฟกระแสตรง



รูปโครงสร้างภายในของอุปกรณ์ควบคุม



โฟลว์ชาร์ตของโปรแกรมในส่วนของการประมวลผลสัญญาณเชิงเลข



โปรแกรมย่อยอินเทอร์รัพท์
ของส่วนประมวลผลสัญญาณดิจิตอล

โปรแกรมข้อมูลภาษาซีในส่วนประมวลผลสัญญาณดิจิตอล

File - EQ.C -

```
#include "aiccomc.c"
#include "MATH.H"

#define  pi 3.14159
#define  n 6   /* number of band */


int AICSEC[4] = {0x162C,0x1,0x4892,0x63};     // INITIAL AIC

/************************/
float Fs=8000,                                // SAMPLING RATE
      fc[n]={80,160,320,640,1280,2560},       // Center frequency
      V0[n]={1,1,1,1,1,1},                    // Gain
      Q[n]={10,10,10,10,10,10};               // Quality Factor

int   Ch[n] = {1,0,1,0,1,0};                  // Filter Characteristic
                                              // Boost=0, Cut=1
int   delay_cnt;


volatile int   *buffer= (volatile int*) 0x809E00,
               *buffer_cd= (volatile int*) 0x809E01,
               *buffer_dt= (volatile int*) 0x809E02,
               *buffer1= (volatile int*) 0x809E03,
               *buffer2= (volatile int*) 0x809E04,


               *buffer_rd= (volatile int*) 0x500000; //Dummy Address


float K,a0[n],a1[n],a2[n],b0[n],b1[n],b2[n];
int   data_in, data_out, i;
float x[n][3]={0}, y[n][3]={0},f;
/************************/

/* EQ Section */
void Coeff(int Charac)        // Function to calculate Coefficient
{

     if (Charac == 0)
     {
     K=tan(2*pi*fc[i]/(2*Fs));
          a0[i]=(1+(V0[i]*K/Q[i])+(K*K))/(1+(K/Q[i])+(K*K));
          a1[i]=2*((K*K)-1)/(1+(K/Q[i])+(K*K));
          a2[i]=(1-(V0[i]*K/Q[i])+(K*K))/(1+(K/Q[i])+(K*K));
          b0[i]=1;
          b1[i]=a1[i];
```

```
            b2[i]=(1-(K/Q[i])+(K*K))/(1+(K/Q[i])+(K*K));
      }
      else
      {
      K=tan(2*pi*fc[i]/(2*Fs));
            a0[i]=(1+(K/Q[i])+(K*K))/(1+(V0[i]*K/Q[i])+(K*K));
            a1[i]=2*((K*K)-1)/(1+(V0[i]*K/Q[i])+(K*K));
            a2[i]=(1-(K/Q[i])+(K*K))/(1+(V0[i]*K/Q[i])+(K*K));
            b0[i]=1;
            b1[i]=a1[i];
            b2[i]=(1-(V0[i]*K/Q[i])+(K*K))/(1+(V0[i]*K/Q[i])+(K*K));
      }
}


void Calculate(void)          //    Function to Calculate Filter
{
      int c;
      for (i=0;i<n;i++)
      {
            c = Ch[i];
            Coeff(c);
      }
}

float IIR(void)      // Function to Calculate Differential Equation
{
      float temp,fl;
      fl=0;

      for (i=0;i<n;i++)
      {
            temp = x[i][1];
            x[i][1] = x[i][0];
            x[i][2] = temp;
            x[i][0] = f;

            temp = y[i][1];
            y[i][1] = y[i][0];
            y[i][2] = temp;

            y[i][0] = a0[i]*x[i][0]+a1[i]*x[i][1]+a2[i]*x[i][2]
                  -b1[i]*y[i][1]-b2[i]*y[i][2];

            f=y[i][0];

      }

      fl = f;
      return fl;
}


void c_int04()              // INT3
```

```
{
     asm("     AND  0FFFFFFF7h,IE");     /* EINE3 = 0 */
     delay_cnt = 0;
     *buffer=*buffer_rd;       // Read data from primary bus (FPGA)
     if (*buffer < 0x5FFF && *buffer > 0x1000)
     {
          *buffer1=*buffer;
          *buffer2=*buffer;
          *buffer_cd=*buffer1 >> 10;                    //  code

          *buffer_dt=*buffer2 << 22 >>22;               //  data

     /***** Decode and Define data to Filter *****/

     if (*buffer_cd == 0x01)        // 80 Hz
     {
          Ch[0] = 1;
          V0[0] = *buffer_dt*0.01;
     }
     else
     if (*buffer_cd == 0x00)
     {
          Ch[0] = 0;
          V0[0] = *buffer_dt*0.01;
     }
     else
     if (*buffer_cd == 0x02 || *buffer_cd == 0x03)
     {
          Q[0] = *buffer_dt*0.1;
     }
     else
     if (*buffer_cd == 0x05)        // 160 Hz
     {
          Ch[1] = 1;
          V0[1] = *buffer_dt*0.01;
     }
     else
     if (*buffer_cd == 0x04)
     {
          Ch[1] = 0;
          V0[1] = *buffer_dt*0.01;
     }
     if (*buffer_cd == 0x06 || *buffer_cd == 0x07)
     {
          Q[1] = *buffer_dt*0.1;
     }
     else
     if (*buffer_cd == 0x09)        // 320 Hz
     {
          Ch[2] = 1;
          V0[2] = *buffer_dt*0.01;
     }
     else
     if (*buffer_cd == 0x08)
```

```
{
     Ch[2] = 0;
     V0[2] = *buffer_dt*0.01;
}
if (*buffer_cd == 0xA || *buffer_cd == 0xB)
{
     Q[2] = *buffer_dt*0.1;
}
else
if (*buffer_cd == 0x0D)        // 640 Hz
{
     Ch[3] = 1;
     V0[3] = *buffer_dt*0.01;
}
else
if (*buffer_cd == 0x0C)
{
     Ch[3] = 0;
     V0[3] = *buffer_dt*0.01;
}
if (*buffer_cd == 0xE || *buffer_cd == 0xF)
{
     Q[3] = *buffer_dt*0.1;
}
else
if (*buffer_cd == 0x11)        // 1280 Hz
{
     Ch[4] = 1;
     V0[4] = *buffer_dt*0.01;
}
else
if (*buffer_cd == 0x10)
{
     Ch[4] = 0;
     V0[4] = *buffer_dt*0.01;
}
if (*buffer_cd == 0x12 || *buffer_cd == 0x13)
{
     Q[4] = *buffer_dt*0.1;
}
if (*buffer_cd == 0x15)        // 2560 Hz
{
     Ch[5] = 1;
     V0[5] = *buffer_dt*0.01;
}
else
if (*buffer_cd == 0x14)
{
     Ch[5] = 0;
     V0[5] = *buffer_dt*0.01;
}
if (*buffer_cd == 0x16 || *buffer_cd == 0x17)
{
     Q[5] = *buffer_dt*0.1;
```

```
      }

      Calculate();
      }
}


void c_int05()  // XINT0       (not be used)
{
}
void c_int10() // TINT0
{
      if (delay_cnt < 500)
            delay_cnt++;
      else if (delay_cnt == 500);
      {
            asm("      AND   0FFFFFFF7h,IE");     /* EINE3 = 0 */
            asm("      OR    00000008h,IE");
      }
      else asm("      OR    00000008h,IE");
}

/************* MAIN ***************/
main()
{

      AICSET();                     // Initial AIC

      PBASE[0x38] = 0x00003D04;    // Set TINT0
      PBASE[0x30] = 0x00000C1;

      *buffer_rd = 0x0;             // Intiail Memory
      *buffer = 0x0;
      *buffer_dt = 0x0;
      *buffer1 = 0x0;
      *buffer2 = 0x0;

      Calculate();
      asm("      OR    00000208h,IE");
      for(;;)
      {
                  data_in = UPDATE_SAMPLE(data_out);
                  f=data_in;
                  data_out = IIR();

      }
}
```

File – AICCOMC.C –

```
/*AICCOMC.C - COMUNICATION ROUTINES FOR AIC*/
#define TWAIT while (!(PBASE[0x40] & 0x2)) /*wait till XMIT buffer
clear*/
extern int AICSEC[4];                      /*array defined in main
prog */
volatile int *PBASE = (volatile int *) 0x808000; /*peripherals base
addr*/

void AICSET()                              /*function to initialize AIC  */
{
volatile int loop;                         /*declare local variables     */
PBASE[0x28] = 0x00000001;                  /*set timer period            */
PBASE[0x20] = 0x000003C1;                  /*set timer control register  */
asm("    LDI      00000062h,IOF");         /*set IOF low to reset AIC    */
for (loop = 0; loop < 90; loop++);         /*keep IOF low for a while    */
PBASE[0x42] = 0x00000131;                  /*set xmit port control       */
PBASE[0x43] = 0x00000131;                  /*set receive port control    */
PBASE[0x40] = 0x0E970300;                  /*set serial port global reg  */
PBASE[0x48] = 0x00000000;                  /*clear xmit register         */
asm("    OR       00000006h,IOF");         /*set IOF high to enable AIC  */
for (loop = 0; loop < 4; loop++)           /*loop to configure AIC       */
{
 TWAIT;                                    /*wait till XMIT buffer clear */
 PBASE[0x48] = 0x3;                        /*enable secondary comm       */
 TWAIT;                                    /*wait till XMIT buffer clear */
 PBASE[0x48] = AICSEC[loop];               /*secondary command for SP0   */
}
}

void AICSET_I()                            /*configure AIC, enable TINTO */
{
 AICSET();                                 /*function to configure AIC   */
 asm("    LDI      00000000h,IF");         /*clear IF Register           */
 asm("    OR       00000010h,IE");         /*enable EXINT0 CPU interrupt */
 asm("    OR       00002000h,ST");         /*global interrupt enable     */
}

int UPDATE_SAMPLE(int output)              /*function to update sample   */
{
 int input;                                /*declare local variables     */
 TWAIT;                                    /*wait till XMIT buffer clear */
 PBASE[0x48] = output << 2;                /*left shift and output sample*/
 input = PBASE[0x4C] << 16 >> 18;          /*input sample and sign extend*/
 return(input);                            /*return new sample           */
}
```

File – EQ.CMD –

```
/*EQ.CMD - COMMAND FILE FOR LINKING          */
-c                  /*USING C CONVENTION         */
-stack 0x100
-heap  0x100
EQ.obj           /*MAIN PROGRAM                */
vec_dsk1.obj       /*INSTALLS INTERRUPT VECTORS*/
-o EQ.out        /*LINKED COFF OUTPUT FILE    */
-l rts30.lib       /*RUN-TIME LIBRARY SUPPORT  */



MEMORY
{
   SRAM : org=0x00100000, len=0x8000        /* EXTERNAL SRAM
*/
   RAMS : org=0x00809800, len=0x0002        /* BOOT STACK
*/
   RAM0 : org=0x00809802, len=0x03FE        /* INTERNAL BLK 0
*/
   RAM1 : org=0x00809C00, len=0x03C4        /* INTERNAL BLK 1
*/
   VECS : org=0x00809FC4, len=0x0006        /* VECTORS
*/
   VECS1: org=0x00809FCA, len=0x0037        /* START OF TINT1 VECTOR
*/
}

SECTIONS
{
  .text:    {} > RAM0       /*CODE                          */
  .cinit:   {} > RAM1       /*INITIALIZATION TABLES         */
  .stack:   {} > RAM1       /*SYSTEM STACK                  */
  .bss:     {} > RAM1       /*BSS SECTION                   */
  vecs:     {} > VECS       /*VECTOR SECTION                */
  vec1:     {} > VECS1
}
```

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

entity centers is
port (    sel : in std_logic_vector(2 downto 0); --สัญญาณควบคุมจากดิปสวิชซ์ 3 bit
          clk : in std_logic;
          out0,out1,out2,out3 : out std_logic;
          add0,add1,add2,add3 : out std_logic); --สัญญาณ Common ของ 7 segmentทั้ง 4 ตัว
end centers;

architecture rtl_cens of centers is
signal state:std_logic_vector(1 downto 0);
signal state_r:std_logic_vector(1 downto 0);
signal addr:std_logic_vector(1 downto 0);
signal output:std_logic_vector(3 downto 0);
begin
process (clk)
begin
 if (clk ='1') and clk'event then
  if state = "00" then
          state_r <= "01";
  elsif state = "01" then
          state_r <= "10";
  elsif state = "10" then
          state_r <= "11";
  elsif state = "11" then
          state_r <= "00";
  end if;
 end if;
 end process;
```

```
data_steering: process (clk)
begin
 if (clk ='1') and clk'event then
--ความถี่กลาง 80 Hz
  if sel = "000" then
   if state_r ="00" then
            output <="0000";
            addr <= "11";
            state <="00";
   elsif state_r ="01" then
     output <="0000";
            addr <= "10";
            state <="01";
   elsif state_r ="10" then
            output <="1000";
            addr <= "01";
            state <="10";
   elsif state_r ="11" then
            output <="0000";
            addr <= "00";
            state <="11";
   end if;
 --ความถี่กลาง 160 Hz
  elsif sel = "001" then
   if state_r ="00" then
            output <="0000";
            addr <= "11";
            state <="00";
    elsif state_r ="01" then
      output <="0001";
            addr <= "10";
            state <="01";
```

```
  elsif state_r ="10" then
            output <="0110";
            addr <= "01";
            state <="10";
  elsif state_r ="11" then
            output <="0000";
            addr <= "00";
            state <="11";
  end if;
--ความถี่กลาง 320 Hz
 elsif sel = "010" then
  if state_r ="00" then
            output <="0000";
            addr <= "11";
            state <="00";
  elsif state_r ="01" then
    output <="0011";
            addr <= "10";
            state <="01";
  elsif state_r ="10" then
            output <="0010";
            addr <= "01";
            state <="10";
  elsif state_r ="11" then
            output <="0000";
            addr <= "00";
            state <="11";
  end if;
--ความถี่กลาง 640 Hz
 elsif sel = "011" then
  if state_r ="00" then
            output <="0000";
```

```
            addr <= "11";
            state <="00";
    elsif state_r ="01" then
      output <="0110";
            addr <= "10";
            state <="01";
    elsif state_r ="10" then
            output <="0100";
            addr <= "01";
            state <="10";
    elsif state_r ="11" then
            output <="0000";
            addr <= "00";
            state <="11";
    end if;
--ความถี่กลาง 1280 Hz
  elsif sel = "100" then
    if state_r ="00" then
            output <="0001";
            addr <= "11";
            state <="00";
    elsif state_r ="01" then
      output <="0010";
            addr <= "10";
            state <="01";
    elsif state_r ="10" then
            output <="1000";
            addr <= "01";
            state <="10";
    elsif state_r ="11" then
            output <="0000";
            addr <= "00";
```

```
            state <="11";
  end if;
--ความถี่กลาง 2560 Hz
 elsif sel = "101" then
  if state_r ="00" then
            output <="0010";
            addr <= "11";
            state <="00";
  elsif state_r ="01" then
    output <="0101";
            addr <= "10";
            state <="01";
  elsif state_r ="10" then
            output <="0110";
            addr <= "01";
            state <="10";
  elsif state_r ="11" then
            output <="0000";
            addr <= "00";
            state <="11";
  end if;
 end if;
end if;
end process;

process (addr)
begin
case addr is
          when "00"   =>  add3 <= '1';
                          add2 <= '1';
                           add1 <= '1';
                          add0 <= '0';
```

```
        when "01"   =>  add3 <= '1';
                        add2 <= '1';
                        add1 <= '0';
                        add0 <= '1';
        when "10"   =>  add3 <= '1';
                        add2 <= '0';
                        add1 <= '1';
                        add0 <= '1';
        when "11"   =>  add3 <= '0';
                        add2 <= '1';
                        add1 <= '1';
                        add0 <= '1';
        when others => null;
end case;
end process;

process (output)
begin
case output is
        when "0000"   => out3 <= '0';
                         out2 <= '0';
                         out1 <= '0';
                         out0 <= '0';
        when "0001"   => out3 <= '0';
                         out2 <= '0';
                         out1 <= '0';
                         out0 <= '1';
    when "0010"   => out3 <= '0';
                         out2 <= '0';
                         out1 <= '1';
                         out0 <= '0';
    when "0011"   => out3 <= '0';
```

```
                    out2 <= '0';
                    out1 <= '1';
                    out0 <= '1';
        when "0100"  => out3 <= '0';
                    out2 <= '1';
                    out1 <= '0';
                    out0 <= '0';
        when "0101"  => out3 <= '0';
                    out2 <= '1';
                    out1 <= '0';
                    out0 <= '1';
        when "0110"  => out3 <= '0';
                    out2 <= '1';
                    out1 <= '1';
                    out0 <= '0';
    when "1000"  => out3 <= '1';
                    out2 <= '0';
                    out1 <= '0';
                    out0 <= '0';
        when others => null;
end case;
end process;
end rtl_cens;
--โปรแกรมสั่ง อ่าน,เขียนของMemoryและควบคุมBuffer
library ieee;
use ieee.std_logic_1164.all;

entity rw is
port ( sel  : in std_logic; --r/w สัญญาณเลือกว่าจะอ่านหรือเขียนข้อมูล
       en   : out std_logic; --enable compare
       oe   : out std_logic; --enable buffer541
       we       : out std_logic); --ram part
```

```
end rw;

architecture arc_rw of rw is
begin
process (sel)
begin
case sel is
 when '0' => we <= '0'; --write
                    en <= '0';
                    oe <= '0';
 when '1' => oe <= '1'; --read
                    we <= '1';
                    en <= '1';
when others => null;
end case;
end process;
end arc_rw;


--โปรแกรมควบคุมFader โดยการนำค่าที่ได้จากMemoryกับFaderมาเปรียบเทียบกัน
library ieee;
use ieee.std_logic_1164.all;

entity compare is
port ( en : in std_logic;
          clk : in std_logic;
           A : in std_logic_vector(7 downto 0);
           B : in std_logic_vector(7 downto 0);
           Q : out std_logic_vector(1 downto 0));
end compare;

architecture arc_comp of compare is
```

```
begin
process (clk)
begin
if clk = '1' and clk'event then
 if en = '1' then --begin compare
  if a > b then --A fader, B ram
    Q <= "10"; --turn up
  elsif a < b then
    Q <= "01";  --turn down
  else
    Q <= "00";  --stop
  end if;
 else --not compare
    Q <="00";
end if;
end if;
end process;
end arc_comp;
```

--โปรแกรมแปลงข้อมูลที่ได้จาก A/D เป็นข้อมูลที่ DSP นำไปใช้ได้

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

entity b2b is
port( datai : in std_logic_vector(7 downto 0);
            QV     : in std_logic;
            datao : out std_logic_vector(9 downto 0);
       BB : out std_logic);
end b2b;
```

```
architecture arc_b2b of b2b is
begin

process (datai,QV)
variable data1,data2:std_logic_vector(7 downto 0);
variable A,C:integer;
variable B:unsigned(7 downto 0);
variable D:std_logic_vector(9 downto 0);
begin
if QV = '0' then –V ขนาด Magnitude มีค่า-1000 ถึง +1000
 if datai < "10000000" then --negative
  data1 := not datai;
  data1(7):='0';
   B := unsigned(data1);
   A := conv_integer(B);
   C := A*8;
   if C > 1000 then
            C := 1000;
         elsif c < 100 then
            C := 100;
   end if;
   D := conv_std_logic_vector(C,10);
   datao <= D;
   BB <= '1';
 elsif datai = "10000000" then --zero
         datao <= "0001100100";
 elsif datai > "10000000" then --positive
   data1 := datai;
   data1(7) := '0';
   B := unsigned(data1);
   A := conv_integer(B);
   C := A*8;
```

```
    if C > 1000 then
            C := 1000;
            elsif c < 100 then
              C := 100;
    end if;
    D := conv_std_logic_vector(C,10);
    datao <= D; -- DSPต้องนำค่าที่ได้ไปคูณ0.01
            BB <= '0'; --sign bit
  end if;
elsif QV ='1' then --Q
    data2 := datai;
    if data2 < "00001010" then  --10
            data2 :="00001010";
    elsif data2 > "11001000" then --200
            data2 := "11001000";
    end if;
   datao <= data2; -- DSPต้องนำค่าที่ได้ไปคูณ0.1
   BB <= '0';
end if;
end process;
end arc_b2b;
--โปรแกรมชี้ตำแหน่งแอดเดรสให้แก่Memory
library ieee;
use ieee.std_logic_1164.all;


entity add is
port ( center : in std_logic_vector(2 downto 0); --switch center
              QV    : in std_logic;
              add   : out std_logic_vector(3 downto 0));
end add;


architecture arc_add of add is
```

```
signal centerQV:std_logic_vector(3 downto 0);
begin
process (center,QV)
begin
centerQV <= QV & CENTER;
end process;

process (centerQV)
begin
case centerQV is
        when "0000" => add <="0000"; --80          0
        when "0001" => add <="0011";       --      3
        when "0010" => add <="0101";       --      5
        when "0011" => add <="0110";       --      6
        when "0100" => add <="1000";       --      8
        when "0101" => add <="1011";       --      11
        when "1000" => add <="1101";       --      13
        when "1001" => add <="1110";       --      14
        when "1010" => add <="0111";       --      7
        when "1011" => add <="0100";       --      4
        when "1100" => add <="0010";       --      2
        when "1101" => add <="1010";       --      10
        when others =>null;
end case;
end process;
end arc_add;
```

--โปรแกรมสร้างสัญญาณ Interrupt ทำงานเมื่อมีการเปลี่ยนแปลงที่Fader

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

```
entity interup is
port ( clk : in std_logic;
        fed : in std_logic_vector (7 downto 0);
        int : out std_logic;
        Z : out std_logic);
end interup;

architecture arc_int of interup is
begin
process (clk) --Feder
variable Fed_B:std_logic_vector(7 downto 0);
begin
if clk ='1' and clk'event then
 if Fed /= Fed_b then
   int <= '0';
   Z <= '1';
 else
   Z <= '0';  --Z
 end if;
Fed_b := Fed;
end if;
end process;
end arc_int;
```

หนังสืออ้างอิง

1. RULPH CHASSAING, " Digital Signal Processing Laboratory Experiments Using C and the TMS320C31 DSK", John WILEY&SON, INC, 1999

2. DOUGLAS L. PERRY, "VHDL",1993

3."IEEE Standard VHDL Language Reference Manual " ,2000 Edition, DASC

เวบไซต์อ้างอิง

www.ti.com

www.icmaster.com

www.national.com