การวัดคลื่นไฟฟ้าหัวใจ บันทึกข้อมูลและแสดงผล

(ECG Measuring Recording and Displaying by Computer)

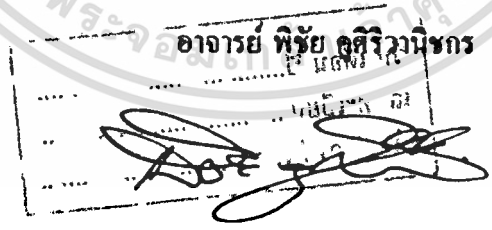
โดย

นาย ชาญวุฒิ มิคาระเศรษฐ์ 34102093

นาย วัฒนพงศ์ สุวรรณธรรมา 34106312

อาจารย์ที่ปรึกษา

อาจารย์ พิรัย คูศิริวานิชกร



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาในหลักสูตร

ปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ประจำปีการศึกษา 2537

การวัดคลื่นไฟฟ้าหัวใจ บันทึกข้อมูลและแสดงผล

(ECG Measuring Recording and Displaying by Computer)

นาย ชาญวุฒิ มิคาระเศรษฐ์ 34102093

นาย วัฒนพงศ์ สุวรรณธรรมา 34106312

อาจารย์ที่ปรึกษา

อาจารย์ พิชัย ภูศิริวานิชกร

ปีการศึกษา 2537

บทคัดย่อ

คลื่นไฟฟ้าหัวใจ เป็นคลื่นที่มีความสำคัญทางการแพทย์มากคลื่นหนึ่ง เพราะแพทย์สามารถวินิจฉัยความผิดปกติของหัวใจของผู้ป่วยที่เกิดขึ้นโดยวิเคราะห์คลื่นไฟฟ้าหัวใจนี้ ดังนั้นจึงเป็นการเหมาะสมที่จะพัฒนาการวัดคลื่นไฟฟ้าหัวใจนี้ โดยการเชื่อมต่อเข้ากับคอมพิวเตอร์ ด้วยการเชื่อมอุปกรณ์การวัดเข้ากับคอมพิวเตอร์ โดยมีการ์ด PA-MA(-H) A/D converter เป็นตัวควบคุมการส่งข้อมูล และพัฒนาเพื่อที่จะสามารถเก็บและ แสดงรูปคลื่นไฟฟ้านี้ทั้งในขณะที่ทำการวัดคลื่นนี้กับผู้ป่วย และยังสามารถที่จะเก็บไว้เป็นข้อมูลของคนไข้ที่สามารถนำมาตรวจสอบและ แสดงรูปคลื่นเพื่อทำการวินิจฉัยในภายหลังได้

ECG Measuring Recording and Displaying by Computer

Chanwut Mikarasert 34102093

Wattanapong Suwantunma 34106312

Advisor:

Mr. Pichai Koosirivanichakorn

Project I , II . 1994

Abstract

Electrocardiogram (ECG) is importance in medical technology. Because doctor can diagnose heart condition of a patient by analysing ECG. So it is suitable to develop the ECG measuring by interfacing it with computer through PA-MA12(-H) card. The card is the a controller of data transmission and is developed and display ECG. Additionally the data can be stored so that it can be checked and displayed for later diagnostic.

สารบัญ

คำนำ

ปริญญานิพนธ์นี้จะอธิบายถึง วัตถุประสงค์ , หลักการ , ทฤษฎี , แนวคิด ของการวัดคลื่นไฟฟ้าหัวใจ ผู้จัดทำหวังว่า ปริญญานิพนธ์นี้จะมีประโยชน์ต่อผู้ศึกษาค้นคว้าทางด้านนี้ หากมีข้อผิดพลาดประการใด ผู้จัดทำขอรับผิดชอบความผิดพลาดเหล่านั้นทั้งหมด ปริญญานิพนธ์นี้ได้รับคำปรึกษา , ความช่วยเหลือ และกำลังใจ จาก อาจารย์ที่ปรึกษา , อาจารย์ภาควิชาอิเล็กทรอนิกส์ , รุ่นพี่ , เพื่อน และ รุ่นน้อง ทุกท่าน ผู้จัดทำขอขอบคุณทุกท่าน และ ความดีของปริญญานิพนธ์นี้ท่านเหล่านี้มีส่วนร่วมด้วยทั้งสิ้น.

ผู้จัดทำ

บทที่ 1 บทนำ

คลื่นไฟฟ้าหัวใจเป็นสัญญาณที่มีศักดาไฟฟ้าที่ต่ำมาก (หน่วย เป็น mV) โดยการวัดคลื่นไฟฟ้าหัวใจนี้ เราสามารถวัดได้ โดยใช้ แผ่น electrode ไปติดที่ร่างกายตามตำแหน่งที่ถูกต้อง จากนั้นเรา ต้องนำสัญญาณที่ได้นี้ไปทำการขยายโดยผ่านวงจรขยาย (Amplifier) โดยลักษณะของวงจรขยายที่จะใช้ในการวัดคลื่นไฟฟ้าหัวใจนี้จะต้องมีลักษณะพิเศษดังนี้ คือ

1 ลักษณะของเครื่องมือวัด

1.1 อัตราการขยายสูง

1.2 ค่า CMRR สูง

1.3 ค่า Input Impredance สูง

1.4 มีการตอบสนองต่อความถี่ต่ำได้ดี

นอกจากนี้ วงจรขยายในภาคแรกจะต้องเป็นวงจรขยายที่เรียกว่า Floating Amplifier คือ เป็นวงจรที่มีสาย common ไม่ได้ต่อที่แท่นเครื่องโดยตรง เพื่อป้องกันอันตรายของผู้ป่วยที่จะเกิดขึ้นโดยใช้วงจร Opto Couple

การนำสัญญาณที่เป็นสัญญาณ analog ไปประมวลผลใน computer จะต้องผ่านวงจรที่เรียกว่า analog to digital converter โดยในโครงการนี้เราใช้การ์ด PA-MA12(-H) ซึ่งเป็นการ์ดที่มีความสามารถในการแปลงสัญญาณจาก analog ไปเป็น digital ด้วยความเร็วสูง โดยที่เราจะต้อง ทำการ Interface การ์ดนี้เข้ากับ computer ซึ่งจะกล่าวพอสังเขปดังนี้

2.การ Installation การ์ด

2.1 ทาง Hardware จะต้อง set ค่าต่าง ๆ ที่ การ์ดต้องมีการกำหนดก่อนที่จะนำการ์ดมาติดตั้ง เข้ากับ computer โดยจะกระทำดังนี้

- การเลือก I/O port address
- การเลือก Interrupt channel
- การเลือก DMA channel
- การเลือก data length
- การเลือก Input type
- การเลือก Input range
- การนำสัญญาณมาต่อเข้ากับตัว connecter

2.2 การเขียนโปรแกรมในการควบคุม data transfer โดยในโครงการนี้เราใช้การเขียน

โปรแกรมโดยใช้ภาษา C เป็นตัวควบคุมโดยเราเลือกการ transfer ข้อมูลแบบ DMA (Direct Memory Access) เพราะมีความสามารถในการ ส่งผ่านข้อมูลได้รวดเร็วโดยการเขียนโปรแกรมมีวิธีการ เขียนดังนี้

การทำ DMA transfer นั้นจะมีตัวควบคุมการทำงานของ DMA transfer โดยการ์ด PA-MA12(-H) จะใช้ตัว DMA controller เป็นตัวควบคุมการทำงานโดยเราจะเขียนโปรแกรมตาม Step ดังนี้

- reset การ์ด PA-MA12(-H) และ disable DMA controller
- กำหนดโปรแกรมการให้ Input เข้าสู่การ์ดตาม channel ที่กำหนดไว้ในโปรแกรม
- โปรแกรมเลือกค่า Sampling Rate
- โปรแกรมเลือกตำแหน่ง memory ในการเก็บข้อมูล
- set จำนวนครั้งในการ transfer ข้อมูล
- ให้ค่าเริ่มต้นกับ DMA mode register
- set ค่า DMA controller Enable
- โปรมแกรมเลือก mode การ transfer แบบ DMA
- เช็คการทำ DMA complete
- disable และ stop DMA โดยการ Reset PA-MA12(-H) และ disable DMA controller

นอกจากนี้แล้วเรายังต้องเขียนโปรแกรมในการนำข้อมูลที่เก็บไว้ใน memory ไปเก็บไว้เป็น file และเราจะนำข้อมูลที่ได้นี้มาแสดงผลเป็นรูปกราฟ เพื่อที่จะได้นำผลของสัญญาณที่เกิดขึ้นมาใช้วิเคราะห์ต่อไป

สำหรับรายละเอียดของการทำงานทั้งหมดนี้ คณะผู้จัดทำจะได้กล่าวไว้ในบทต่อไปโดยละเอียด

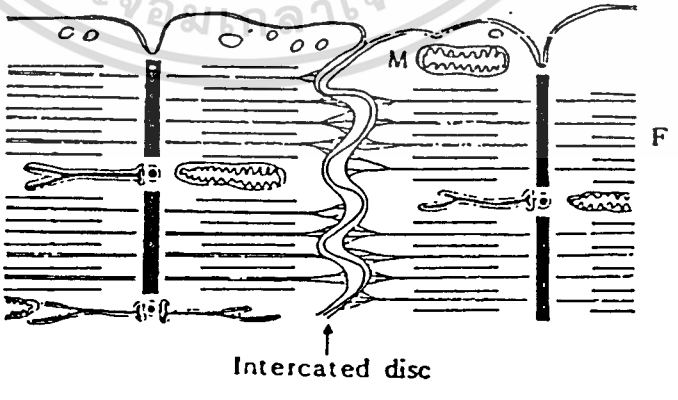
บทที่ 2 การวัดคลื่นไฟฟ้าหัวใจ

กราฟไฟฟ้าหัวใจ หรือ electrocardiogram (ECG) มักบันทึกลงบนกระดาษที่เคลื่อนที่ไปในทิศทางหนึ่ง ด้วยเข็มของเครื่องบันทึก ซึ่งเคลื่อนที่ขึ้นลง มีลักษณะเป็นขีดขึ้นลงสลับกันและเกิดขึ้นอย่างต่อเนื่อง

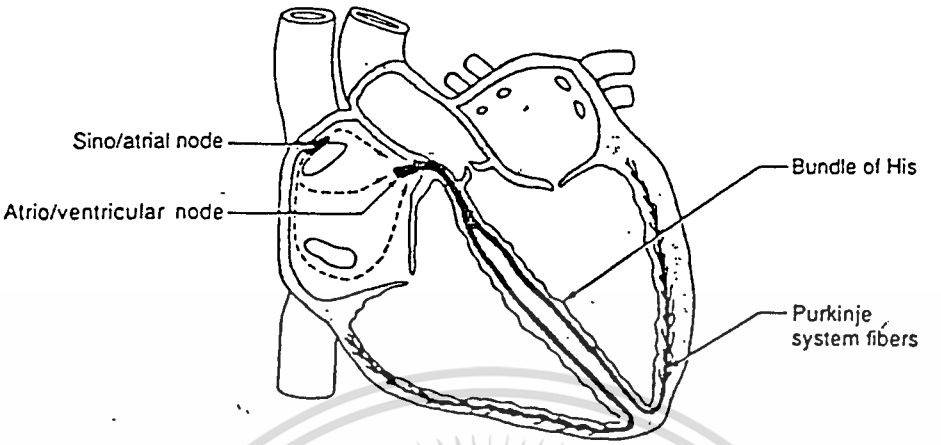
การเคลื่อนที่ของเข็มเกิดจากพลังไฟฟ้าที่เป็นผลมาจากความแตกต่างของศักย์ไฟฟ้า (electrical potencial difference) บนผิวกายของผู้ที่เราบันทึก ECG ศักย์ไฟฟ้าตามจุดต่าง ๆ บนผิวกายได้มาจากการเปลี่ยนแปลงสถานะของเซลล์กล้ามเนื้อหัวใจ

2.1 การกำเนิดของพลังไฟฟ้าจากหัวใจ

กล้ามเนื้อหัวใจประกอบไปด้วยเซลล์จำนวนมากมาย ภายในเซลล์มีเส้นใย actin , myocin และ ส่วนประกอบอื่น ๆ เป็นเซลล์ที่มีรูปร่างยาว และมีส่วนของผนังเชื่อมต่อกับเซลล์ข้างเคียงแบบปลายต่อปลาย ผนังเซลล์ส่วนนี้เรียกว่า intercalated disc ซึ่งมีความต้านทานการนำไฟฟ้าต่ำมาก ไฟฟ้าจากเซลล์หนึ่งจึงผ่านไปอีกเซลล์หนึ่งได้ง่าย (รูปที่ 2.1.1) เอเตรียม และ เวนตริเคิลแยกออกจากกันเป็นเนื้อเยื่อ 2 กลุ่มโดย fibrous tissue ของ atrioventricular ring มีเนื้อเยื่อพิเศษทำหน้าที่นำไฟฟ้าระหว่าง 2 ส่วนนี้ เรียกว่า atrioventricular condyation system (AV condystion system) ซึ่งประกอบด้วย atrioventricular node (AV node) , Bundle of his (His bundle), bundle branch และ purkinje fibre (Purkinje System) (รูปที่ 2.1.2)



รูปที่ 2.1.1 ภาพวาดแสดงส่วนปลายของ 2 เซลล์ของกล้ามเนื้อหัวใจที่บริเวณ intercalated disc M = mitochondria , F = muscle filament



รูปที่ 2.1.2 แสดงตำแหน่งของกล้ามเนื้อที่ส่งผ่านไฟฟ้าสำหรับกระตุ้นกล้ามเนื้อหัวใจ

การทำงานของหัวใจจะเกี่ยวเนื่องกับศักคาไฟฟ้า ซึ่งเราสามารถตรวจรับได้โดยใช้ ECG electrode ติดบนร่างกาย ทางผ่านของศักคาไฟฟ้านี้แสดงไว้ในรูปที่ 2

ต่อไปนี้เป็นขั้นตอนในการเกิดสัญญาณ ECG

2.1.1. การกระตุ้นจะเกิดจากจุด sino/atrial node (S/A node)

กล้ามเนื้อหัวใจจะเป็นแบบที่มีลักษณะต่าง จากร่างกายส่วนอื่น ๆ คือมันจะ เป็นทั้งกล้ามเนื้อและ เหมือนกล้ามเนื้อลาย (sketetal muscle) กล้ามเนื้อลายจะใช้ในการเคลื่อนไหว ซึ่งมีคุณสมบัติคล้ายกับเส้นประสาท คือมีความสามารถมีการกระตุ้นทางไฟฟ้าได้ S/A node เป็นชิ้นของเนื้อเยื่อ หัวใจที่สามารถให้การกระตุ้นได้อย่างแรงและ เป็นเสมือน pacemaker ของหัวใจ ศักคาไฟฟ้าที่เกิดขึ้นและนำไปสู่การบีบตัวของหัวใจนี้จะเกิดจากการทำงานของ S/A node

2..1.2 การบีบตัวของหัวใจห้องบน

ดังที่ทราบมาแล้วว่ากล้ามเนื้อหัวใจสามารถกระตุ้นทางไฟฟ้าได้ เมื่อ S/A node เกิดการ depolarize คลื่นของการกระตุ้นก็จะกระจายไปยังกล้ามเนื้อหัวใจห้องบน (atrial muscle) และมันก็จะบีบตัวส่งแรงของโลหิตไปยังหัวใจห้องล่าง เพื่อให้ห้องล่างมีโลหิตบรรจุเต็ม

2.1.3. การกระตุ้นของ Atrio / ventricular node (A/V node)

ณ บริเวณตอนใต้ของหัวใจห้องบนขวา (right atrium) จะเป็นส่วนของเนื้อเยื่อหัวใจอีกอันหนึ่งที่สามารถเกิดการกระตุ้นได้อย่างสูง คือ A/V node เมื่อคลื่นของการกระตุ้นจากหัวใจห้องบนมาถึงที่จุด A/V node มันก็จะถูก depolarize อีก และการ depolarize นี้จะแยกไปยังแขนงมัดของโปรตีนชนิดหนึ่งที่เรียกว่า bundle of his

2.1.4. การแพร่กระจายลงสู่ด้านล่างของ bundle of his

นั่นคือการไหลผ่านของกระแสกระตุ้นจากหัวใจห้องล่าง และจะผ่านการกระตุ้นลงไปเริ่มทางด้านล่างของหัวใจห้องล่าง การบีบตัวก็จะเริ่มจากส่วนล่างของหัวใจห้องล่าง

2.1.5. Purkinje system fibers

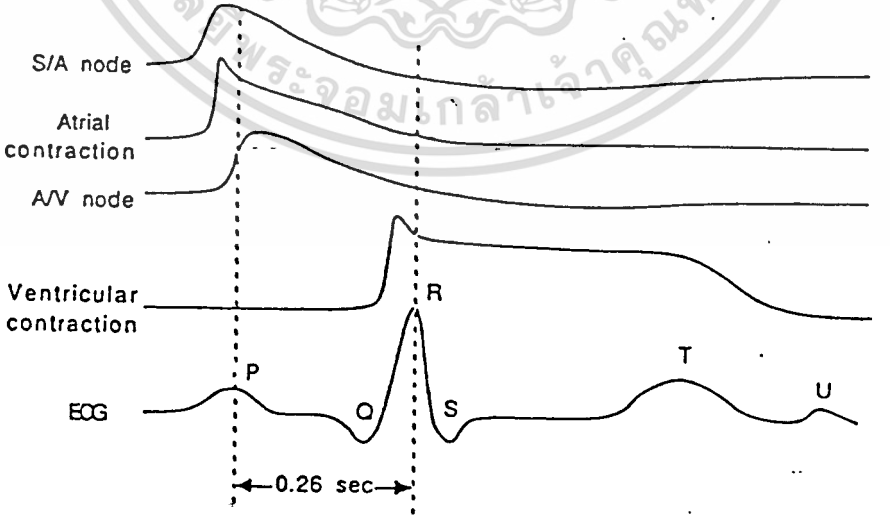
เป็นส่วนต่อเชื่อมระหว่างการกระจายลงมาทางด้านล่างของ bundle of his และการบีบตัวของหัวใจห้องล่าง ซึ่ง Purkinje system fibers

2.1.6. การบีบตัวของหัวใจห้องล่าง

เป็นเฟสสุดท้ายของการทำงานของหัวใจ (cardiac cycle) หัวใจห้องล่างจะเป็นเครื่องสูบฉีดโลหิตในระบบหลอดโลหิตหัวใจ ทำหน้าที่ส่งโลหิตไปยังปอด (จากหัวใจห้องล่างขวา) และส่งไปเลี้ยงอวัยวะของร่างกาย (จากหัวใจห้องล่างซ้าย) ดังนั้นโหลดของหัวใจห้องล่างซ้ายจะมากกว่าห้องล่างขวา ผนังของหัวใจห้องล่างซ้ายก็จะมีขนาดใหญ่และแข็งแรงมากกว่าห้องล่างขวา

2.2 ศักดารวมในรอบการทำงานของหัวใจ

ศักดาไฟฟ้ารวมที่เกิดขึ้นในรอบการทำงานของหัวใจนั้นสามารถวัดได้โดยใช้อิเลคโทรดติดที่บริเวณผิวผนังของร่างกายได้ทุก ๆ แห่ง รูปที่ 2.2 แสดงให้เห็นถึงไดอะแกรมของศักดาในแต่ละเฟสของการทำงานของหัวใจ กราฟเส้นล่างสุดจะเป็นผลของการวัดที่ผิวหนังโดยใช้อิเลคโทรดซึ่งเป็นผลรวมของการทำงานทั้งหมดใน 1 รอบการทำงานของหัวใจ

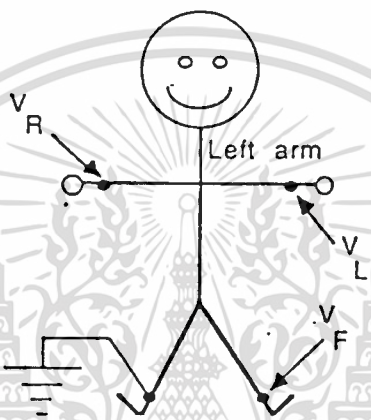


รูปที่ 2.2 แสดงส่วนประกอบของคลื่นไฟฟ้าหัวใจ

2.3 ECG อิเลคโทรดและการตรวจวัด

2.3.1 Standard Limb Leads

สัญญาณ ECG นั้นมีขนาดใหญ่เพียงพอที่จะสามารถวัด ณ จุดใด ๆ ก็ได้บนร่างกาย แบบมาตรฐานนั้นได้มาจากการวัด โดยการเปรียบเทียบหลาย ๆ ตำแหน่ง standard limb จะใช้ในการวัด ECG จากบริเวณแขนและขาของผู้ป่วย โดยใช้อิเลคโทรดแบบ silver-silver chloride จำนวน 4 อัน คือ ที่ข้อมือและข้อเท้า ดังรูปที่ 2.3



รูปที่ 2.3 แสดงตำแหน่งการวัดอิเลคโทรดในการวัดแบบ standard limb leads

เมื่อเราติดอิเลคโทรดในตำแหน่งที่ถูกต้องแล้ว เราก็จะมาเลือกว่าจะใช้คู่ไหนมาต่อกับ input ของวงจร differential amplifier การต่อมาตรฐานของ standard limb leads มีดังนี้

$$\text{Lead 1} \quad V_L^+ - V_R$$

$$\text{Lead 2} \quad V_F^+ - V_R$$

$$\text{Lead 3} \quad V_F^+ - V_L$$

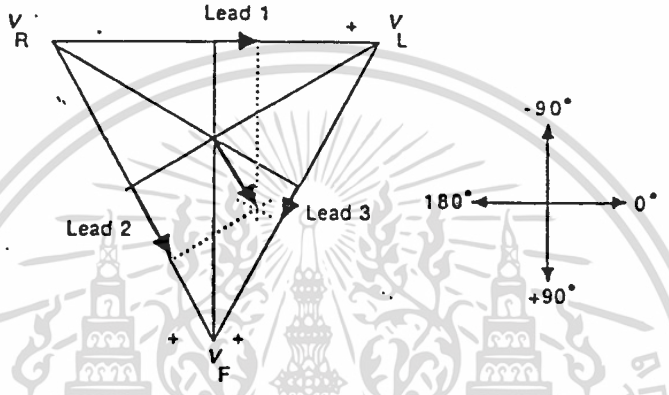
เครื่องหมาย + ของ superscript หมายถึง Noninverting input ของวงจร คำว่า "lead" ในที่นี้ไม่ได้หมายถึงสายสัญญาณ แต่หมายถึงลักษณะการต่อสัญญาณ input ซึ่งเป็นศัพท์เฉพาะทางของเรื่องการวัดคลื่นไฟฟ้าหัวใจ

2.4 Einthoven 's Triangle

อิเลคโทรดจะถูกยึดติดไว้กับข้อมือและข้อเท้า เนื่องจากหัวใจได้สร้างศักดาแรงดันที่ไหลผ่านไปตลอดร่างกายและแขน ขา ดังนั้นตำแหน่งของ limb นี้ถูกเขียนให้เป็นลักษณะสามเหลี่ยมในขณะนี้จะสมมุติให้หัวใจวางอยู่ตำแหน่งกึ่งกลางพอดี ในระหว่าง cardiac cycle) สัญญาณ ECG

สามารถถูกแทนให้เป็นเสมือนขั้วหัวใจสองขั้ว (cardiac dipole ซึ่งมีขนาดและทิศทางที่หันเหไปมาได้ตลอดช่วงการทำงานของหัวใจ dipole นี้เราเรียกว่า cardiac vector Einthoven 's triangle และ cardiac vector สามารถแสดงได้ในรูปที่ 2.4

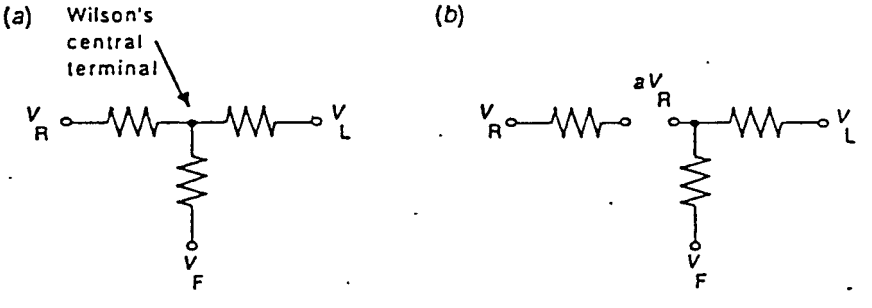
เนื่องจากทางการแพทย์ต้องการวัด cadiac vector ซึ่งมีทิศทางการไหลด้วยดังนั้นจะต้องมีค่ามาตรฐานเปรียบเทียบดังแสดงดังรูปที่ 2.4 เช่นเดียวกัน



รูปที่ 2.4 Einthoven 's triangle and the cardiac vector

2.5 Augmented Limb Leads (Leads เพิ่มเติม)

จากการใช้เทคนิคในการวัดต่าง ๆ ที่ดีขึ้น (เช่นการต่อ ground ของคนไข้และการใช้วงจร differential amplifier ที่มี CMRR ที่ดี) อัตราส่วน Signal-to-noise ของสัญญาณ ECG ที่วัดได้นั้นจะดีหรือไม่ก็ขึ้นอยู่กับลักษณะการใช้อิเลคโทรดด้วย Augmented limb leads จะมีตำแหน่งการวางอิเลคโทรดเหมือนกับ standard limb leads แต่อิเลคโทรดจะต่อผ่านตัว resistor ไปยังขั้วอิเลคโทรดดังรูปที่ 2.5 จุดร่วมนี้เราเรียกว่า Wilson's central terminal



รูปที่ 2.5 The augmented limb leads

ในการวัด augmented lead ตัวอย่างเช่น VR. resistor ที่ต่อจากแขนขวาจะถูกถอดออกจาก node ร่วมและบันทึกศักดาระหว่าง resistor ที่แขนขวา และ node เทคนิคนี้จะช่วยขจัดสัญญาณรบกวน (noise) และเพิ่มขนาดของสัญญาณที่รับมาจากแขนขวาโดยแฟคเตอร์ขนาด 1.5 เท่า ซึ่งแสดงให้เข้าใจดังนี้

จากรูป สามารถคำนวณหา VR และ a VR ได้ดังนี้

$$aV_R = V_R - 0.5(V_L + V_F)$$

$$2aV_R = 2V_R - (V_L + V_F)$$

ซึ่งผลรวมของ Voltage รอบ Einthoven 's triangle จะต้อง = 0 (จาก Kirchoff 's voltage law)

$$V_R + V_L + V_F = 0$$

หรือ

$$V_R = - (V_L + V_F)$$

และ

$$2aV_R = 2V_R + V_R = 3V_R$$

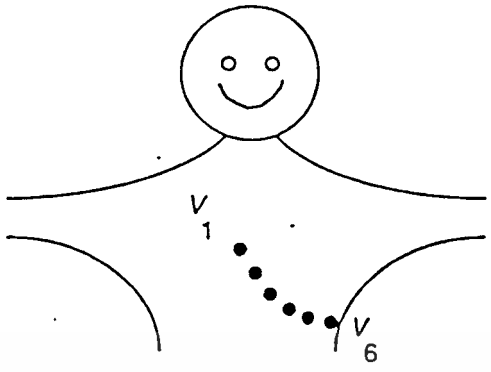
หรือ

$$aV_R = 1.5\ V_R$$

ดังนั้น augmented limb leads สามารถที่จะให้อัตราส่วน Signal / Noise ที่ดีขึ้น

2.6 Unipolar Eletrode Configuration

ในการวิเคราะห์ทางการแพทย์บางอย่างนั้น การใช้วิธี standard limb leads ในการวัดสัญญาณ ECG นั้น ไม่สามารถให้ข้อมูลในการวิเคราะห์ได้เพียงพอ ดังนั้นจำเป็นที่จะต้องใช้ลักษณะการวางอิเลคโทรดแบบ unipolar ซึ่งแสดงได้ในรูปที่ 7 เป็นการใช้ standard limb lead ต่อร่วมกับอิเลคโทรด 6 ชิ้น บริเวณสันหน้าอกไปจนถึงใต้แขนวิธีนี้จะทำให้การวิเคราะห์เป็นในลักษณะ 3 มิติ



รูปที่ 2.6 การวางอิเลคโทรค แบบ unipolar electrode

บทที่ 3 เครื่องวัดคลื่นไฟฟ้าหัวใจ

3.1 คุณสมบัติของเครื่องวัดคลื่นไฟฟ้าหัวใจ

เครื่องวัดคลื่นไฟฟ้าหัวใจ หรือ electrocardiogram (ECG) ต้องมีคุณสมบัติดังต่อไปนี้ คือ

3.1.1. อัตราการขยายสูง

การที่ศักดาไฟฟ้าจากหัวใจมีค่าเพียงประมาณ 0.1 - 5 มิลลิโวลท์ และอยู่ในช่วงความถี่ 0.2-100 Hz โดยการรับสัญญาณจากอิเลคโตรดที่ติดอยู่บนผิวหนัง จึงต้องใช้วงจรขยายที่มีอัตราการขยายสูงมากตลอดช่วงความถี่ดังกล่าว เพื่อให้ได้สัญญาณที่มีความแรงพอ เพื่อแสดงผลการวัด ออกมาโดยไม่มีความผิดเพี้ยนของสัญญาณ ซึ่งในการสร้างเครื่องขยายให้มีอัตราการขยายสูงมาก ๆ นี้ จะมีปัญหาเกิดขึ้นคือสัญญาณรบกวน โดยเฉพาะอย่างยิ่งสัญญาณจากสายไฟกระแสสลับหรือไฟบ้าน ซึ่งมีความถี่ 50 Hz และความถี่ของไฟฟ้ากระแสสลับนี้อยู่ในช่วงของความถี่ของคลื่นไฟฟ้าหัวใจ ดังนั้นในการสร้างเครื่องวัดคลื่นไฟฟ้าหัวใจจึงต้องแก้ปัญหาเรื่องสัญญาณรบกวนนี้ โดยใช้วงจรขยายแบบดิฟเฟอร์เรนเชียล เนื่องจากมีคุณสมบัติที่สำคัญคือ จะขยายเฉพาะสัญญาณที่เป็น differential mode ส่วนสัญญาณที่เป็น common mode จะไม่ถูกขยายให้ออกไปที่เอาท์พุท และเนื่องจากสัญญาณรบกวนส่วนมากจะเข้าไปในวงจรขยายในลักษณะ common mode ดังนั้นสัญญาณรบกวนจึงไม่ไปปรากฏที่เอาท์พุทของวงจรขยาย

3.1.2. ค่า CMRR (Common Mode Rejection Ratio) สูง

Common Mode Rejection Ratio คือ อัตราส่วนระหว่างกำลังขยายของสัญญาณที่เป็น differential mode ต่อกำลังขยายของสัญญาณที่เป็น common mode ค่า CMRR นี้ เป็นคุณสมบัติอย่างหนึ่งของวงจรขยายความแตกต่างที่จะสามารถกำจัดสัญญาณรบกวนได้ คือต้องมีอัตราขยายของสัญญาณ differential mode สูง และมีอัตราขยายของสัญญาณ common mode ต่ำ ทั้งนี้เนื่องจากสัญญาณที่ต้องการขยาย (ECG) จะเข้าไปที่อินพุทในลักษณะของสัญญาณ diferential mode ส่วนสัญญาณรบกวน (50Hz) จะเข้าไปที่อินพุทในลักษณะของสัญญาณ common mode ดังนั้นถ้าค่า CMRR มีค่าสูงย่อมหมายความว่าสัญญาณรบกวนจะมีโอกาสไปปรากฏที่เอาท์พุทได้น้อย แต่ถ้าค่า CMRR มีค่าต่ำ สัญญาณรบกวนจะมีโอกาสไปออกที่เอาท์พุทได้มาก

8.1.3. ค่า Input Impedance สูงมาก

เนื่องจากในการวัดสัญญาณไฟฟ้าจากหัวใจนั้น ไม่ได้วัดที่หัวใจโดยตรง แต่จะใช้ขั้วไฟฟ้า (electrode) ติดที่ผิวหนังบริเวณ แขน ขา และทรวงอก จึงทำให้เกิดความต้านทานขึ้นตรงบริเวณรอยสัมผัสของขั้วไฟฟ้ากับผิวหนัง ซึ่งค่าความต้านทานนี้จะมีค่าสูงตั้งแต่ 0.1- 0.8 เมกกะโอห์ม ดังนั้นวงจรขยายจึงต้องมีค่า Input Impedance สูงมากๆ เมื่อเทียบกับความต้านทานตรงรอยสัมผัสหรือผิวหนัง เพื่อให้สัญญาณที่วัดสูญเสียที่รอยสัมผัสน้อยที่สุด นอกจากนั้นแล้วยังเป็นการเพิ่มค่า CMRR ให้กับวงจรได้ด้วย จึงเป็นการป้องกันการเสียสมดุลย์ของวงจร ซึ่งเกิดจากการที่สัญญาณรบกวนที่เข้ามาในลักษณะสัญญาณ common mode ไม่สามารถกำจัดออกไปได้ และในการวัดค่า CMRR นั้น จะต้องวัดรวมทั้งสายขั้วไฟฟ้าด้วย และค่าความต้านทานของสายขั้วไฟฟ้าแต่ละเส้นนี้จะมีโอกาสเท่ากันได้ยาก ดังนั้นถ้าค่า Input Impedance ของวงจรมีค่าสูงมากๆ เมื่อเทียบกับความต้านทานของสายขั้วไฟฟ้า ความแตกต่างของความต้านทานในสายขั้วไฟฟ้า ก็จะมีผลต่อวงจรน้อยลง

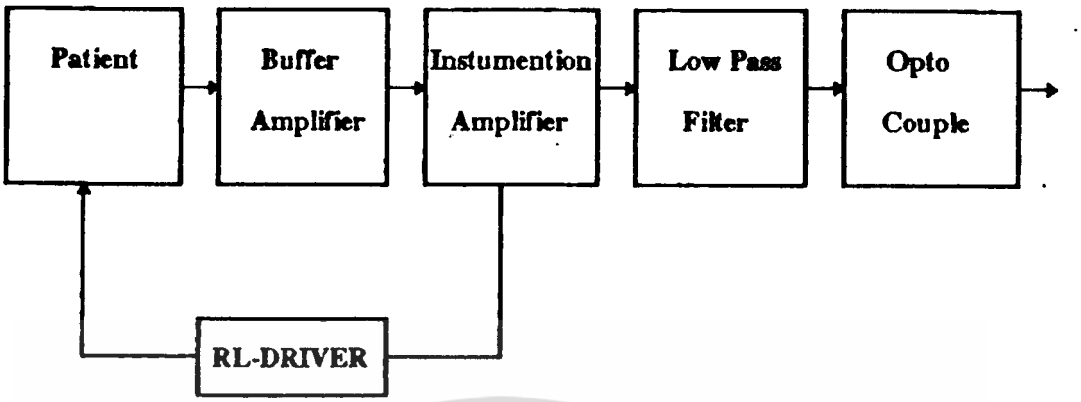
8.1.4. มีการตอบสนองความถี่ต่ำดี

เนื่องจากคลื่นไฟฟ้าหัวใจมีความถี่อยู่ในช่วง 0.2 - 100 Hz ดังนั้นวงจรขยายจะต้องขยายสัญญาณในช่วงความถี่นี้ได้ดี โดยที่สัญญาณไม่ผิดเพี้ยน วงจรที่ใช้จึงต้องเป็นวงจรขยาย ไฟฟ้า ดี. ซี. (D.C. Amplifier)

นอกจากคุณสมบัติดังกล่าวมาแล้ว ในการออกแบบเครื่องวัดคลื่นไฟฟ้าหัวใจยังจะต้องคำนึงถึงความปลอดภัยในการวัดด้วย เนื่องจากในการวัดต้องใช้สายขั้วไฟฟ้าต่ออยู่ระหว่างผู้ป่วยและเครื่องวัด ดังนั้นถ้าหากเครื่องวัดมีกระแสไฟฟ้ารั่วไหลมาก็อาจทำให้ผู้ป่วยได้รับอันตราย หรืออาจจะทำให้ผลการวัดออกมาผิดพลาดได้ ดังนั้นเครื่องวัดคลื่นไฟฟ้าหัวใจ จึงมีคุณสมบัติพิเศษแตกต่างจากวงจรขยายทั่วๆไป คือ วงจรขยายในภาคแรกจะต้องเป็นวงจรขยายแบบ Floating Amplifier คือเป็นวงจรที่มีสาย common ไม่ได้ต่อที่แท่นเครื่องโดยตรง จึงทำให้ผู้ป่วยและแท่นเครื่องแยกจากกัน นอกจากนี้ยังมีส่วนของวงจร Opto Couple ที่สามารถป้องกันอันตรายที่อาจจะเกิดกับผู้ป่วย

8.2 ระบบรับสัญญาณไฟฟ้าหัวใจจากร่างกาย

เครื่องวัดคลื่นไฟฟ้าหัวใจประกอบไปด้วยระบบวงจรที่สำคัญดังต่อไปนี้

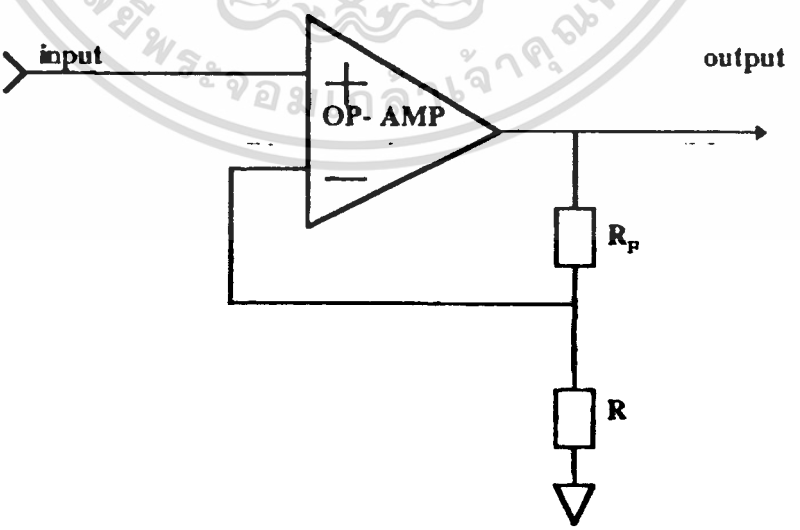


รูปที่ 3.1 Block Diagram ของเครื่องวัดคลื่นไฟฟ้าหัวใจ

ระบบวงจรในส่วนต่างๆของเครื่องวัดคลื่นไฟฟ้าหัวใจ ประกอบด้วยส่วนต่างๆดังต่อไปนี้

3.2.1. Buffer Amplifier

คุณสมบัติที่สำคัญของวงจร Buffer Amplifier คือมีค่า Input Impedance สูง เพื่อให้สัญญาณที่วัดได้มีการสูญเสียที่รอยสัมผัสของผิวหนังกับขั้วไฟฟ้าน้อยที่สุด ดังนั้นวงจรที่เหมาะสมที่จะนำมาใช้เป็นวงจร Buffer Amplifier คือ วงจร Non-Inverting Amplifier ดังแสดงในรูปที่ 3.2



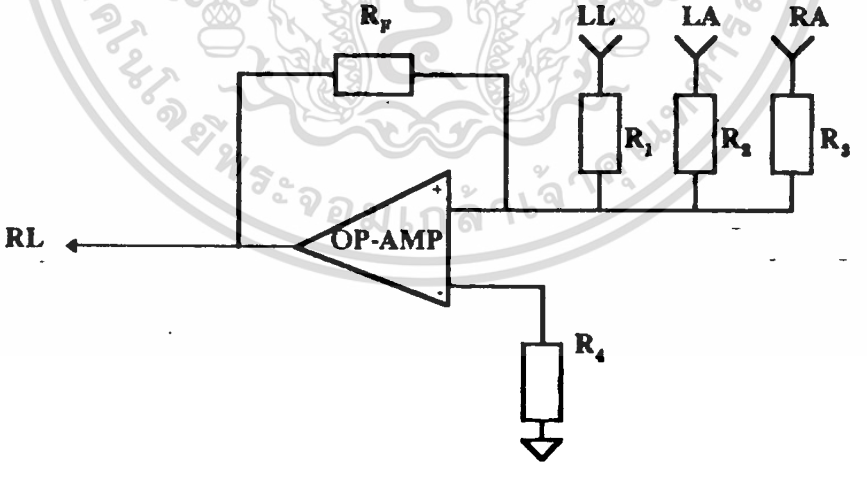
รูปที่ 3.2 Buffer Amplifier

นอกจากนี้วงจร Buffer Amplifier ยังทำหน้าที่ขยายสัญญาณไฟฟ้าให้มีค่าสูงขึ้นระดับหนึ่งก่อนที่จะผ่านไปยังวงจร Instrumention Amplifier สิ่งที่สำคัญนอกจากนี้คือ วงจรขยายในภาคแรกไม่ควรให้มีอัตราขยายมากเกินไป เพราะว่าถ้าเกิดมีศักดาไฟฟ้าออฟเซ็ทเกิดขึ้นที่อินพุทไม่ว่าจะสาเหตุใดก็ตาม จะทำให้สัญญาณออกที่เอาท์พุทลอยขึ้นไปหรือต่ำลงจากระดับศูนย์มาก ซึ่งบางครั้งอาจจะอิ่มตัวอยู่ที่ค่าใกล้กับศักดาไฟฟ้าของแหล่งจ่ายไฟ ทำให้วงจรไม่สามารถทำงานได้

ภาค Buffer Amplifier นี้ จะประกอบด้วย Non-Inverting Amplifier 4 ชุด เพื่อขยายสัญญาณที่วัดจากขั้วไฟฟ้าในแต่ละเส้น คือ RA, LA, LL, และ C

3.2.2. RL Driver

เนื่องจากในการวัดคลื่นไฟฟ้าหัวใจนั้น ได้กำหนดให้ขาขวาเป็น Reference และมีสายขั้วไฟฟ้าไปต่อที่จุด common ของวงจรขยาย แต่เนื่องจากจุด common ของวงจรต่อโดยตรงกับ Power Supply จึงอาจจะทำให้เกิดอันตรายแก่ผู้ป่วยได้ ถ้ามีกระแสรั่วไหล ดังนั้นจึงจำเป็นต้องมีวงจรที่ทำหน้าที่เป็น Reference แทนจุด common เพื่อแยกผู้ป่วยออกจาก Power Supply โดยตรง วงจรดังกล่าวนี้มีลักษณะเป็นวงจร Summing ดังในรูปที่ 3.3



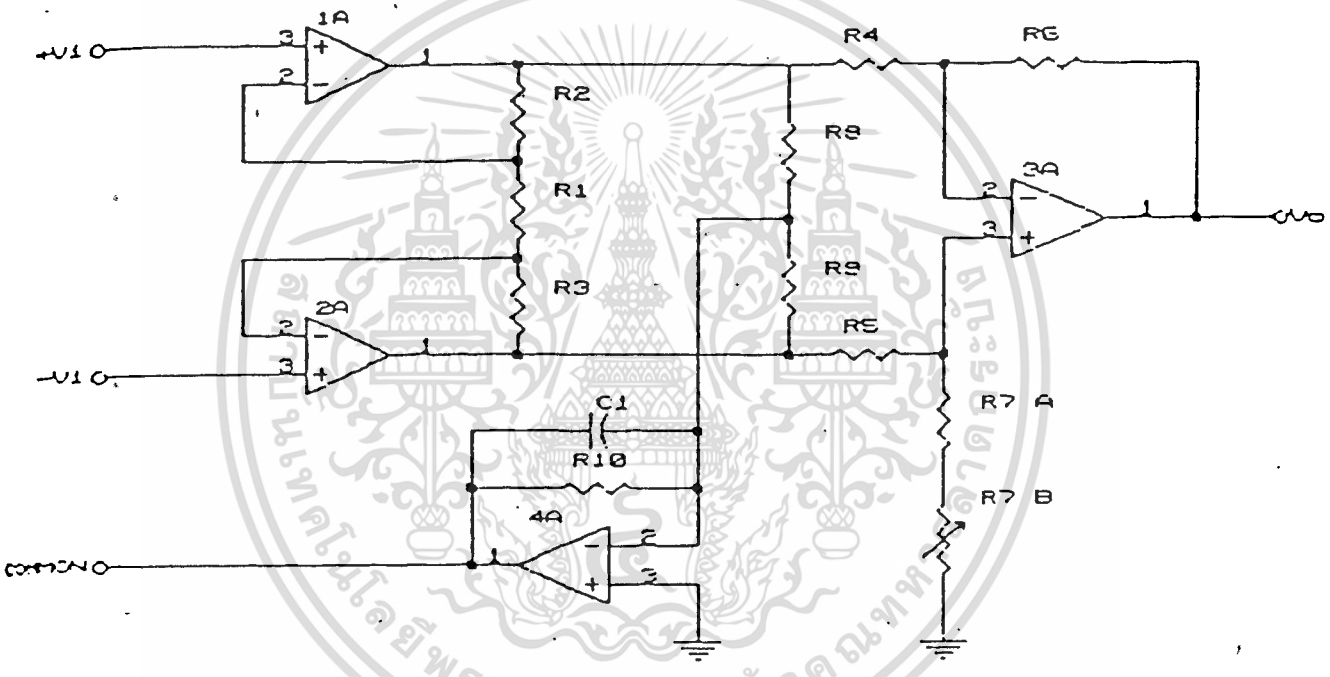
รูปที่ 3.3 RL Driver

Input ของวงจร Summing นี้ จะต่อจาก Output ของ Buffer amplifier 3 ชุด คือ RA, LA, และ LL เนื่องจากคลื่นไฟฟ้าหัวใจมีคุณสมบัติอย่างหนึ่งคือ ศักดาไฟฟ้าที่วัดได้จากแขนขวา แขนซ้าย และขาซ้าย เมื่อรวมกันแล้วจะมีค่าเท่ากับศูนย์ ดังนั้น Output ของวงจร

Summing นี้จึงมีค่าศูนย์อยู่ตลอดเวลาและทำหน้าที่เป็น Reference ได้

3.2.3. วงจรขยายสัญญาณแบบอินสตรูเมนชั่น (Instrumention Amplifier)

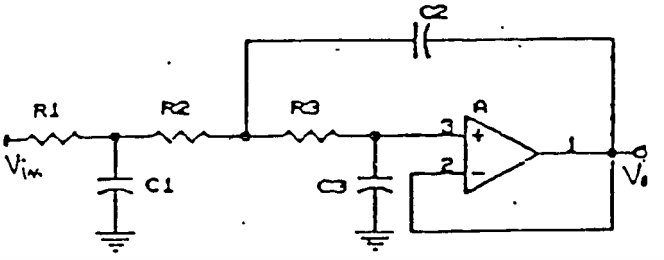
Instrumention Amplifier เป็นวงจรที่ขยายคลื่นไฟฟ้าหัวใจที่มีอัตราการขยายของสัญญาณมากกว่า Buffer Amplifier และมี Input Impedance สูง ส่วนสัญญาณที่ถูกขยายมานั้นจะมีสัญญาณรบกวนจากไฟฟ้าบ้าน กระแสสลับความถี่ 50 Hz ปนมาด้วย ดังนั้นวงจรที่เราจะใช้ควรจะเป็นแบบ Differential Amplifier เพื่อลดสัญญาณรบกวนให้ได้มากที่สุด เพราะว่าคุณสมบัติของ Differential Amplifier คือมีค่า CMRR สูง



รูปที่ 3.4 วงจรขยายความแตกต่างแบบ Instumention

3.2.4. วงจรกรองความถี่ต่ำ (Low Pass Filter)

วงจรนี้ใช้สำหรับแยกเอาเฉพาะคลื่นไฟฟ้าหัวใจ ซึ่งมีความถี่ระหว่าง 0.5-200 Hz กรองความถี่ที่ต่ำกว่า 200 Hz ผ่านเท่านั้น ดังนั้นสัญญาณรบกวนที่มีความถี่มากกว่านี้ ก็ไม่สามารถผ่านไปได้

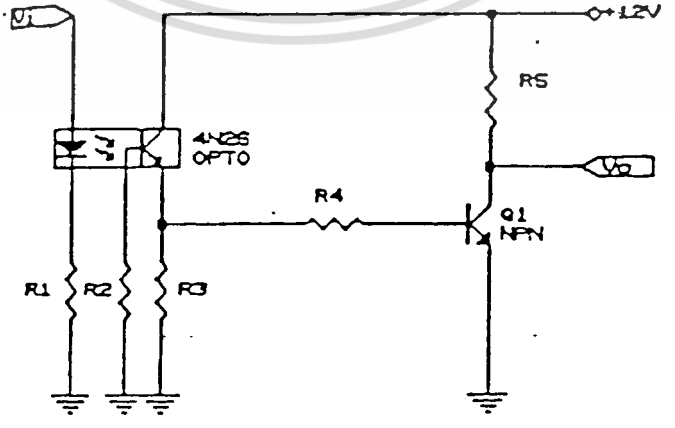


รูปที่ 3.5 Low Pass Filter

3.2.5. วงจรออปโต้ คัปเปิ้ล (Opto Couple)

ดังได้กล่าวมาแล้วว่าในการออกแบบเครื่องวัดคลื่นไฟฟ้าหัวใจจะต้องคำนึงถึงความปลอดภัยในการใช้ด้วย คือหลังจากที่กรองสัญญาณที่ไม่ต้องการไม่ให้ผ่านได้แล้ว ก็ผ่านมาที่วงจรส่งผ่านสัญญาณด้วยแสง เพื่อทำการแยกจุดดินระหว่างวงจรขยายในส่วนหน้าที่สัมผัสกับร่างกายกับวงจรถัดไป เพื่อป้องกันกระแสรั่วไหลจากเครื่องเข้าไปทำอันตรายต่อคนไข้ได้ หรืออาจจะทำให้ผลการวัดผิดพลาดได้

การทำงานของวงจรนี้ คือ เมื่อมีสัญญาณอินพุทเป็นระดับสูง จะทำให้ LED สว่าง และทำให้ Opto Transistor ในตัว Opto couple ทำงานส่งสัญญาณต่อไป และทรานซิสเตอร์ Q1 จะทำหน้าที่ขยายสัญญาณที่ได้จากตัว Opto couple เพื่อให้มีขนาดเพียงพอที่จะนำไปใช้งานในภาคต่อไป

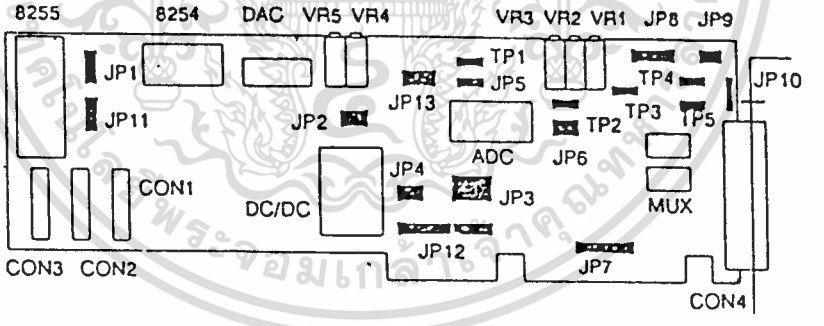


รูปที่ 3.6 Opto Couple

บทที่ 4 การบันทึกผล

การที่จะบันทึกผลสัญญาณ ECG ที่ได้จากการวัด ซึ่งเป็นสัญญาณ analog เราจะต้องนำสัญญาณนี้ไปเปลี่ยนเป็นสัญญาณ digital เสียก่อน จากนั้นจึงนำสัญญาณนี้ไปประมวลผลใน computer จึงสามารถเก็บข้อมูลของสัญญาณ ECG นี้ได้ในรูปของ file

สำหรับการแปลงสัญญาณจาก analog ไปเป็นสัญญาณ digital นั้นเราจะใช้วงจรที่เรียกว่า analog to digital converter ทำการเปลี่ยนสัญญาณ จากนั้นเราจะต้องนำสัญญาณนี้ใส่ computer ดังนั้นเราจะต้องทำการ Interface โดยเราเลือกใช้การ์ด PA-MA 12(-H) ของบริษัท Acqutek ซึ่งมีความสามารถในการแปลงสัญญาณ analog ไปเป็นสัญญาณ digital ได้ด้วยค่า Sampling Rate ที่สูงมาก (สูงสุด 100 Khz) และยังสามารถเก็บค่าความละเอียดของสัญญาณ ในรูปของ digital ได้ถึง 12 bit มากกว่าการ์ดทั่วไป (8 bit) และยังสามารถใช้การ Interface ข้อมูลได้หลาย ๆ วิธี และยังสามารถใช้การ transfer ข้อมูล แบบ DMA (Direct Memory Access) ทั้งยังเขียนโปรแกรมควบคุมการ์ดได้โดยง่าย

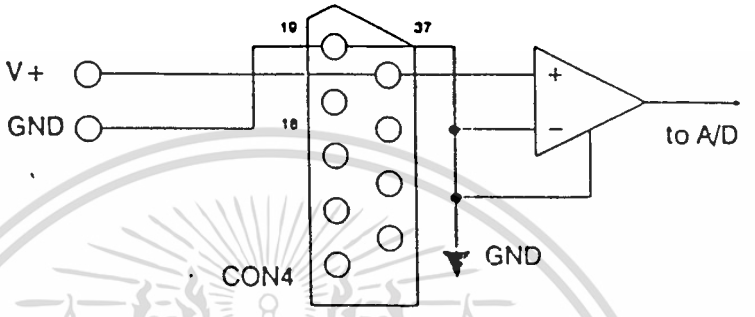


รูปที่ 4.1 แสดงตำแหน่งของ Jumper ภายในการ์ด PA-MA12(-H)

ดังนั้นสิ่งที่จะต้องทำอันดับแรกคือ เราจะต้องทำการ Interface การ์ด PA-MA 12(-H) เข้ากับเครื่อง computer เสียก่อน (ในโครงงานนี้ เราใช้ computer PC แบบ AT) โดยการ Installation การ์ดนี้รวมถึงการ transfer ข้อมูลนั้นสามารถแบ่งได้เป็นหลักใหญ่ ๆ ได้คือ Hardware และ Solfware

ก่อนที่จะติดตั้งหรือควบคุมการ์ด PA-MA12(-H) เราจะต้องทราบถึงลักษณะของสัญญาณที่

เราจะนำไปเข้าการ์ดเสียก่อน เพื่อที่จะสามารถ set configuration ของการ์ดได้อย่างถูกต้อง ในโครงการนี้หลังจากขยายสัญญาณ ECG ที่วัดนี้เรียบร้อยแล้ว เราจะให้สัญญาณนี้เป็นแบบ single-ended Input สำหรับการ์ด PA-MA(-H) รูปแบบในการนำเข้าการ์ด



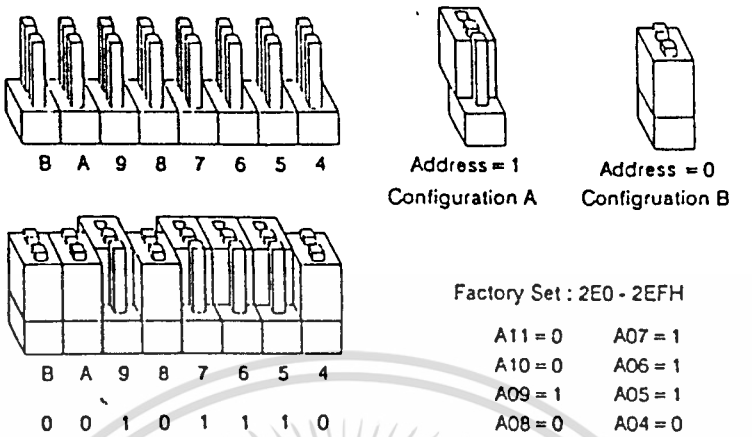
รูปที่ 4.2 แสดงการนำสัญญาณแบบ single ended เข้าสู่การ์ด

เนื่องจากสัญญาณ ECG มีลักษณะที่สามารถมีค่าเป็นลบได้ เราจึงต้องให้ Input แบบ unipolar ดังนั้นการติดตั้งการ์ด จะต้องทำดังนี้

4.1 Hardware Installation

4.1.1. เลือก I/O port address จากการ์ด เราต้องเลือก I/O address ที่ยังว่างอยู่ใน PC มาใช้เป็น address การ์ด ซึ่งเราเลือกใช้ address 2E0-2EF

การ set การ์ดนี้เราใช้ address ตามที่บริษัทเป็นผู้ set มาให้ ถ้าต้องการเลือก address ด้วยตัวเองมีวิธีดังรูป(X10H-XFFH)



รูปที่ 4.3 แสดงการ set address ที่ Jumper JP7

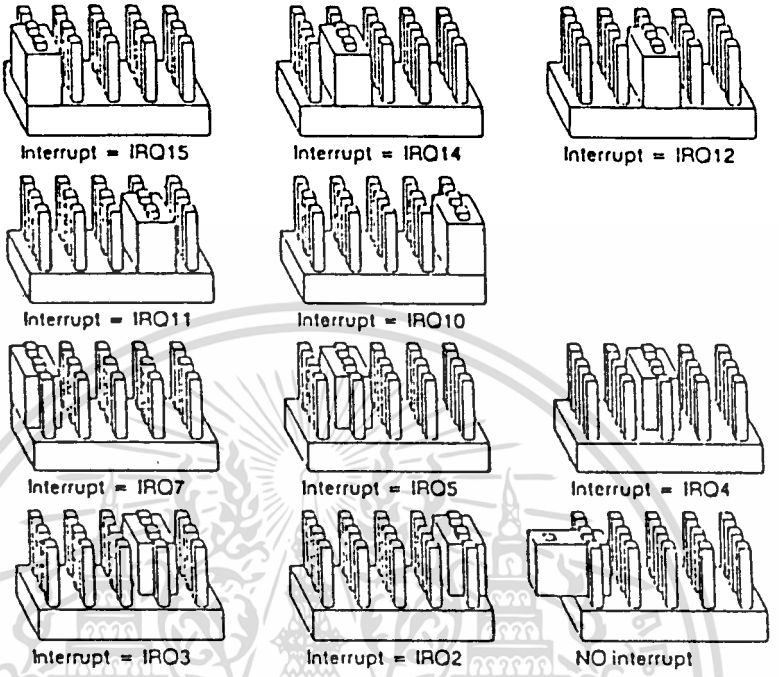
4.1.2.การเลือก interrupt channel เพื่อกำหนดช่องสัญญาณ Interrupt request ซึ่ง PC จะมีช่อง IRQ ให้เลือกใช้ 15 ช่อง แต่ละช่องใช้งานดังนี้

IRQ	Address	Description	Detected	Handled By
0	0C3C : 1875	Timer Clock	Yes	Block Device
1	0C3C : 1923	Keyboard	Yes	Block Device
2	0433 : 0057	Second 8259A	Yes	Default Handless
3	0433 : 006F	COM2 : COM4 :	COM2 :	Default Handless
4	0433 : 0087	COM1 : COM3 :	COM1 :	Default Handless
5	0433 : 009F	LPT2 :	No	Default Handless
6	0433 : 00B7	Floppy Disk	Yes	Default Handless
7	0070 : 06F4	LPT1 :	Yes	System Area
8	0433 : 0052	Real - Time Clock	Yes	Default Handless
9	F000 : EC3D	Redirected IRQ2	Yes	Bios
10	0433 : 00CF	(Reserved)		Default Handless
11	0433 : 00E7	(Reserved)		Default Handless
12	0433 : 00FF	(Reserved)		Default Handless
13	F000 : EC46	Math Coprocessor	Yes	Bios
14	0433 : 0117	Fixed Disk	Yes	Default Handless
15	0433 : 012F	(Reserved)		Default Handless

*** ที่มาตาราง IRQ Status จากโปรมแกรมใน DOS (MSD.EXE)

ตารางที่ 4.1 แสดง address ของ IRQ และบอกการใช้งานของ แต่ละ IRQ

เราเลือก Interupt ที่ยังว่างอยู่ในที่นี้เลือก IRQ 10 โดยมีวิธี set ค่าดังรูป



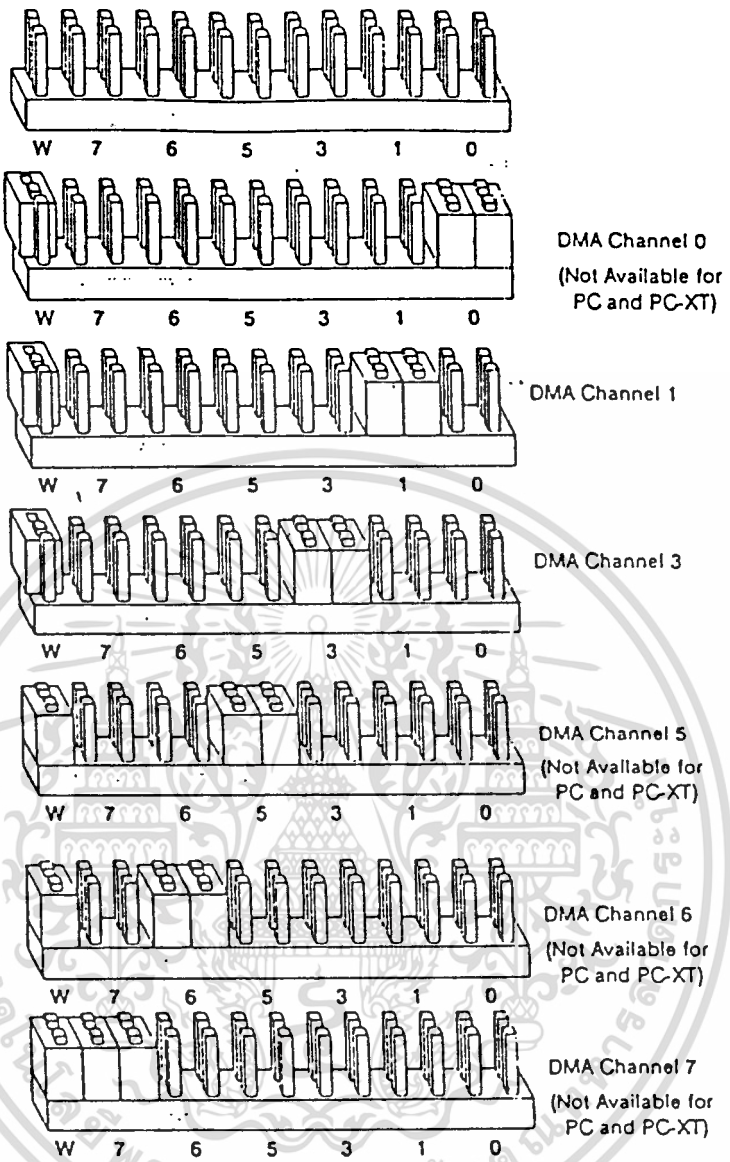
รูปที่ 4.4 แสดงการ set Interupt request โดย set ค่า ที่ JP3.

4.1.3. การเลือก DMA channel เนื่องจากการ transfer ข้อมูลจะสามารถส่งข้อมูลได้ 2 แบบ คือ แบบ byte และ แบบ word

-แบบ byte จะสามารถ transfer ข้อมูลได้ทีละ 8 bit

-แบบ word จะสามารถ transfer ข้อมูลได้ทีละ 16 bit

ดังนั้น การ transfer ข้อมูลนี้ ข้อมูลมีขนาด 12 bit ถ้า transfer แบบ byte เราจะต้อง transfer ข้อมูล 2 ครั้ง ใน 1 ค่าข้อมูล ถ้า transfer แบบ word เราจะใช้การ transfer ข้อมูลเพียง 1 ครั้ง ใน 1 ค่าข้อมูล แต่การ transfer ข้อมูลแบบ word นี้ เราจะใช้ได้ กับ computer แบบ AT เท่านั้น และต้องใช้กับ slot 16 bit เท่านั้น แต่โครงงานนี้ ใช้ computer แบบ AT อยู่แล้วเราจึงเลือกใช้การ transfer ข้อมูลแบบ word โดยการ เลือก DMA channel จะเลือกได้เพียง channel 5 ,6 ,7 เท่านั้นซึ่งมีวิธีเลือกดังรูป

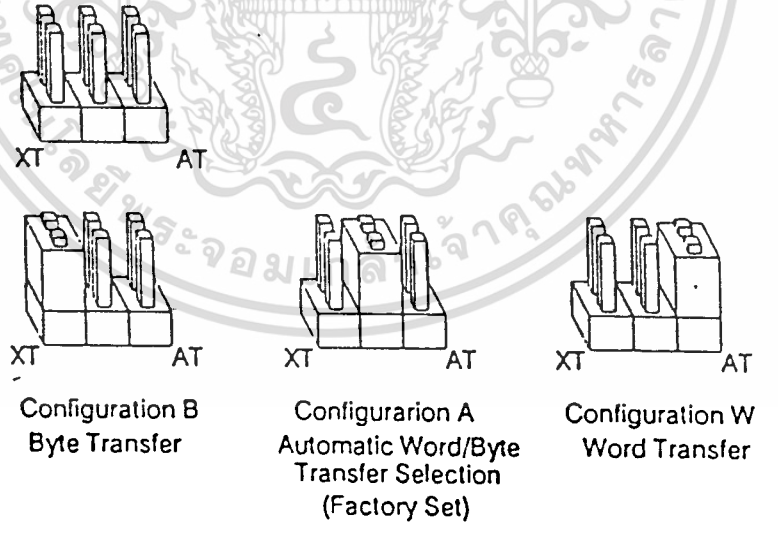


รูปที่ 4.5 แสดงการ set DMA channel โดย set JP12

4.1.4. หลังจากเลือก DMA channel แบบ word แล้วเราจะต้อง set ค่า data length ให้เป็นแบบ word ด้วย โดยมีวิธี set ดังรูป

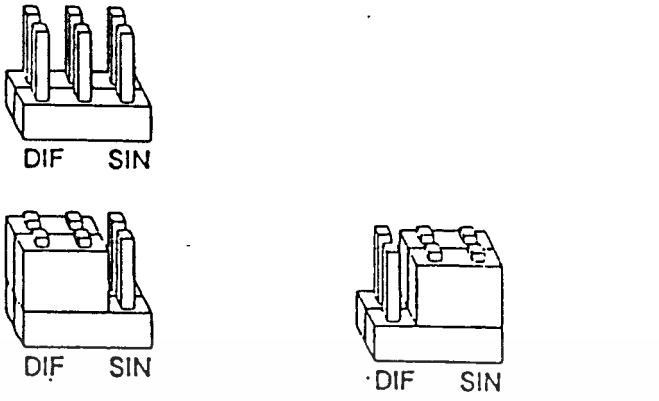
JP 4	Computer	Slot	Data length
Configuration B	8088 (PC/XT)	Any slot	Byte
	80286 (PC-AT)	Any slot	Byte
	80386 or 80486 PC	Any slot	Byte
Configuration W	8088 (PC/XT)	Any slot	N/A
	80286 (PC-AT)	8 bit slot	N/A
		16 bit slot	Word
	80386 or 80486 PC	8 bit slot	N/A
		16 bit slot	Byte
Configuration A	8088 (PC/XT)	Any slot	Byte
	80286 (PC-AT)	8 bit slot	Byte
		16 bit slot	Word
	80386 or 80486 PC	8 bit slot	Byte
		16 bit slot	Word

ตารางที่ 4.2 แสดง configuration ของการติดตั้งการ์ด



รูปที่ 4.6 แสดงการ set รูปแบบ data โดย set JP4

4.1.5. การเลือก Input type เราเลือกให้เป็น single ended มีวิธี set ดังรูป

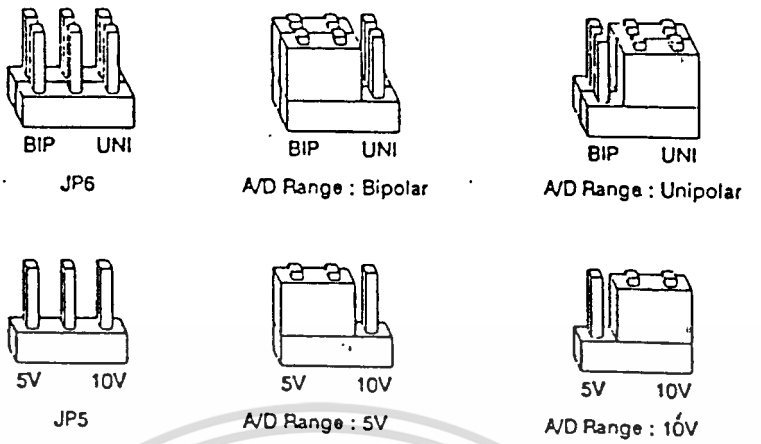


รูปที่ 4.6 แสดงการ set Input type โดยใช้การ set JP 9

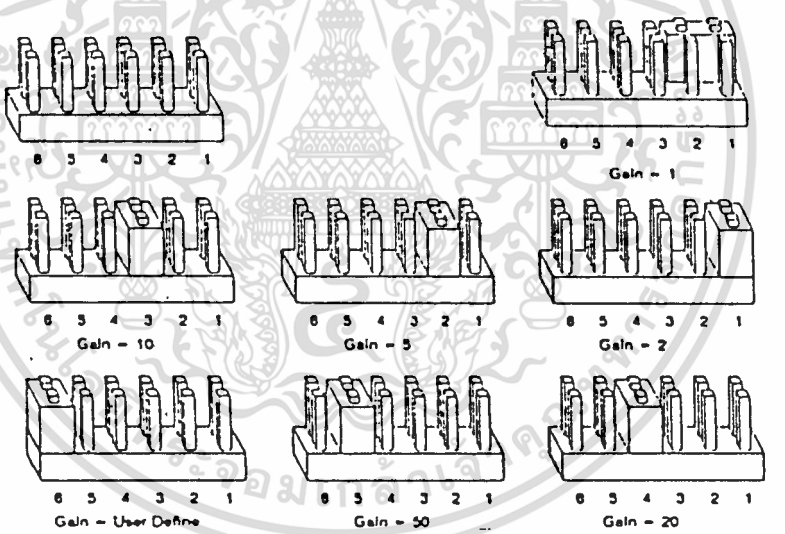
4.1.6 เลือก Input range เลือกค่า Voltage สูงสุด และ ต่ำสุดที่จะเข้าการ์ดว่ามีค่าเท่าไร และให้มีค่าเป็นแบบ bipolar คือ มีทั้ง + และ - โดยให้มี Voltage range -5, +5 ขวิธีการ set มีแสดงดังตาราง

Jumper Setting				Analog to Digital Conversion
JP 5	JP 6	JP 8		
		Jumper pin	Gain	Voltage Range
10 V	Unipolar	don't care		N/A
	Bipolar	Not Installed	1	-10 V to +10 V
5V	Unipolar	Position 6	User	User define (R14)
		Position 5	50	0 V to +0.25 V
		Position 4	20	0 V to +0.5 V
		Position 3	10	0 V to +1 V
		Position 2	5	0 V to +2 V
		Position 1	2	0 V to +5 V
		Not Installed	1	0 V to +10 V
	Bipolar	Position 6	User	User define(R14)
		Position 5	50	-0.1 V to +0.1 V
		Position 4	20	-0.25 V to +0.25 V
		Position 3	10	- 0.5 V to +0.5 V
		Position 2	5	-1 V to +1 V
		Position 1	2	-2.5 V to +2.5 V
		Not Installed	1	-5 V to +5 V

ตารางที่ 4.3 แสดงค่าการ set JP ต่าง ๆ ในตาราง

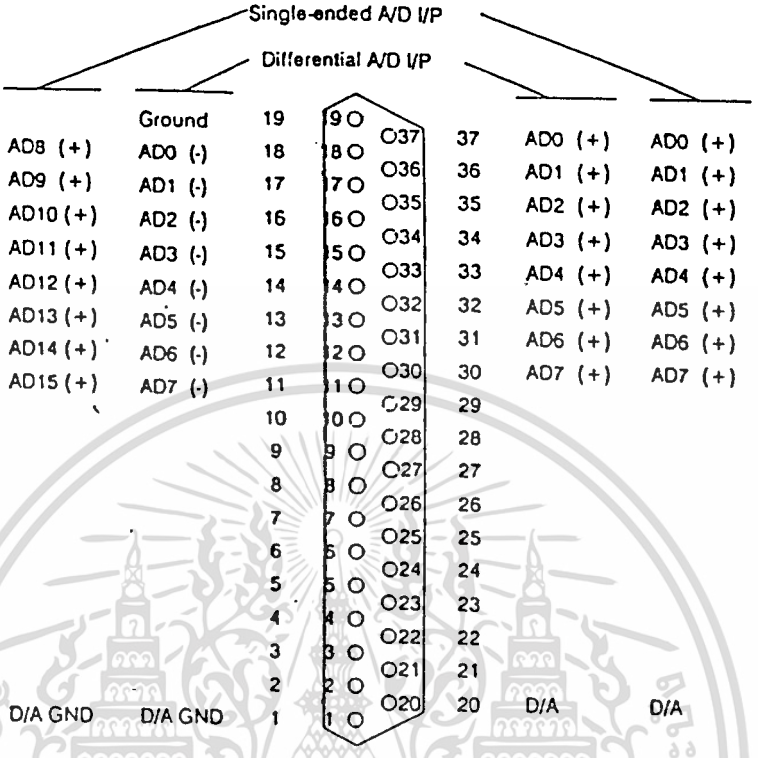


รูปที่ 4.7 การ set Voltage Range โดย set JP5 และ JP 6



รูปที่ 4.8 แสดงการ set Input gain โดย set JP 8

4.1.7. ต้องทำการต่อสายสัญญาณเข้ากับ connecter ของการ์ด เพื่อนำสัญญาณเข้าสู่การ์ดได้ถูกต้องตามที่เขียน Solfware ควบคุมสัญญาณ



รูปที่ 4.10 แสดงตำแหน่ง pin ของ connecter ของ การ์ด PA-MA12(-H)

4.2 Solfware Programming

หลังจากที่เราทำการติดตั้งการ์ดเรียบร้อยแล้ว เราจะต้องทำการเขียนโปรแกรมเพื่อที่จะทำการควบคุมการ transfer ข้อมูลและต้องเขียนโปรแกรมในการ Interface การ์ดกับ computer อีกด้วย โดยการ transfer ข้อมูลนั้นมีอยู่ 3 วิธีใหญ่ ๆ คือ

ก. CPU polling เป็นวิธีในการ transfer ข้อมูลที่ง่ายที่สุดใน 3 วิธี โดยการ transfer ข้อมูลวิธีนี้ใช้หลักการคือ ในการ ทำงาน 1 conversion เมื่อจบการทำงานแล้วจะทำให้สัญญาณ EOC (End Of Conversion) เปลี่ยนจาก 1 ไปเป็น 0 CPU จะคอยตรวจสอบสัญญาณ EOC ว่าเมื่อ ใดที่สัญญาณ EOC เปลี่ยนสัญญาณไป จาก 1 ไปเป็น 0 CPU ก็จะอ่านข้อมูลจาก register ในการ์ด แล้วไปเก็บไว้ใน memory ด้วยตัวเอง แต่วิธีนี้ทำให้ CPU ไม่สามารถไปทำงานอื่น ๆ ได้ ทำให้ การทำงานสูญเสียประสิทธิภาพไป

ข. Interrupt วิธีนี้สามารถเพิ่มประสิทธิภาพของการทำงานของ CPU ได้โดยการทำงานของวิธีนี้มีหลักการดังนี้

ในการทำการ conversion CPU สามารถไปทำงานอื่น ๆ ได้ โดยการ conversion นี้จะทำ

การเก็บข้อมูลของการ conversion ไว้ใน Interrupt Data Buffer ที่เรา set ไว้ หลังจากจบการ conversion แล้วจะเกิดสัญญาณ Interrupt ไปที่ CPU เพื่อบอกให้ CPU ทราบว่าการ conversion ได้เสร็จสมบูรณ์แล้วจากนั้น CPU ก็จะทำการอ่านข้อมูลที่เก็บอยู่ใน Interrupt Data Buffer ไปเก็บใน memory จากนั้นสัญญาณ Interrupt ก็จะ disable แสดงว่าจบการ Interrupt แล้ว CPU ก็จะไปทำงานอย่างอื่น จนกว่าจะมีสัญญาณ Interrupt ใหม่อีกครั้งก็จะทำการอ่านข้อมูลที่เก็บอยู่ใน Interrupt Data Buffer อีกครั้งเป็นอย่างนี้เรื่อยไป

ค. DMA (Direct Memory Access) วิธีนี้เป็นวิธีที่มีประสิทธิภาพมากที่สุดเพราะการ transfer แบบนี้ มีความรวดเร็วสูงโดยใช้การติดต่อระหว่างการ์ดกับ memory โดยตรงไม่ผ่าน CPU ทำให้ CPU สามารถไปทำงานอื่น ๆ ที่มีความสำคัญกว่าได้แต่ต้องใช้การเขียนโปรมแกรมที่ยุ่งยาก มีหลักการค่อนข้างจะยุ่งยากและซับซ้อนโดยการทำงานจะใช้ ตัว DMA controller เป็นตัวควบคุมการทำงานของการ DMA คือการ DMA จะเกิดขึ้นได้เราจะต้องทำให้ DMA controller enable เสียก่อน จะต้องทำการเลือกตำแหน่งของ memory และ ข้อมูลจะถูกเก็บอยู่ใน word register โดยจะมี register อีกตัวทำหน้าที่นับการ transfer จนเมื่อจบการ transfer count register จะมีค่าเป็น 0 และจะเกิดสัญญาณ Interrupt เพื่อบอกว่า การ transfer สมบูรณ์ และหลังจากการ transfer สมบูรณ์ แล้วจะเกิดการ disable DMA controller ขึ้นอีกครั้ง

จะเห็นได้ว่าการ transfer ข้อมูลแบบที่ 3 ดีที่สุดเราจึงเลือกใช้การ DMA ในการ transfer ข้อมูลเพื่อความรวดเร็วซึ่งการเขียนโปรแกรมและการ ทำงานของ DMA มีดังนี้

ในการส่งผ่านข้อมูลแบบ DMA นั้นเราสามารถส่งผ่านข้อมูลได้ 2 แบบคือแบบ word และแบบ byte ในการส่งผ่านข้อมูลทั้ง 12 bit

-การ transfer แบบ byte จะใช้การส่งผ่านข้อมูล 2 ครั้งในการ transfer ข้อมูลหนึ่งข้อมูลและเนื่องจากการ transfer จะส่งข้อมูล 16 bit นี้ จะเป็นข้อมูล 12 บิท และอีก 4 บิทจะเป็น channel ของสัญญาณที่เข้ามาในการ์ด การ transfer แบบ นี้สามารถทำการ transfer ข้อมูลได้มากที่สุด 64K (65,536) bytes หรือ 32K ครั้งของการ transfer

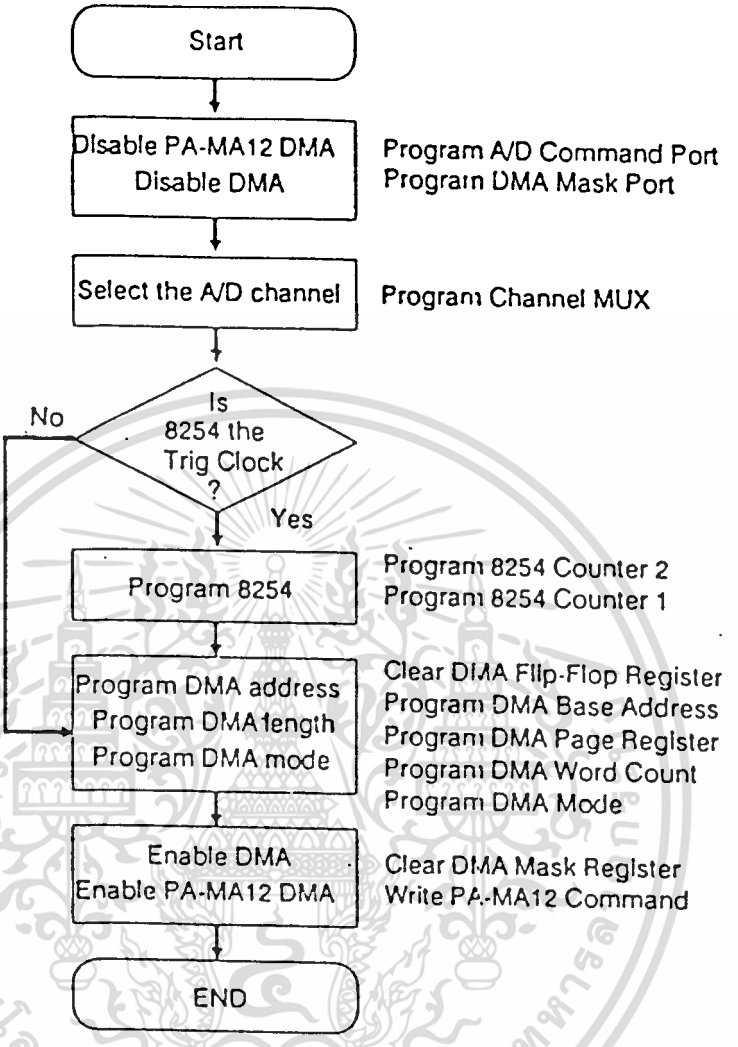
-การ transfer แบบ word จะใช้การส่งผ่านข้อมูลเพียง 1 ครั้งในการ transfer ข้อมูลหนึ่งข้อมูล และเนื่องจากการ transfer แบบ word จะส่งข้อมูล 16 bit นี้ จะเป็นข้อมูล 12 บิท และอีก 4 บิทจะเป็น channel ของสัญญาณที่เข้ามาในการ์ด การ transfer แบบ นี้สามารถทำการ transfer ข้อมูลได้มากที่สุด 64K (65,536) words หรือ 64K ครั้งของการ transfer ข้อมูล

ข้อแตกต่างระหว่างการ trasfer ทั้ง 2 แบบนี้ สามารถแสดงได้ ดังตารางที่ 4.4

Major Difference	Word Transfer	Byte transfer
Data lenght per transfer	16 bit (one word)	8 bit (one byte)
DMA Channel	5 , 6 , 7	0 , 1 , 3
Number of DMA transfer needed to be made per conversion	1 Transfer 4 bits channel number and all 12 bits data simultaneous	2. Transfer 4 bits Channel number and 4 A/D LSB data first then 8 A/D MSB data
Estimate total time need for one A/D conversion	less than 1 us	more than 2 us
Maximun A/D conversion	64K (65,536) times	32K (32,768) times
Memory	A/D data can be trasferred to any of the all 16MB	A/D data can be transferred to any of all 16 MB
H/W Installation	Must install the PA-MA(-H) in a 16 bit soit of PC-AT computer	Can be Installed in any slot of any PC compatible computer
Advantage	Fast Converaion anf saves time for CPU to do other important tasks	Can be installed in any slot of any PC compatible computer
Disadvantage	Can be installed only in a 16 bit slot PC-AT compatible computer	Takes more time to transfer A/D data to memory,may result in slowing down the CPU foreground task's proformance
Note	There is no effect on to the conversion	speed; even at 100 Khz A/D conversion

ตารางที่ 4.4 แสดงความแตกต่างระหว่าง word กับ byte transfer

เราสามารถแสดงการทำงานของ DMA transfer ได้ดัง Flow chart ต่อไปนี้



รูปที่ 4.11 Flow chart แสดงขั้นตอนการทำงานของ DMA transfer

เราจะต้องเขียนโปรแกรมตาม Programming Procedures ดังนี้

4.2.1 : Reset the PA-MA12 and disable the DMA

ขั้นตอนแรกที่เราจะต้องทำคือในการเขียนโปรแกรมเพื่อ disable การ์ด PA-MA12 และ DMA controller ก่อนที่จะทำการ transfer ข้อมูล เพื่อ ป้องกัน error ที่อาจจะเกิดขึ้น โดยดูค่าต่าง ๆ ดังตารางข้างล่าง

(1) reset การ์ด PA-MA12

ทำได้โดยการ Disable 8254/external clock trig และ DMA request โดยการ เขียน 00H ไปที่ A/D command port (base +3)

Write Sequent	All channel (7 , 6 , 5 , 3 , 1 , 0)	
	Port	Data
1-1	P + 3	00H
Note : P PA-MA12(-H) base port address		

ตารางที่ 4.5 แสดงการเขียนค่าเพื่อ reset PA-MA12

(2) Disable the DMA controller

เราสามารถ disable DMA controller ได้ โดยเขียนค่าดังตาราง

Write	Channel 3		Channel 1		Channel 0	
Sequent	Port	Data	Port	Data	Port	Data
1-2	00AH	07H	00AH	05H	00AH	04H

Write	Channel 7		Channel 6		Channel 5	
Sequent	Port	Data	Port	Data	Port	Data
1-2	D4H	07H	D4H	06H	D4H	05H

ตารางที่ 4.6 แสดงการเขียนเพื่อ disable DMA controller

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer ที่ DMA channel 3คือ

```
/* Disable DMA transfer first  */
outp(PORT+3,0);
outp(0x0A,0x07);
```

4.2.2 : Program A/D Input multiplexer

เราจะต้องเลือกใช้ channel ในการ transfer ข้อมูลว่าเราจะนำสัญญาณเข้าในการ์ดที่ channel ใดบ้าง โดยเขียนโปรมแกรมไปที่ port +2 ตามลักษณะของสัญญาณ input ที่ได้ทำการเลือกไว้แล้ว

A/D Channel MUX - Single Ended Input							
D7	D6	D5	D4	D3	D2	D1	D0
H3	H2	H1	H0	L3	L2	L1	L0
A/D high channel scan limit				A/D low channel limit			

ตารางที่ 4.7 แสดงการเลือก channel ของ MUX

จากตารางจะเห็นได้ว่าเราจะต้องเลือก channel ของ MUX ซึ่งมีให้เลือกได้ทั้งหมด 15 ช่อง ตารางข้างล่างจะแสดงค่าที่จะต้องเขียนไปที่ port +2 ตามการ convert ของ MUX ที่เรจะต้องการ ใช้ว่าจะใช้การแปลี่ยนแปลงของ MUX กี่ channel และ channel ใดไปสู่ channel ใด กล่าวคือ ถ้าเรา ให้มีการแปลี่ยนแปลง channel ของ MUX เราจะต้องเขียน channel ที่สูงสุดของการเปลี่ยนแปลงไปที่ A/D high scan limit และ เขียน channel ที่ต่ำสุดไปที่ A/D low scan limit ซึ่งค่าต่าง ๆ จะแสดงได้ ดังตาราง

MUX	Ch(s)	MUX	Ch(s)		MUX	Ch(s)	MUX	Ch(s)
00H	0	10H	0 to 1		B0H	0 to 14	F0H	0 to 15
01H	1	11H	1		B1H	1 to 14	F1H	1 to 15
02H	2	12H	2		B2H	2 to 14	F2H	2 to 15
03H	3	13H	3		B3H	3 to 14	F3H	3 to 15
04H	4	14H	4		B4H	4 to 14	F4H	4 to 15
05H	5	15H	5	.....	B5H	5 to 14	F5H	5 to 15
06H	6	16H	6		B6H	6 to 14	F6H	6 to 15
07H	7	17H	7	20H	B7H	7 to 14	F7H	7 to 15
08H	8	18H	8	to	B8H	8 to 14	F8H	8 to 15
09H	9	19H	9	DFH	B9H	9 to 14	F9H	9 to 15
0AH	10	1AH	10		BAH	10 to 14	FAH	10 to 15
0BH	11	1BH	11		BBH	11 to 14	FBH	11 to 15
0CH	12	1CH	12		BCH	12 to 14	FCH	12 to 15
0DH	13	1DH	13		BDH	13 to 14	FDH	13 to 15
0EH	14	1EH	14		BEH	14	FEH	14 to 15
0FH	15	1FH	15		BFH	15	FFH	15

ตารางที่ 4.8 แสดงการเขียนข้อมูลไปที่ port +2 ตามการเปลี่ยนแปลง channel ของ MUX

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer ข้อมูลที่ channel 0

```
/* Define the A/D channel MUX */
outp(POTR+2,0x00);
```

4.2.3 : Program 8254 to decide the frequency

เราจะต้องทำการกำหนดค่า Sampling Rate ที่เราจะทำการ Sampling เพื่อที่จะเลือกค่าการ Sampling ที่ถูกต้องโดยเขียนโปรแกรมไปที่ 8254 Counter/timer โดยกำหนดค่าตามตาราง

Analog		Data For 8254 Initialization					
Digital (A/D)		8254 Counter 2			8254 Counter 1		
Conversion		Base	Base	Base	Base	Base	Base
Frequency		+15	+14	+14	+15	+14	+14
100KHz	10 us	B4H	4	0	74H	10	0
80KHz	12.5 us	B4H	4	0	74H	25	0
50KHz	20 us	B4H	4	0	74H	20	0
40KHz	25 us	B4H	4	0	74H	25	0
25KHz	40 us	B4H	4	0	74H	40	0
20KHz	50 us	B4H	4	0	74H	50	0
10KHz	100 us	B4H	4	0	74H	100	0
5KHz	200 us	B4H	4	0	74H	200	0
2KHz	500 us	B4H	4	0	74H	244	1
1KHz	1 ms	B4H	4	0	74H	232	3
500Hz	2 ms	B4H	4	0	74H	208	7
200Hz	5 ms	B4H	4	0	74H	136	19
100Hz	10 ms	B4H	4	0	74H	13	39
50Hz	20 ms	B4H	4	0	74H	32	78
20Hz	50 ms	B4H	4	0	74H	80	195
10Hz	100 ms	B4H	40	0	74H	16	39
5Hz	200 ms	B4H	40	0	74H	32	78
2Hz	500 ms	B4H	40	0	74H	80	195
1Hz	1 s	B4H	144	1	74H	16	39
0.1Hz	10 s	B4H	160	15	74H	16	39
0.01Hz	100 s	B4H	64	156	74H	16	39

ตารางที่ 4.9 แสดงการเขียนโปรแกรมไปที่ port ที่กำหนดเพื่อกำหนดค่า Sampling Rate ที่ต้องการ

ตัวอย่างการเขียนโปรแกรม

1. เขียน B4H ไปที่ 8254 commond port (base+15)
2. เขียน 04H (4) to counter 2 data port (base+14)
3. เขียน 00H (0) to counter 2 data port (base+14)
4. เขียน 74H ไปที่ 8254 commond port (base+15)

5. เขียน E8H (232) to 8254 counter 1 data port (base+13)

6. เขียน 03H (3) to 8254 counter 1 data port (base+13)

จากการทำงานตามขั้นตอนบนจะสามารถโปรแกรม 8254 ที่มีความถี่ 1 KHz

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer ข้อมูลด้วยความถี่ Sampling 100 Hz

```
/*Initial 8254 Time*/
outp(PORT+15,0xB4);
outp(PORT+14,0x04);
outp(PORT+14,0x00);
outp(PORT+15,0x74);
outp(PORT+13,13);
outp(PORT+13,39);
```

4.2.4 : Program the transfer memory address

เป็นการเลือกตำแหน่ง memory ที่เราจะทำการ transfer ข้อมูล โดยจะต้องทำดังนี้

(1) Clear the DMA pointer filp-flop

เพื่อป้องกันความผิดพลาดที่อาจ ขึ้น เราต้อง clear filp-flop เสียก่อนที่จะทำการ transfer ข้อมูล

(2) Program DMA base address and page register

เป็นการเขียนค่า buffer address ไป ที่ current address register เพื่อบอกว่าจะ transfer ข้อมูลไปที่ใดใน memory

ทั้ง 2 ขั้นตอนนี้สามารถแสดงได้ในตาราง

แบบ byte

Write	Channel 8		Channel 1		Channel 0	
Sequent	Port	Data	Port	Data	Port	Data
4-1	0CH	00H	0CH	00H	0CH	00H
4-2	06H	A0-A7	02H	A0-A7	06H	A0-A7
4-3	06H	A8-A15	02H	A8-A15	06H	A8-A15
4-2	82H	A16-A23	83H	A16-A23	82H	A16-A23

แบบ word

Write	Channel 7		Channel 6		Channel 5	
Sequent	Port	Data	Port	Data	Port	Data
4-1	0CH	00H	0CH	00H	0CH	00H
4-2	06H	A0-A7	02H	A0-A7	06H	A0-A7
4-3	06H	A8-A15	02H	A8-A15	06H	A8-A15
4-2	82H	A16-A23	83H	A16-A23	82H	A16-A23

ตารางที่ 4.10 แสดงการเขียนค่าไป ที่ port ตาม channel ของการทำ DMA

ตัวอย่างการเขียนค่าตามตารางขั้นต้นนี้สามารถแสดงได้ตามตารางข้างล่าง

Example for DMA base address reg. and page progmaning									
System	Starting Address for DMA Transfer	Byte (8 bit) Transfer				Word (16 bit) Transfer			
		Data to be written			Max. Xfer Avail	Data to be written			Max. Xfer Avail
		Page reg.	Base address MSB	Base address LSB		Page reg.	Base address MSB	Base address LSB	
PC/XT and PC-AT	00000H	00H	00H	00H	64KB	00H	00H	00H	128KB
	10000H	01H	00H	00H	64KB	00H	80H	00H	64KB
	20000H	02H	00H	00H	64KB	02H	00H	00H	126KB
	30000H	03H	00H	00H	64KB	02H	80H	00H	64KB
	40000H	04H	00H	00H	64KB	04H	00H	00H	128KB
	40002H	04H	00H	02H	65,534	04H	00H	01H	131,070
	40010H	04H	00H	10H	65,520	04H	00H	08H	131,056
	40400H	04H	04H	00H	63KB	04H	02H	00H	127KB
	40800H	04H	08H	00H	62KB	04H	04H	00H	126KB
	41000H	04H	10H	00H	60KB	04H	08H	00H	124KB
	42000H	04H	20H	00H	56KB	04H	10H	00H	120KB
	44000H	04H	40H	00H	48KB	04H	20H	00H	112KB
	48000H	04H	80H	00H	32KB	04H	40H	00H	96KB
	4C000H	04H	C0H	00H	16KB	04H	60H	00H	80KB
	4FFFEH	04H	FFH	FEH	2	04H	7FH	FFH	65,538
	50000H	05H	00H	00H	64KB	04H	80H	00H	64KB
	58000H	05H	80H	00H	32KB	04H	C0H	00H	32KB
	5FFFEH	05H	FFH	FEH	2	04H	FFH	FFH	2
	60000H	06H	00H	00H	64KB	06H	00H	00H	128KB
	70000H	07H	00H	00H	64KB	06H	80H	00H	64KB
	80000H	08H	00H	00H	64KB	08H	00H	00H	128KB
	90000H	09H	00H	00H	64KB	08H	80H	00H	64KB
PC-AT only	100000H	10H	00H	00H	64KB	10H	00H	00H	128KB
	123456H	12H	34H	56H	52,138	12H	14H	2BH	117,674
	F12468H	F1H	24H	68H	56,216	F0H	12H	34H	56,216

ตารางที่ 4.11 แสดงตัวอย่างการเขียนค่าเพื่อกำหนด Address ของ memory ที่จะนำข้อมูลไปเก็บไว้

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer ข้อมูลไปที่ memory 80000H

```
outp(0x0C,0);       /* Clear DMA filp-flop */
outp(0x06,0x08);    /* DMA address A00-A07 or A01-A08 */
outp(0x06,0x00);    /*  DMA address A08-A15 or A09-A16 */
outp(0x82,0x00);    /* DMA address  A16-A19 or A17-A23 */
```

4.2.5 : Program the DMA current word counter

ตัว current word counter จะเป็นตัวเก็บค่าจำนวนการ transfer ข้อมูลที่เราจะทำการ transfer ไว้เพื่อที่จะใช้เป็นตัวนับการ transfer ข้อมูลที่เกิดขึ้น โดยเราจะเลือกขนาดของการ transfer ข้อมูลว่าจะให้ transfer ข้อมูลมากเท่าใดโดยเขียนค่าไปที่ current word counter โดยดูค่าที่จะทำการเขียนตามตาราง

แบบ byte

Write	Channel 3		Channel 1		Channel 0	
Sequent	Port	Data	Port	Data	Port	Data
5-1	07H	W0-W7	03H	W0-W7	01H	W0-W7
5-2	07H	W8-W15	03H	W8-W15	01H	W8-W15

แบบ word

Write	Channel 7		Channel 6		Channel 5	
Sequent	Port	Data	Port	Data	Port	Data
5-1	CEH	W0-W7	CAH	W0-W7	C6H	W0-W7
5-2	CEH	W8-W15	CAH	W8-W15	C6H	W8-W15

ตารางที่ 4.12 แสดงค่า address ของ DMA channel ที่จะต้องโปรแกรมค่า counter ของการ DMA

Example for DMA base count register programming								
Number of A/D Conver-sions	Byte (8 bit) DMA transfer				Word (16 bit) DMA transfer			
	Number of DMA transfer	Data to be written to DMA base count register			Number of DMA transfer	Data to be written to DMA base count register		
		Count	LSB	MSB		Count	LSB	MSB
1	2	1	01H	00H	1	0	00H	00H
2	4	3	03H	00H	2	1	01H	00H
3	6	5	05H	00H	3	2	02H	00H
4	8	7	07H	00H	4	3	03H	00H
5	10	9	09H	00H	5	4	04H	00H
:								
126	252	251	FBH	00H	126	125	7DH	00H
127	254	253	FDH	00H	127	126	7EH	00H
128	256	255	FFH	00H	128	127	7FH	00H
129	258	257	01H	01H	129	128	80H	00H
130	260	259	03H	01H	130	129	81H	00H
:								
32766	65532	65531	FBH	FFH	32766	32765	FDH	7FH
32767	65534	65533	FDH	FFH	32767	32766	FEH	7FH
32768	65536	65535	FFH	FFH	32768	32767	FFH	7FH
32769	N/A				32769	32768	00H	80H
32770	N/A				32770	32769	01H	80H
:								
65534	N/A				65534	65533	FDH	FFH
65535	N/A				65535	65534	FEH	FFH
65536	N/A				65535	65535	FFH	FFH
65537	N/A				N/A			
65538	N/A				N/A			
:	N/A				N/A			

ตารางที่ 4.13 แสดงค่าที่จะต้องเขียนสำหรับ DMA count register

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer ข้อมูลทีละ 5 ครั้งในแบบ byte transfer

```
outp(0x07,0x09);   /* DMA word W00-W07 */
outp(0x07,0x00);   /* DMA word W08-W15 */
```

4.2.6 : Initializes the DMA mode register

เป็นการ set ค่าเริ่มต้นให้ กับ DMA เป็นการกำหนดโหมดของการ DMA นั่นเองโดยกำหนดค่าตามตาราง

Write	Channel 3		Channel 1		Channel 0	
Sequent	Port	Data	Port	Data	Port	Data
6	09H	47H	09H	45H	09H	44H

Write	Channel 7		Channel 6		Channel 5	
Sequent	Port	Data	Port	Data	Port	Data
6	D2H	47H	D2H	46H	D2H	45H

ตารางที่ 4.14 แสดงการ set ค่า DMA mode

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการ

```
outp(0x0B,0x47);  /* Wriet DMA mode */
```

4.2.7 : Enable the DMA controller

หลังจากกำหนดค่าต่าง ๆ เรียบร้อยแล้ว หลังจากเรา enable DMA controller แล้ว DMA controller พร้อมที่จะทำการ transfer data ไปที่ memory แล้ว

Write	Channel 3		Channel 2		Channel 1	
Sequent	Port	Data	Port	Data	Port	Data
7	0AH	03H	0AH	02H	0AH	01H

Write	Channel 3		Channel 2		Channel 1	
Sequent	Port	Data	Port	Data	Port	Data
7	D4H	03H	D4H	02H	D4H	01H

ตารางที่ 4.15 การ Set ค่า Enable DMA controller

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer

```
outp(0x0A,0x03);  /* Enable DMA */
```

4.2.8 : Program the PA-MA12 commaned port

เป็นการเลือกว่าเราใช้ counter แบบไหน โดยการเลือกสามารถแสดงได้ดังตาราง โดยการเขียนค่าไปที่ commond port +3

Command						Trig Source	Interrupt
7-4	3	2	1	0	Hex		
x	1	0	1	0	0AH	Exteinal clock	Disabled
x	1	0	1	1	0BH		Enabled
x	1	1	0	0	0CH	8254 Timer	Disabled
x	1	1	0	1	0DH		Enabled
x	1	1	1	0	0EH	8254 Timer perced	Disabled
x	1	1	1	1	0FH	by external clock	Enabled

ตารางที่ 4.16 การเลือกค่าตามวิธีการใช้ตัว counter แบบ ต่าง ๆ

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการโดยทำการ transfer

```
/* Enable PA-MA12 DMA transfer and 8254 time trigger */
outp(PORT+3,0x0C);     /* Enable PA-MA12 DMA */
```

4.3 โครงสร้างของข้อมูล

เนื่องจากการ transfer ข้อมูลนี้ ให้ผลลัพธ์ เป็นข้อมูล 12 bit ดังนั้นการ transfer ข้อมูลจะต้อง transfer 16 bit โดยมีการกำหนด MSB และ LSB กล่าวคือข้อมูล 8 bit แรกจะเป็นข้อมูลเสีย 4 บิท อีก 4 บิทที่เหลือเป็น channel ของ MUX ที่ทำการ transfer ข้อมูลโดย ข้อมูลจะคือค่า ของบิทที่ 7-4 และ channel ของ MUX คือ บิทที่ 3-0 ข้อมูลนี้คือ บิทต่ำสุดของข้อมูล ส่วน 8 บิทหลังจะ เป็น บิทสูงของ ข้อมูล

ตัวอย่างเช่นถ้าเรา เลือก MUX จาก 0 ไปถึง 3 ข้อมูลที่ได้จากการ transfer ครั้งแรก จะเป็นข้อมูลของ channel ที่ 0 ของ MUX หลังจากนั้น ข้อมูลที่ได้ครั้งที่ 2 จะเป็นข้อมูลของ channel ที่ 1 ของ MUX และ ครั้งต่อไป เป็นของ channel ที่ 3 หลังจากนั้น ข้อมูลที่ ได้ ก็จะ เป็นของ channel ที่ 0 อีกครั้ง เป็น เช่นนี้เรื่อยไป

4.4 การตรวจสอบสถานะ

การทำ DMA transfer นั้นจะมี register ที่ชื่อว่า word count ทำหน้าที่ในการนับการ

transfer ข้อมูล ดังนั้นการ ตรวจสอบสถานะของการ transfer ถ้า word count เป็น 0 แสดงว่าการ transfer จบลงอย่างสมบูรณ์ โดยการเขียนโปรแกรมจะใช้การ check ที่ DMA staus register โดยการแสดงค่าของ staus register แสดงได้ดังตาราง

Write Sequent	Channel 3		Channel 2		Channel 1	
	Port	Data	Port	Data	Port	Data
C-read	08H	R = D7 C = D3	08H	R = D5 C = D1	08H	R = D4 C = D0

Write Sequent	Channel 7		Channel 6		Channel 5	
	Port	Data	Port	Data	Port	Data
C-read	D0H	R = D7 C = D3	D0H	R = D6 C = D2	D0H	R = D5 C = D1

ตารางที่ 4.17 แสดงการ Check สถานะของการ transfer

bit 7-4 เป็น 1 แสดงว่าเกิดการ requset

bit 3-0 เป็น 1 แสดงว่าเกิด สัญญาณ การ transfer Complete

หลังจากการ transfer เสร็จเรียบร้อยแล้วเราจะต้องทำการ disable การ์ด PA-MA12 และ DMA controller โดยการทำขั้นตอนนี้เหมือนกับขั้นตอนที่ 1 หลังจากนี้ต้องเขียนโปรแกรมในการนำข้อมูลที่ได้นี้ไปเก็บไว้ในรูปของ file เพื่อที่จะสามารถนำมาแสดงผลข้อมูลภายหลังได้

ตัวอย่างโปรแกรมภาษา C ที่เราใช้ในโครงการ

```
/* Check conversion status */
do  {
      t=0;
      t=inp(0x08);
      h=(t & 0x08 );   /*  Read DMA status  */
      outp(0x0C,0);    /*  Clear DMA f-f */
    }while(h==0);
printf(" the end of the data transmission ");
outp(PORT+3,0);    /* Disable A/D command */
outp(0x0A,0x07);   /* Disable DMA transfer */
```

บทที่ 5 การแสดงผล

การแสดงผล คือ การนำข้อมูลที่บันทึกได้มาแสดง บนจอภาพ (monitor) โดยเราจะใช้การเขียนโปรมแกรมภาษา C ในการนำข้อมูลที่ได้รับการบันทึกลงใน file ในส่วนของการบันทึกผลมาแสดงผลบนจอภาพ โดยใช้การแสดงผลในภาษา C ในภาวะกราฟฟิก ซึ่งจะมีลักษณะการแสดงผลเป็นจุดมาประกอบเป็นรูปภาพ ดังนั้นใน 1 จอภาพ นั้นจะมีจุดมาประกอบเป็นจอภาพตามขนาดต่าง ๆ เช่น 320*200, 640*350 ซึ่งความละเอียดของจอภาพมาก ก็จะมีจำนวนจุดทางแถวและทางคอลัมน์มาก การอ้างถึงตำแหน่งเป็นแถวเป็นคอลัมน์ โดยจุดเริ่มต้นของตำแหน่งจะอยู่ที่ แถวที่ 0 คอลัมน์ที่ 0 และความละเอียดของจอภาพขึ้นอยู่กับคุณภาพของวงจรที่ใช้ในการแสดงผลในภาวะกราฟิก ดังนั้นการเขียนโปรมแกรมในภาวะกราฟฟิก ควรจะต้องรู้ว่าวงจรที่ใช้ในการแสดงผลในภาวะกราฟฟิกของเครื่องของตนเป็นของบริษัทใด เพราะใน C จะมี graphic driver อยู่หลายแฟ้มข้อมูล เพื่อเตรียมไว้ใช้สำหรับวงจรแบบต่าง ๆ แต่ในที่นี้ผู้ทำได้ใช้ graphic driver แบบ EGAVGA.BGI ในการแสดงผล

นอกจากนี้ ฟังก์ชันที่ใช้ในภาวะกราฟฟิก จะเก็บอยู่ในแฟ้มข้อมูลชื่อ graphic.h ดังนั้น เราจึงต้อง #include แฟ้มข้อมูล graphic.h ก่อนเขียนโปรมแกรม

การแสดงผลถ้ายิ่งมีความละเอียดมากยิ่งดีเราจึงใช้ความละเอียดของ VGAHI ซึ่งมีความละเอียด 640*480

5.1 การแสดงผลของ Computer

ในการแสดงผลข้อมูล ข้อความ ตัวอักษร หรือ กราฟฟิก ของ CPU ให้มนุษย์ได้รับรู้นั้น เราจะอาศัยจอภาพ(monitor)ในการแสดงผล เป็นตัวกลางในการตอบรับข้อมูลที่มนุษย์สามารถเข้าใจได้ แต่ในการเชื่อมต่อระหว่าง monitor กับตัว CPU นั้น เราจะอาศัยการ์ดแสดงผล ซึ่งจอแสดงผลนั้นมีหลายแบบหลายชนิด ซึ่งส่วนแสดงผลจะแบ่งตามความละเอียดของตัวอักษร ส่วนแสดงผลจะแบ่งตามความละเอียดของตัวอักษร ความละเอียดของกราฟฟิก อัตราการวาดภาพ จำนวนของสีที่สามารถแสดง และการ์ดที่ใช้ในการแสดงผลดังตารางที่ 5.1

จอแสดงผล	การ์ดแสดงผล	สี	ความละเอียดของตัวอักษร	ความละเอียดของกราฟิกส์	อัตราการกวาดภาพ
จอโมโนโครม	เฮอร์คิวลีส EGA	2	MDA 80 * 25	640 * 350 720 * 350 720 * 348	แนวตั้ง 50 Hz แนวนอน 15.8 KHz
จอสีมาตราฐาน	CGA EGA	16	40 * 25 80 * 25	320 * 200 640 * 200	แนวตั้ง 60 Hz แนวนอน 15.8 KHz
จอ ECD	CGA VGA	16 จาก 64	40 * 25 80 * 25	320 * 200 340 * 200 640 * 350	แนวตั้ง 60 Hz แนวนอน 15.8 KHz หรือ 21.8 KHz
จอดิจิตอล มัลติซิงก์	CGA EGA	16 จาก 64	80 * 25	320 * 200 640 * 200 640 * 350	เปลี่ยนแปลงได้
จออนาลอก มัลติซิงก์	VGA	256 จาก 256K	80 * 25	640 * 480 800 * 600	เปลี่ยนแปลงได้
จอ VGA จอโมโนโครม	VGA	256 จาก 256K	80 * 25	320 * 400 640 * 400 320 * 350 640 * 350 720 * 350 720 * 400 640 * 480	แนวตั้ง 70 Hz แนวนอน 31.5 KHz

ตารางที่ 5.1 จอแสดงผลแบบต่างๆ

จากตารางจะเห็นว่า มีจอแสดงผลและการ์ดแสดงผลมากมาย แต่ที่นิยมใช้กันมาก และใช้กันทั่วไป มักจะใช้การ์ดแสดงผล EGA และ VGA ซึ่งเป็นส่วนที่เราได้ศึกษาพิจารณา ฉะนั้นจึงมุ่งเน้นเนื้อหาไปที่การ์ด EGA และ VGA

การ์ด EGA และ VGA ไม่สามารถประมวลผลได้ด้วยตนเอง แต่จะแสดงผลตามข้อมูลที่มนุษย์ป้อนเข้าไปในหน่วยความจำแสดงผล (bit-mapped displaymemory) ในลักษณะของ Software ซึ่งการ์ด EGA และ VGA มีการทำงานและสถาปัตยกรรมที่เหมือนกัน แต่จะมีความแตกต่างทางด้านประสิทธิภาพ ความสามารถดังแสดงได้ในตารางที่ 5.2

การ์ดแสดงผล	ความละเอียด	ความสามารถในการแสดงสี	การ Interface
EGA	ความละเอียดในแนวนอน 640 จุด แนวตั้ง 350 จุด	แสดงสีได้พร้อมกัน 16 สี จาก 64 สี	สามารถนำไปใช้กับจอแสดงผลทั่วไป
VGA	ความละเอียดในแนวนอน 640 จุด แนวตั้ง 480 จุด	แสดงสีได้พร้อมกัน 256 สี แต่ความละเอียดจะลดลง	ใช้กับจอทั่วไปไม่ได้จึงได้มีการพัฒนาจอสีและโมโนโครมสำหรับ VGA

ตารางที่ 5.2

โหมด	ชนิดของการแสดงผล	สี	ความละเอียด	จอแสดงผล
0 , 1	ข้อความสี	16	40*25 8*8 char cell	CD,BCD,VGA จอซิงค์หลายความถี่
0˚ , 1˚	ข้อความสี	16	40*25 8*14 char cell	BCD,VGA จอซิงค์หลายความถี่
0+ , 1+	ข้อความสี	16	40*25 8*16 char cell	VGA จอซิงค์หลายความถี่
2,3	ข้อความสี	16	80*25 8*8 char cell	CD,BCD,VGA จอซิงค์หลายความถี่
2˚ , 3˚	ข้อความสี	16	80*25 8*14 char cell	BCD,VGA จอซิงค์หลายความถี่
2+ , 3+	ข้อความสี	16	80*25 9*16 char cell	VGA จอซิงค์หลายความถี่
4 , 5	กราฟิกสี	4	320*200	CD,BCD,VGA จอซิงค์หลายความถี่
6	กราฟิกสี	2	640*200	CD,BCD,VGA จอซิงค์หลายความถี่
7	ข้อความโมโนโครม	2	80*25 8*14 char cell	จอโมโนโครม VGA
7+	ข้อความโมโนโครม	2	80*25 9*16 char cell	เฉพาะ VGA
8,9,A	เฉพาะ PC jr			
D	กราฟิกสี	16	320*200	CD,BCD,VGA จอซิงค์หลายความถี่
E	กราฟิกสี	16	640*200	CD,BCD,VGA
F	กราฟิกโมโนโครม		640*350	จอโมโนโครม VGA
10	กราฟิกสี	16	640*350	BCD,VGA จอซิงค์หลายความถี่
11	กราฟิกสี	2	640*480	VGA จอซิงค์หลายความถี่
12	กราฟิกสี	16	640*480	VGA จอซิงค์หลายความถี่
13	กราฟิกสี	256	320*200	VGA จอซิงค์หลายความถี่

ตารางที่ 5.3 โหมดการแสดงผลของการ์ด EGA/VGA

5.2 จอ VGA

ในส่วนที่ได้ศึกษาอยู่นี้ จะต้องมุ่งไปที่การใช้งานที่ใช้กันทั่วไป ฉะนั้นเราจึงมุ่งไปศึกษาเกี่ยวกับการ์ด EGA และ VGA ซึ่งจากตารางที่ 5.1 จะเห็นว่า การ์ด VGA สามารถนำไป Interface กับจอได้หลายชนิด แต่ที่เรามุ่งเน้นก็คือ จอภาพ VGA ซึ่งจอภาพ VGA มีความละเอียดสูง และได้แบ่งออกเป็น จอสี และ โมโนโครม โดยจอโมโนโครมสามารถแสดงรายละเอียดสีในรูปแบบระดับสีเทา และจอทั้งสองแบบสามารถใช้งานทดแทนกันได้ นั่นคือ จอสีสามารถใช้งานแบบโมโนโครมได้ และจอโมโนโครมก็ใช้งานจอสีได้

5.3 โหมดการทำงานมาตรฐานของ EGA และ VGA

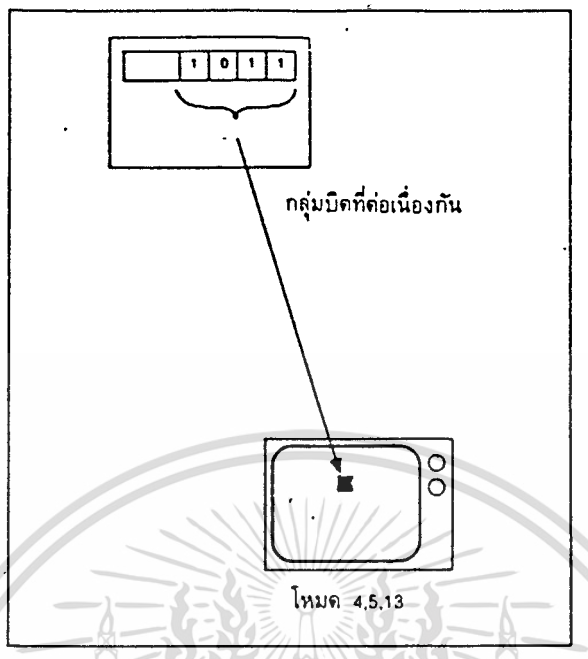
ดังที่กล่าวมาแล้วข้างต้น การติดต่อระหว่างมนุษย์และจอแสดงผล เราอาศัย Software ในการเชื่อมต่อ แต่เนื่องจากมีการ์ดแสดงผลหลายชนิด ซึ่งแต่ละชนิดจะมีคุณสมบัติต่างๆกัน เช่น จำนวนสี ความละเอียด เป็นต้น ซึ่งเป็นข้อมูลสำคัญในการเขียน Software ฉะนั้นการนำ Software ที่ใช้กับจอแสดงผลชนิดหนึ่ง ไม่อาจนำไปใช้กับจอภาพชนิดอื่นได้ ดังนั้นจึงมีการกำหนดมาตรฐานในการทำงานในโหมดต่างๆ ดังตารางที่ 5.3

ข้อมูลต่างๆที่ต้องการแสดงบนจอภาพนั้น จะถูกประมวลผลโดย CPU แล้วเก็บข้อมูลในหน่วยความจำแสดงผลเพื่อแสดงแต่ละจุดภาพออกหน้าจอ โดยการทำงานของ EGA และ VGA นี้เรียกว่า การรีเฟรชจอภาพ (display refresh) ซึ่ง EGA จะทำการแสดงผลซ้ำๆกันด้วยอัตรา 60 ครั้งต่อ วินาที ส่วน VGA ใช้อัตรา 70 ครั้งต่อวินาที

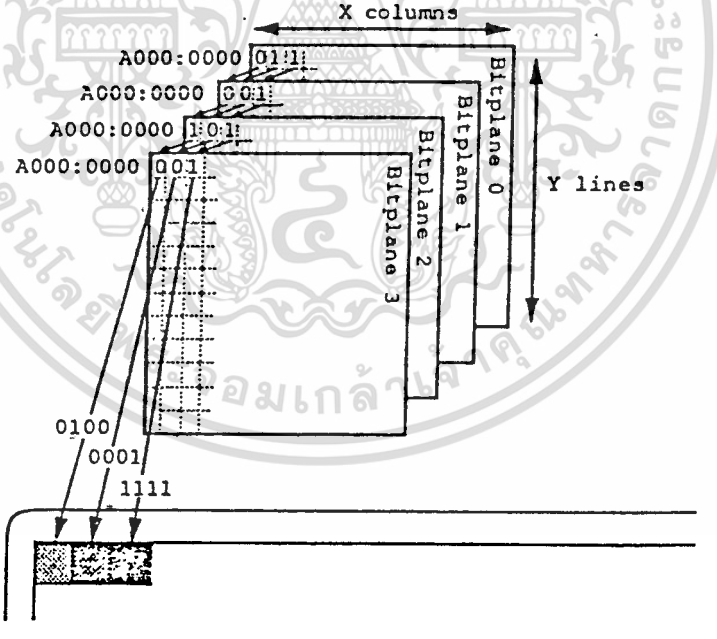
5.4 การเก็บข้อมูลสี มี 2 วิธี คือ

5.4.1. Packed pixels ใช้สำหรับการเก็บข้อมูลจากภาพแต่ละจุดภาพ จะถูกเก็บลงในหน่วยความจำที่ต่อเนื่องกัน ดังรูปที่ 5.1 แสดง Packed pixels แบบ 4 บิทต่อจุดภาพ

5.4.2. Color plane จะแบ่งหน่วยความจำแสดงผลออกเป็นหลายแพลนที่เป็นอิสระต่อกัน โดยแต่ละแพลนจะใช้สำหรับควบคุมสีแต่ละสี และแต่ละบิทบนแพลนจะแทนจุดภาพบนจอแต่ละจุด แล้วรวมข้อมูลที่ตรงกันของทุกแพลนเป็นข้อมูลตัวแสดงผลหนึ่งภาพ ดังรูปที่ 5.2



รูปที่ 1

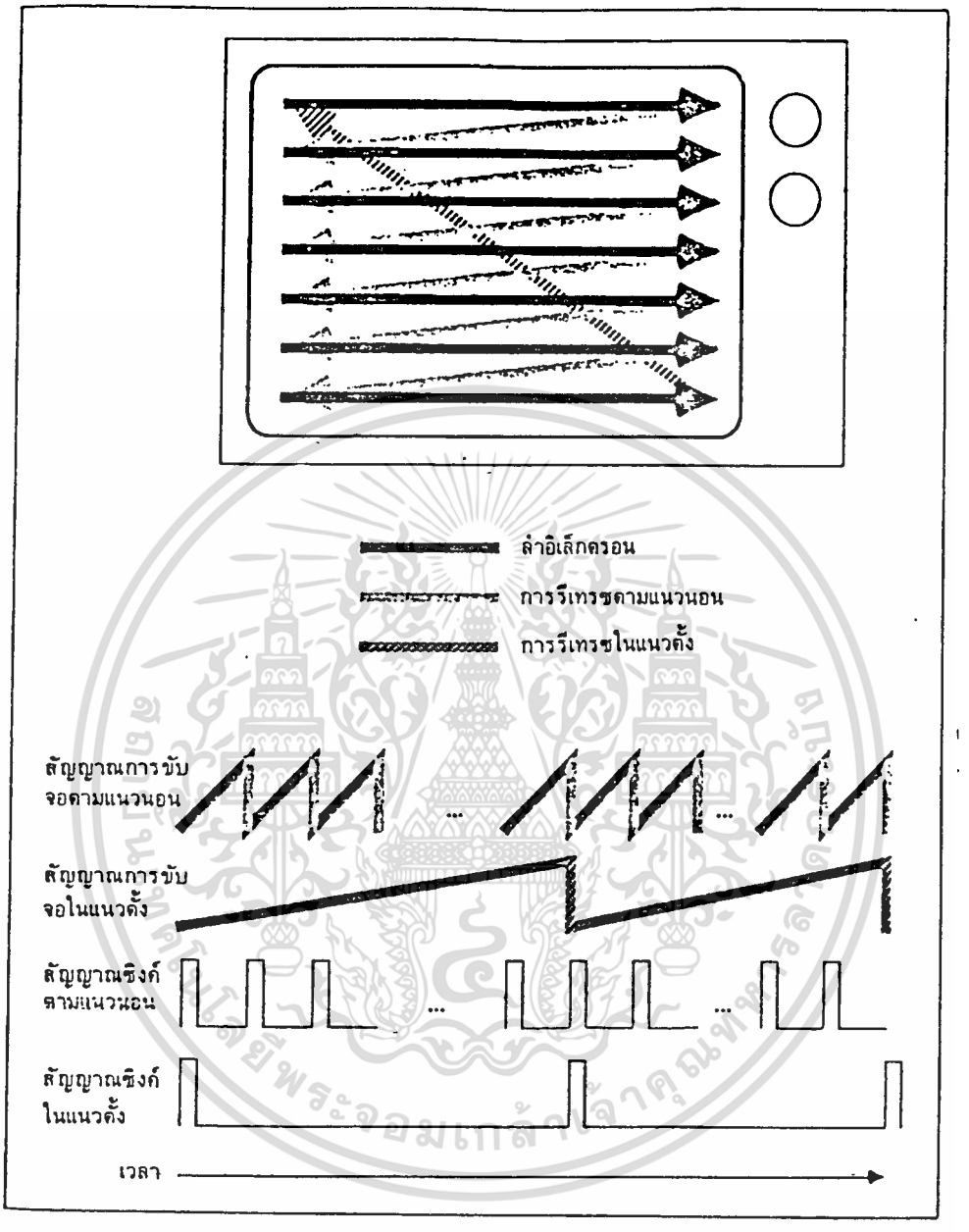


รูปที่ 2

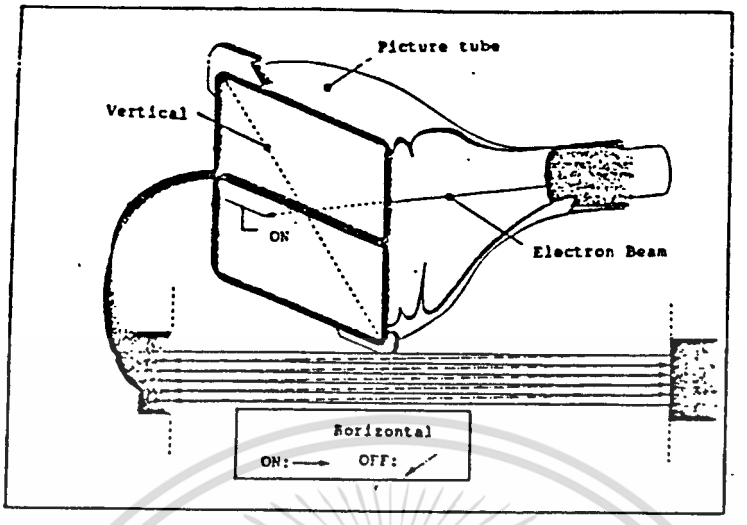
5.5 การทำงานของจอแสดงผล (จอ CRT)

การทำงานของจอ CRT (Cathode ray tube display) ใช้หลักการคล้ายกับโทรทัศน์โดยอาศัยการ Scan ของลำอิเลคตรอนไปกระทบผิวจอที่ฉาบด้วยสารฟอสฟอรัส ซึ่งตำแหน่งแต่ละจุดภาพบนจอ คือ ตำแหน่งของลำอิเลคตรอนที่ Scan ผ่าน และสีต่างๆที่เกิดขึ้นแต่ละจุดคือ ค่า

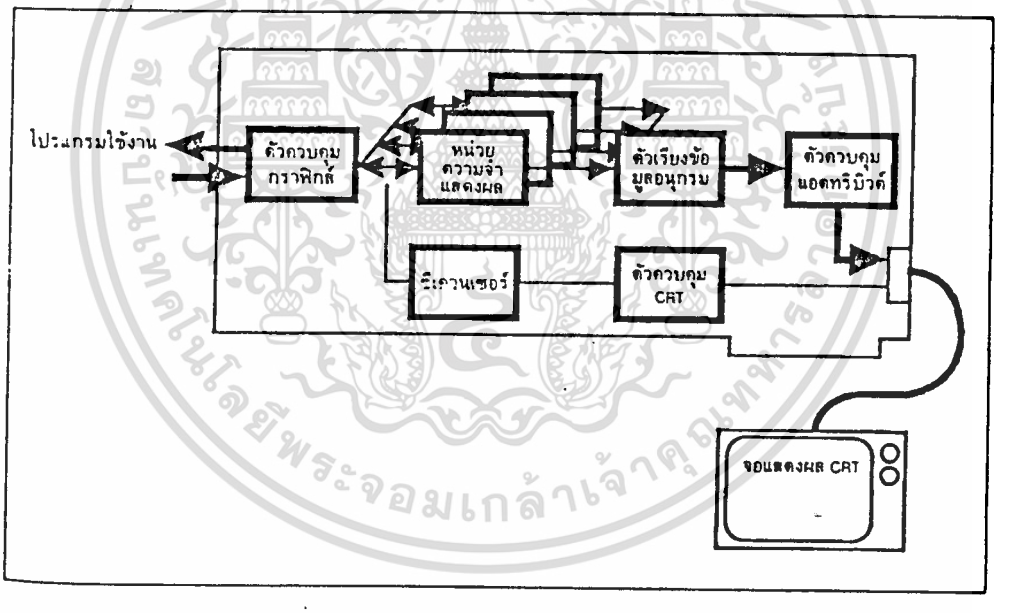
ข้อมูลที่เก็บไว้ในหน่วยความจำแสดงผล ลำอิเลคตรอนจะถูกกวาดซ้ำไปมาด้วยอัตราการกวาดภาพประมาณ 50, 60, หรือ 70 ครั้งต่อวินาที ขึ้นอยู่กับชนิดของจอ ซึ่งเรียกขบวนการนี้ว่า การรีเฟรชจอภาพ (display refrech) และเรียกการกวาดของลำอิเลคตรอนบนจอว่า แรสเตอร์ (raster) ซึ่งเป็นการกวาดลำอิเลคตรอนจากมุมบนซ้ายแล้วกวาดไปทางขวา เมื่อกวาดถึงขวาสุด ก็จะทำการบิดลำอิเลคตรอน (blanking) จากนั้นจะไปเริ่มต้นทางซ้ายใหม่ (retrace) เพื่อกวาดเส้นการสแกนในแนวระดับเส้นถัดไป เมื่อกวาดภาพตามแนวระดับถึงมุมขวาล่าง จะบิดลำอิเลคตรอน จากนั้นจะไปเริ่มต้นที่มุมบนซ้ายใหม่ดังรูปที่ 5.3



รูปที่ 3 การทำงานของจอ CRT



รูปที่ 4 หลักการทำงานของการสแกนของลำแสงอิเล็กตรอน



รูปที่ 5 บล็อกไดอะแกรมของ EGA/VGA

5.6 ฟังก์ชั่นที่ใช้ในภาวะกราฟิกในภาษา C (graphics mode)

จากที่กล่าวไว้ข้างต้น คือ ในการที่เราจะเรียกใช้ในภาวะกราฟิก เราจะต้อง #include แฟ้มข้อมูล graphic.h เสียก่อนจึงจะสามารถเรียกใช้ฟังก์ชั่นเหล่านั้นได้ และในการที่จะเข้าสู่ภาวะกราฟิกจะต้องทำดังนี้

5.6.1 เรียกใช้ฟังก์ชั่น initgraph( )

initgraph( ) เป็นฟังชั่นที่ใช้เพื่อที่จะเข้าสู่ภาวะกราฟิก โดยฟังชั่นนี้จะทำหน้าที่โหลดแฟ้ม

ข้อมูลที่เป็น graphics driver ที่อยู่ในดิสก์มาเก็บไว้ในหน่วยความจำ

รูปแบบ

```
initgraph ( *driver,*mode,*path);
```

dirver เป็นตำแหน่งของตัวแปรจำนวนเต็มที่ทำหน้าที่กำหนดชนิดของแฟ้มข้อมูลที่เป็น graphics driver การกำหนดชนิดของแฟ้มข้อมูลนี้สามารถกำหนดเป็นชื่อแมคโครหรือ ค่าคงที่ได้

mode เป็นการกำหนดขนาดของความละเอียดในการแสดงผลของแต่ละ graphics driver ซึ่งสามารถกำหนดเป็นชื่อแมคโครหรือ ค่าคงที่ได้

path คือการกำหนดเส้นทางเพื่อค้นหาแฟ้มข้อมูล graphics driver โดยเขียนอยู่ภายในเครื่องหมายคำพูด ถ้าไม่กำหนดถือว่าแฟ้มข้อมูล graphics driver นั้นอยู่ที่ working directory

ในโครงการนี้เราใช้กราฟิก driver ของ EGAVGA.BGI โดยให้มีความละเอียด 640*480 จุด โดยการเขียนโปรแกรมในการเรียกใช้ ฟังก์ชั่น initgraph มีดังนี้

```
int gdriver = DETECT ,gmode,errocode;
initgraph (&gdriver,&gmode,"");
```

5.6.2 ฟังก์ชั่น closegraph( ) , restoremode( )

เป็นฟังชั่นที่ใช้ยกเลิกภาวะกราฟิก กลับไปสู่ภาวะข้อความ

รูปแบบ

```
closegraph( );
```

หรือ

```
restoremode( );
```

5.6.3 ฟังก์ชั่น line

เป็นฟังก์ชั่นที่ใช้ในการวาดรูปเส้นตรง โดยจะต้องกำหนดจุดเริ่มต้น และจุดสุดท้ายของเส้น

รูปแบบ

```
line (startx,starty,endx,endy);
```

startx ,starty เป็นจุดเริ่มต้นของเส้น

endx ,endy เป็นจุดสุดท้ายของเส้น

5.6.4 ฟังก์ชั่น rectangle ( )

เป็นฟังก์ชั่นที่ใช้สร้างรูปสี่เหลี่ยม

รูปแบบ

```
rectangle( left ,top ,right ,bottom);
```

left ,top เป็นการกำหนดจุดเริ่มต้นของมุมซ้ายด้านบนของรูปสี่เหลี่ยม โดย left เป็นตำแหน่งคอลัมน์ top เป็นตำแหน่งแถว

right ,bottom เป็นการกำหนดจุดสุดท้ายของมุมขวาด้านล่างบนของรูปสี่เหลี่ยม โดย right เป็นตำแหน่งคอลัมน์ bottom เป็นตำแหน่งแถว

5.6.5 ฟังก์ชั่น putpixel ( )

เป็นฟังก์ชั่นให้สีกับจุดที่ตำแหน่งต่าง ๆ ตามต้องการ

รูปแบบ

```
putpixel( X , Y , color);
```

X ,Y เป็นตำแหน่งที่ต้องการระบายสี

color เป็นสีที่ต้องการระบาย

5.6.6 ฟังก์ชั่น setcolor ( )

เป็นฟังก์ชั่นให้สีกับจุดที่ต้องการระบายสี

รูปแบบ

```
setcolor( color );
```

color เป็นสีที่ต้องการใช้วาดรูป

เราได้กล่าวถึงฟังก์ชั่นที่จำเป็นสำหรับโครงการนี้ไปแล้วในข้อ 5.6 ตอนต่อไปนี้จะเสนอถึงวิธีการ plot ที่คณะผู้จัดทำได้เขียนโปรแกรมขึ้นมาเพื่อใช้ในการแสดงผล คลื่นไฟฟ้าหัวใจ

5.7 การ Plot

คณะผู้จัดทำได้ใช้วิธีในการเขียนรูปด้วยการเขียน ทับซ้ำ ๆ กันไป เรื่อย ๆ เนื่องจากว่าเราไม่สามารถเขียนสัญญาณที่เข้าโดยแสดงผลคลื่นทั้งหมดที่เกิดขึ้น ดังนั้นเมื่อเราทำการ plot ค่าข้อมูลที่มีอยู่จนเต็มจอภาพ (คณะผู้จัดได้กำหนดไว้ให้การ plot มีความยาวเพียงแค่ 400 ค่า) เราจะต้องลบรูปเก่าทิ้งเพื่อที่จะแสดงผลของสัญญาณที่เข้ามาใหม่ โดยเราจะลบค่าข้อมูลที่เก่าที่สุด ออกเป็นอันดับแรก จากนั้นเราจะ plot ค่าใหม่ที่ได้ลงที่ตำแหน่งที่เพิ่งจะลบค่าเก่าทิ้ง ทำเช่นนี้เรื่อยไป ดังนั้นภาพที่ปรากฏขึ้นบนจอภาพก็จะมองเห็นการ plot ค่าข้อมูลใหม่ไล่กันไปเรื่อย ๆ โดยวิ่งไปใน

แนวเดียวกัน การเขียน flow chart ได้ดังนี้

รูปที่ 6 flow chart แสดงการขั้นตอนการทำงานของการ plot

โปรแกรมในส่วนของการ plot

```
#include <graphics.h>
void plotgraph(int count);
BOOL opengraph(void);
BOOL opengraph(void)
{
  int gdriver = DETECT,gmode,errorcode;
  initgraph(&gdriver, &gmode, "");          /* initialize graphics mode */
  errorcode = graphresult();            /* read result of initialization */
  if (errorcode != grOk)                /* an error occurred */
```

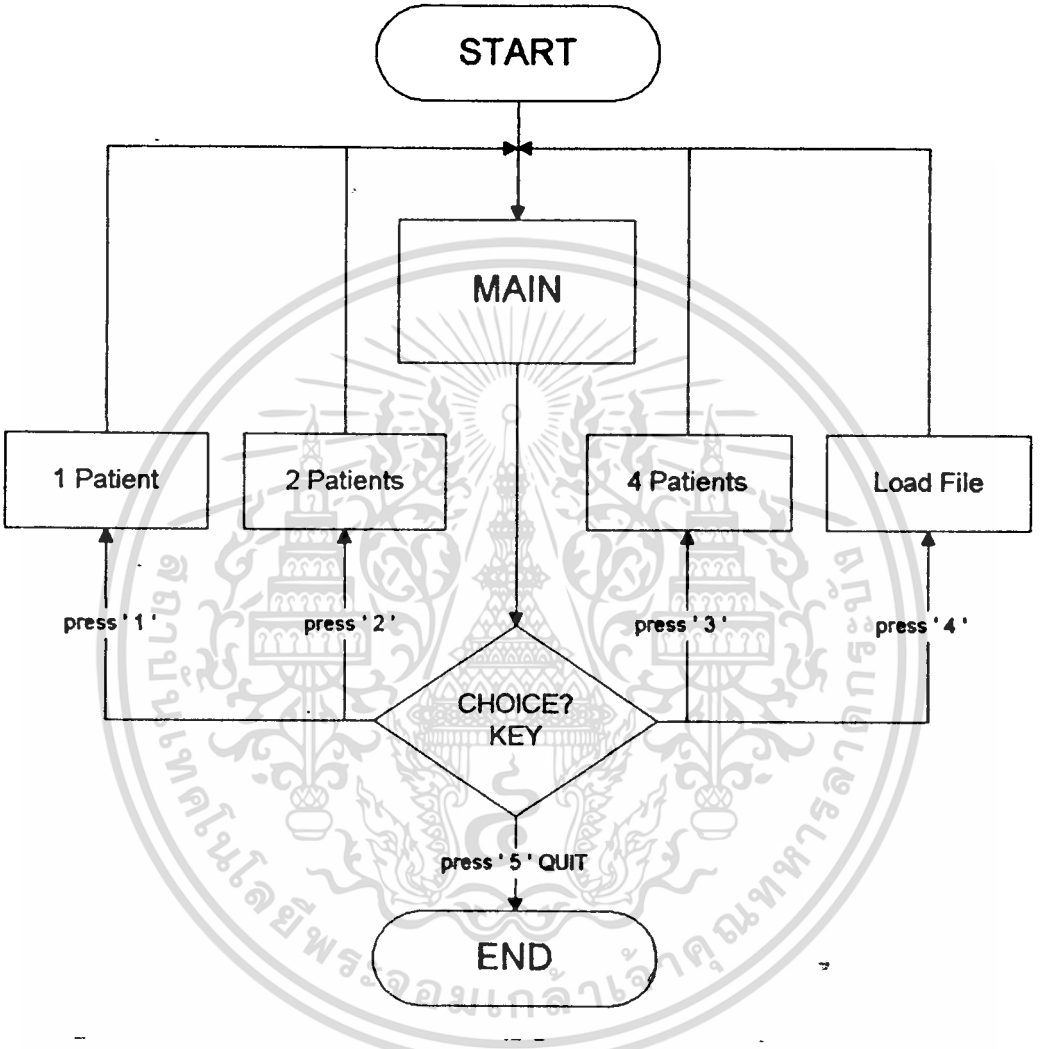
```
            return FALSE;
    return TRUE; }

    void plotgraph(int count)
  {
     int a=0;
      while(a<count)
      {
        setcolor(BLACK);
        line((ice+120),116-s[ice],(ice+121),116-s[ice+1]);
         s[ice]=ss[a];
         a++;
      if (ice==0)
        {
         putpixel(ice+120,116-s[ice],WHITE);
         delay(10);
        }
      else
        {
         setcolor(WHITE);
         line(ice+119,116-s[ice-1],ice+120,116-s[ice]);
         delay(10);
        }
        ice++;
      }
      if(ice==399)
      {
        /* setcolor(WHITE);
            line(399+119,116-s[398],399+120,116-s[399]); */
        ice=0;
      }
  }
```

ผลการทดลอง

เนื่องจากการที่จะนำสัญญาณ ECG มาวัดและทำการทดลองจริงจะต้องใช้ความระมัดระวังมาก เพราะอาจเกิดอันตรายต่อตัวผู้ที่ถูกทดลอง ในส่วนของการทดลอง เราจึงใช้การจำลองสัญญาณ ECG แทนการวัดสัญญาณจริง ๆ โดยใช้เครื่อง Simmulate สัญญาณ ECG แทน โดยเสมือนหนึ่งว่าเราได้วัดสัญญาณนี้จริง ๆ แล้วนำสัญญาณไป เข้าเครื่องวัดสัญญาณไฟฟ้าหัวใจ จากนั้นนำไปเข้า การ์ด PA-MA12(-H) และนำไปแสดงผลและเก็บข้อมูลเป็น file โดย computer ผลการทดลองที่เกิดขึ้นคือ

1. สามารถ Interface การ์ด PA-MA12(-H) ได้
2. สามารถเขียนโปรแกรมควบคุมการทำงานของการ์ดได้
3. การ transfer ข้อมูลเราเลือกใช้การ transfer ข้อมูลแบบ DMA และเราสามารถเขียนโปรแกรมควบคุมได้
4. เราสามารเลือกค่าการ Sampling Rate ในการ transfer ข้อมูลได้
5. สามารถนำสัญญาณเข้าการ์ดและทำการ transfer ข้อมูลเป็นสัญญาณ digital ได้
6. นำข้อมูลที่ได้นี้มาแสดงผลบนจอภาพได้
7. สามารถบันทึกผลของข้อมูลในรูปของ file ได้
8. สามารถนำข้อมูลที่แสดงผลไปแล้วขึ้นมาแสดงผลภายหลังได้อีกครั้ง
9. สามารถวัดคลื่นหัวใจได้พร้อมกันถึง 4 ช่องสัญญาณ และ สามารถเก็บข้อมูลของสัญญาณคลื่นหัวใจทั้ง 4 ช่องได้
10. สามารถคำนวณค่า Heart Rate และแสดงค่า Heart Rate บนจอภาพได้

โปรแกรมที่ใช้นี้สามารถเขียนเป็น Flow Chart ได้ดังนี้



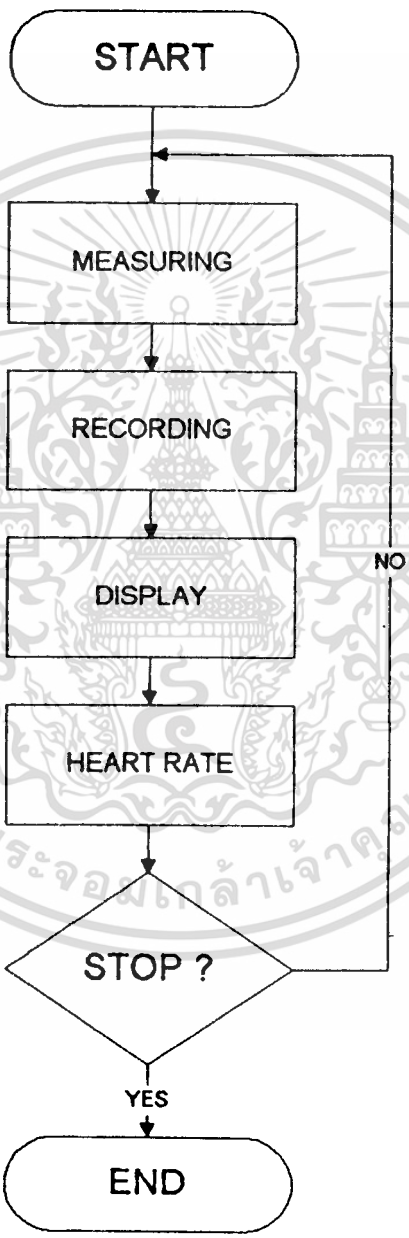
รูปที่ 1 แสดง Flow chart โปรแกรมหลัก

เป็นโปรแกรมหลักที่ใช้ในการเลือกที่จะทำงานว่าจะเลือกวัดคลื่นไฟฟ้าหัวใจกับผู้ป่วยกี่คน ในโปรแกรมนี้สามารถรับผู้ป่วยได้สูงสุดคือ 4 คนพร้อมกัน นอกจากนี้ยังสามารถเรียกข้อมูลของคลื่นไฟฟ้าหัวใจที่เราได้ทำการบันทึกเป็น File เรียบร้อยแล้วขึ้นมาแสดงผลใหม่ได้อีกด้วย โดยการทำงานของโปรแกรมหลักนี้ใช้หลักการกด keyboard เลือกการทำงานที่ต้องการ ส่วนของการวัดจะอธิบายรายละเอียดได้ดัง Flow chart รูปที่ 2

Load File

ในส่วนของการ load file นี้เราจะทำการเปิด file ที่เราเก็บไว้ขึ้นมาแสดงผลโดยการ plot โดยหลักการ plot ใช้หลักการเดิมที่ได้กล่าวมาแล้วใน บทที่ 5

เราสามารถอธิบายการทำงานของการวัดคลื่นไฟฟ้าหัวใจได้ดัง Flow chart ข้างล่าง



รูปที่ 2 แสดง Flow chart การทำงานของ โปรแกรมการวัดคลื่นไฟฟ้าหัวใจ

เราสามารถอธิบายการทำงานได้ดังนี้

MEASURING

เป็นส่วนของโปรแกรมในการ Interface การ์ด PA-MA12 กับ computer หรือโปรแกรมในส่วนนี้คือการทำงานของ DMA นั่นเองโดยการที่โปรแกรมนี้จะทำงานต้องมีการใส่ค่าสัญญาณที่จะทำการ transfer เสียก่อนรายละเอียดของโปรแกรมนี้ได้อธิบายไว้ในบทที่ 4 แล้ว

RECORDING

เป็นส่วนของโปรแกรมในการบันทึกค่าที่ได้จากการ transfer ในข้อ 2 โดยค่าที่ได้จากข้อ 2 จะอยู่ใน memory โปรแกรมในส่วนนี้จะนำค่าใน memory นี้ไปบันทึกเก็บไว้เป็น file ดังนั้นส่วนของโปรแกรมในข้อนี้จะเป็นส่วนของการเขียนโปรแกรมในการ open file และการอ่านค่าไปบันทึกไว้ใน file เป็นส่วนในการติดต่อระหว่างหน่วยความจำหลักกับหน่วยความจำสำรอง

DISPLAY

เป็นส่วนของการแสดงผลข้อมูลที่ได้จากการ transfer เรียบร้อยแล้ว นำมา plot แสดงผลในรูปของกราฟิก ซึ่งรายละเอียดได้อธิบายไว้ในบทที่ 5 แล้ว

ซึ่งหลังจากการแสดงผลแล้วถ้ายังไม่มีการ Interrupt จาก keyboard โปรแกรมจะทำงานในส่วนของทั้ง 3 นี้วนเวียนไปเรื่อย ๆ จนกว่าจะมีการกด keybord อยุดการทำงานและการทำงานของการวัดนี้จะมีการ คำนวณค่า Heart Rate และนำมาแสดงผลบนจอภาพอีกด้วย

HEART RATE

ในส่วนของการคำนวณ ค่า Heart Rate นั้น การทำงานในการหาค่า Heart Rate นั้นจะมีการคำนวณทุกครั้งที่ทำการ plot ไปจนเต็มจอภาพ เราสามารถอธิบายการคำนวณได้ดังนี้

เนื่องจากเราทราบว่าค่า Heart Rate ก็คืออัตราการเต้นของหัวใจ ซึ่งเราสามารถทราบได้การทำงาน 1 ลูกคลื่นของคลื่นไฟฟ้าหัวใจ ดังนั้นถ้าเรานับส่วนที่เป็นยอดคลื่นของ คลื่นไฟฟ้าหัวใจเราก็จะสามารถหาค่า Heart Rate ได้ และการคำนวณก็จะเทียบการนับลูกคลื่น กับค่าความถี่ที่ใช้ในการ Sampling เราใช้ค่าการ Sampling Rate 100 Hz ดังนั้นเราจะได้ว่า ข้อมูลที่เก็บได้ 100 ค่า จะมีค่าเวลาเท่ากับ 1 วินาทีแต่การ plot บนจอภาพในโครงการนี้เรา plot 1 หน้าจอใช้ความยาว 550 จุด ค่าเวลาที่เกิดขึ้นในจำนวนข้อมูลในจอภาพจะเท่ากับ 5.5 วินาทีดังนั้น ถ้าเรานับลูกคลื่นได้ทั้งหมดกี่ลูกแล้วนำไปคำนวณ เราก็จะได้ Heart Rate แต่การคำนวณเพียงแค่นี้จะได้ค่า Heart Rate ที่เกิดจากการประมาณค่า ซึ่งได้ค่าที่ไม่ละเอียดเท่าที่ควรเราจึงใช้วิธี นับคลื่นหัวใจแล้วเปรียบเทียบอัตราส่วนตามรอบที่เพิ่มขึ้นโดยอธิบายได้ดังนี้

รอบที่ 1 ค่าเวลาที่เกิดขึ้นเป็น 5.5 ดังนั้น Heart Rate = $\frac{\text{จำนวนลูกคลื่น} * 6000}{550 * 1}$

รอบที่ 2 ค่าเวลาที่เกิดขึ้นเป็น 5.5*2 ดังนั้น Heart Rate = $\frac{\text{จำนวนลูกคลื่นที่นับได้} * 6000}{550 * 2}$

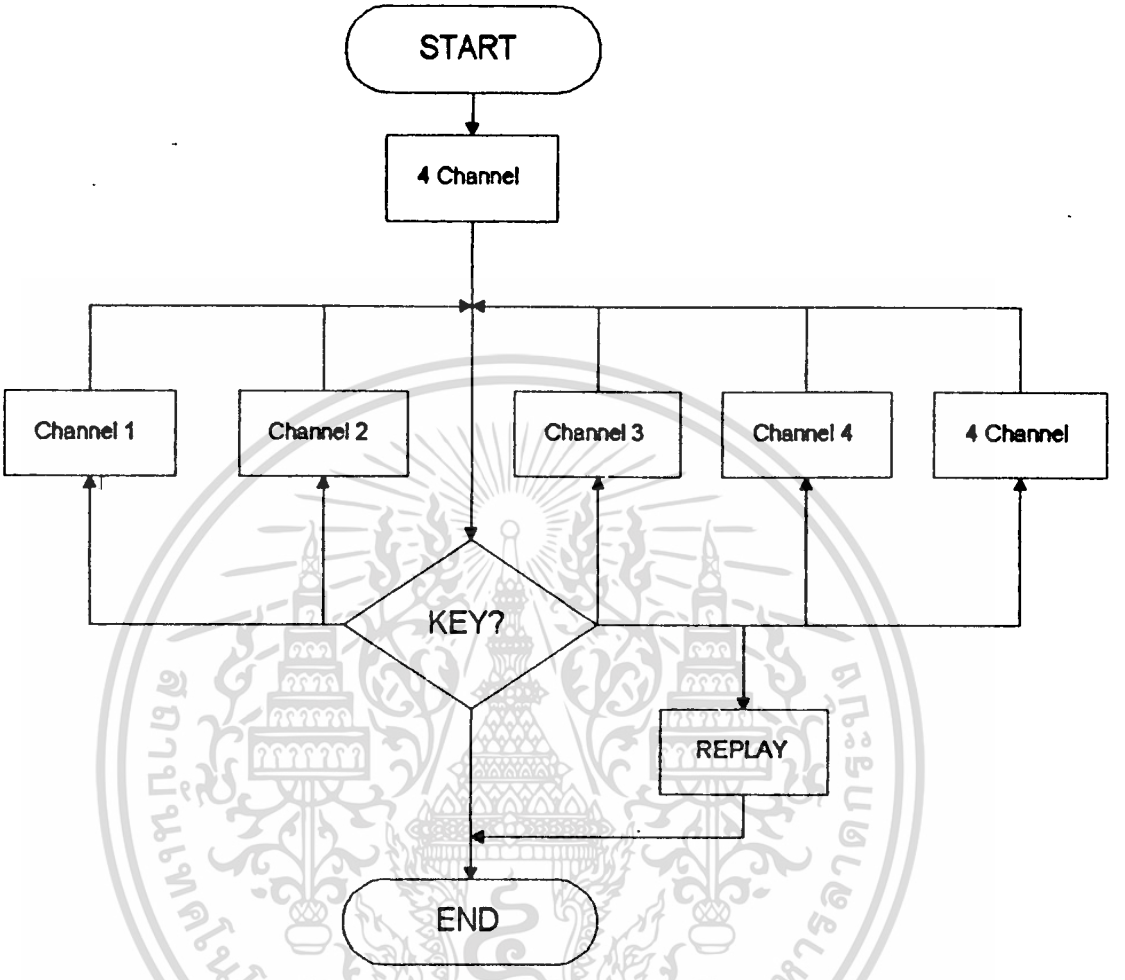
รอบที่ 1 ค่าเวลาที่เกิดขึ้นเป็น 5.5*3 ดังนั้น Heart Rate = $\frac{\text{จำนวนลูกคลื่นที่นับได้} * 6000}{550 * 3}$

จนถึงรอบที่ 11

รอบที่ 11 ค่าเวลาที่เกิดจะใกล้เคียงกับ 1 นาที ดังนั้น Heart Rate = จำนวนลูกคลื่นที่นับได้ทั้งหมด

จากนั้นเราจะเริ่มนับใหม่อีกครั้ง ด้วยวิธีนี้ทำให้เราสามารถวัด Heart Rate ของคลื่นหัวใจที่จะเกิดความเปลี่ยนแปลงได้ทันเวลา โดยการนับลูกคลื่นจะใช้การกำหนดค่า กลางค่าหนึ่งที่มีค่าเกินลูกคลื่นเล็ก ๆ ของคลื่นไฟฟ้าหัวใจที่เกิดขึ้นแต่มีค่าไม่เกินค่าสูงสุดของคลื่นไฟฟ้าหัวใจ แล้วใช้การนับโดยเปรียบเทียบค่าว่าเมื่อไหร่ที่ค่าข้อมูล มีค่ามากกว่าค่าที่กำหนดนี้แล้วเราจะทำการเพิ่มค่าตัวนับขึ้น 1 และจะไม่มีการนับอีกจนกว่าค่าข้อมูลจะต่ำกว่า ค่าที่กำหนดแล้วค่อย ๆ เพิ่มขึ้นมาจนมากกว่า ค่าที่กำหนดอีกครั้ง

นอกจากนี้เราจะอธิบายรายละเอียดของการวัดคลื่นไฟฟ้าหัวใจพร้อมกัน 4 channel ได้ดัง Flow chart ต่อไป



รูปที่ 3 Flow chart แสดงการวัดคลื่นไฟฟ้าหัวใจพร้อมกัน 4 channel

เราจะอธิบายการทำงานของ Flow Chart ได้ดังนี้

4 Channel

เป็นการทำงานที่วัดคลื่นไฟฟ้าหัวใจทั้ง 4 channel แล้วมาแสดงผลบนจอภาพทั้ง 4 channel

Channel 1

เมื่อเราเลือกการทำงานนี้ การเก็บข้อมูลของคลื่นไฟฟ้าหัวใจจะยังคง เก็บข้อมูลทั้ง 4 channel แต่เราจะนำสัญญาณ ใน Channel ที่ 1 มาแสดงผลเพียง Channel เดียวเท่านั้นโดยการ plot จะ plot คลื่นสัญญาณไฟฟ้าหัวใจจนเต็มจอภาพ

นอกจากนี้ใน ส่วนของ Channel อื่น ๆ ก็จะนำค่าของ สัญญาณใน channel นั้น ๆ มาแสดงผลเช่นเดียวกับ Channel 1

REPLAY

โปรแกรมในส่วนนี้เป็นการนำข้อมูลที่ได้บันทึกไว้ในขั้นตอนการ RECORDING มาแสดงผลใหม่อีกรอบโดยโปรแกรมจะมีการรับค่าการแสดงผลย้อนหลัง โดยมีหน่วยเป็น นาที จากนั้นจะไปนำข้อมูลที่ย้อนไปตามเวลาที่เราต้องการมาแสดงผลเป็นรูปของกราฟิกอีกครั้ง

โปรแกรมในส่วนนี้จะทำงานโดยตรวจสอบค่าของ EOF (End of File) โดยการทำงานของส่วนนี้จะ plot รูปไปเรื่อย ๆ จนกระทั่งหมด file การทำงานของโปแกรมก็จะจบลง การทำงานจะกลับไปสู่ MAIN อีกครั้งเพื่อรอที่จะทำการ run โปรแกรมอีกด้วยการกด keyboard ใหม่อีกครั้ง หรือ จบการทำงานด้วยการกด keyboard เลข 5

โปรแกรมที่ใช้ในการทำงานทั้งหมดนี้คือ

```
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#define BOOL  int
#define FALSE 0
#define TRUE  1
#define PORT 0x2E0     /* PA-MA12(-H) port address   */
#define PORT1 0x2E1    /* PA-MA12(-H) port address   */
#define PORT2 0x2E2    /* PA-MA12(-H) port address   */
#define PORT3 0x2E3    /* PA-MA12(-H) port address   */
#define PORT4 0x2E4    /* PA-MA12(-H) port address   */
#define PORT5 0x2E5    /* PA-MA12(-H) port address   */
#define PORT6 0x2E6    /* PA-MA12(-H) port address   */
#define PORT7 0x2E7    /* PA-MA12(-H) port address   */
#define PORT8 0x2E8    /* PA-MA12(-H) port address   */
#define PORT9 0x2E9    /* PA-MA12(-H) port address   */
#define PORT10 0x2EA    /* PA-MA12(-H) port address   */
#define PORT11 0x2EB    /* PA-MA12(-H) port address   */
#define PORT12 0x2EC    /* PA-MA12(-H) port address   */
#define PORT13 0x2ED    /* PA-MA12(-H) port address   */
#define PORT14 0x2EE    /* PA-MA12(-H) port address   */
#define PORT15 0x2EF    /* PA-MA12(-H) port address   */
#define BA 0x0000
#define BL 128  /*A/D buffer address and length*/
void plotgraph1(int count,unsigned int *x1);
void plotgraph2(int count,unsigned int *x1,unsigned int *x2);
void plotgraph3(int count,unsigned int *x1,unsigned int *x2,unsigned int *x3,unsigned int *x4);
void menu1(void);
BOOL opengraph(void);
unsigned int BS= 0x8;
unsigned char far *mem;
long double ic,second=0;
char c1,c2;
```

```
unsigned long int q,ss=0;
unsigned int ss1[330],ss2[330],ss3[330],ss4[330],ss=0;
unsigned int heart1=0,heart2=0,heart3=0,heart4=0;
int pop=0,rr=0,mm=0;
int ss1,ss2,ss3,ss4;
char hr1[5],hr2[5],hr3[5],hr4[5];
void main(void)
{
 do
 {
 clrscr(),
 textmode(3);
 gotoxy(10,3),
 textcolor(WHITE),
 cprintf("*****************************************************************"),
 gotoxy(18,5),
 highvideo(),
 textcolor(LIGHTCYAN);
 cprintf(" EKG Measuring Recording and Displaying by computer ");
 normvideo(),
 textcolor(LIGHTGREEN),
 gotoxy(20,6),
 cprintf("          By 54106312 & 54102093 "),
 textcolor(LIGHTBLUE);
 gotoxy(18,8),
 cprintf("Electronic Department , Faculty of Engineering "),
 textcolor(LIGHTRED),
 gotoxy(16,9),
 cprintf("King Mongkut's Institute of Technology Ladkrabang "),
 textcolor(YELLOW),
 gotoxy(20,11),
 cprintf("          Project Engineering "),
 textcolor(WHITE),
 gotoxy(10,13),
 cprintf("*****************************************************************"),
 gotoxy(20,15),
 textcolor(WHITE),
 cprintf(" Select Operation   "),
 gotoxy(20,16),
 cprintf("  1  RUN ECG with one patient "),
 gotoxy(20,17),
 cprintf("  2  RUN ECG with two patient "),
 gotoxy(20,18),
 cprintf("  3  RUN ECG with four patient "),
 gotoxy(20,19),
 cprintf("  4  LOAD patient's data for display "),
 gotoxy(20,20),
 cprintf("  5  QUIT "),
 textcolor(BLINK+LIGHTMAGENTA),
 gotoxy(20,22);
 cprintf(" Select 1 , 2 , 3 , 4 , 5   "),
 textcolor(WHITE),
 c1 = getch(),
 switch(c1)
 {
  case '1'  main1(),
  break,
  case '2'  main1();
  break,
  case '3'  main1();
  break;
  /* case '4' . main20;
```

```
   break;  */
  default : cprintf("\n");
  }
} while(c!='5');
textmode(-1);
clear();
}
void main1(void)
{
/*-------------------------------------------
            *   Program Name  : DMA-3.C
            *   Model Number  : PA-MA12(-H)
            *   Language      : C
            *   Function      : A/D sample program (DMA Transfer, 8254 trig)
            *   Date          : Jan 01, 1993
            *   Copyright (c) Acrotek Corporation, 1991 - 1994
 -------------------------------------------*/
int P[10][10],D[10][10],
unsigned int ch,i,con,T1,T2,num,AP=0,AM=0,AL=0,count,
unsigned int s1[550],s2[550],s3[550],s4[550],
unsigned char b=0,j=0;
unsigned long int size,size1,size2,size3,size4,
FILE *infile1,*infile2,*infile3,*infile4,
unsigned char c11,c12,c13,c14,
char file1[12],file2[12],file3[12],file4[12],c3,
int col,x,y,xx=0,ib=0,
clear();
count=4,
 while(ix<550)
 {
   s1[ix] = 50,
   s2[ix] = 50,
   s3[ix] = 50,
   s4[ix] = 50,
   ix++,
 }
/* 8 bit DMA port address ---------------------------------*/
P[0][0]=0x87 , P[1][0]=0x83 , P[3][0]=0x82, /*  DMA page register address  */
P[0][1]=0x00 , P[1][1]=0x02 , P[3][1]=0x06, /*  DMA current address register  */
P[0][2]=0x01 , P[1][2]=0x03 , P[3][2]=0x07, /*  DMA word counter register  */
P[0][3]=0x08 , P[1][3]=0x09 , P[5][3]=0x08, /*  Byte-transfer DMA status register */
P[0][4]=0x0A , P[1][4]=0x0A , P[3][4]=0x0A, /*  Byte-transfer DMA mask register  */
P[0][5]=0x0B , P[1][5]=0x0B , P[3][5]=0x0B, /*  Byte-transfer DMA mode register  */
P[0][6]=0x0C , P[1][6]=0x0C , P[3][6]=0x0C, /*  Byte-transfer DMA clear flip-flop register  */

/*  8 bit DMA commands ------------------------------------- */
D[0][0]=0x44 , D[1][0]=0x45 , D[3][0]=0x47,/*  DMA mode command   */
D[0][1]=0x00 ; D[1][1]=0x01 , D[3][1]=0x03,/*  DMA enable command  */
D[0][2]=0x04 ; D[1][2]=0x05 , D[3][2]=0x07,/*  DMA disable command */
D[0][3]=0x01 , D[1][3]=0x02 , D[3][3]=0x08,/*  DMA complete status */
/* 16 bit DMA port address ---------------------------------- */
P[5][0]=0x8B , P[6][0]=0x89 ; P[7][0]=0x8A,/* DMA page register address */
P[5][1]=0xC4 , P[6][1]=0xC8 , P[7][1]=0xCC,/* DMA current address register */
P[5][2]=0xC6 , P[6][2]=0xCA , P[7][2]=0xCE,/* DMA word counter register */
P[5][3]=0xD0 , P[6][3]=0xD0 , P[7][3]=0xD0,/* Word-transfer DMA status register */
P[5][4]=0xD4 , P[6][4]=0xD4 , P[7][4]=0xD4,/* Word-transfer DMA mask register  */
P[5][5]=0xD6 , P[6][5]=0xD6 , P[7][5]=0xD6,/* Word-transfer DMA mode register  */
P[5][6]=0xD8 , P[6][6]=0xD8 ; P[7][6]=0xD8,/* Word-transfer DMA clear flip-flop register */

/* 16 bit DMA commands ------------------------------------- */
D[5][0]=0x45 , D[6][0]=0x46 , D[7][0]=0x47,/* DMA mode command */
D[5][1]=0x01 , D[6][1]=0x02 ; D[7][1]=0x03;/* DMA enable command */
```

```
D[5][2]=0x05 , D[6][2]=0x06 , D[7][2]=0x07 /* DMA disable command */
D[5][3]=0x02 , D[6][3]=0x04 , D[7][3]=0x08 /* DMA complete status  */
 switch(c1)
 {
 case '1'
 { printf("enter your file you want to save in window 1 (********.acg) ");
   scanf("%s",file1);
   infile1 = fopen (file1,"wb");
 }
 break;
 case '2'
 {
   printf("enter your file you want to save in window 1 (********.acg) ");
   scanf("%s",file1);
   infile1 = fopen (file1,"wb");
   printf("enter your file you want to save in window 2 (********.acg) ");
   scanf("%s",file2);
   infile2 = fopen (file2,"wb");
 }
 break;
 case '3'
 {
   printf("enter your file you want to save in window 1 (********.acg) ");
   scanf("%s",file1);
   infile1 = fopen (file1,"wb");
   printf("enter your file you want to save in window 2 (********.acg) ");
   scanf("%s",file2);
   infile2 = fopen (file2,"wb");
   printf("enter your file you want to save in window 3 (********.acg) ");
   scanf("%s",file3);
   infile3 = fopen (file3,"wb");
   printf("enter your file you want to save in window 4 (********.acg) ");
   scanf("%s",file4);
   infile4 = fopen (file4,"wb");
 }
 break;
 }
/* Select DMA channel                                    */
printf(" PA MA12 DAM Transfer sample program \n");
printf ("Select DMA channel  \n");
printf (" 8 bit  (Byte transfer) DMA channel  0, 1 and 2\n");
printf (" 16 bit (Word transfer) DMA channel  5, 6 and 7  ");
scanf("%d",&ch);
printf("\n");
/* Determine conversion frequency ------  ----  ---------- ---*/
printf ("Select Conversion frequency  \n" );
printf (" 1 100KHz  2 50KHz  3 10KHz  4 1KHz  5 100Hz  6 10Hz  7 1Hz  8 0.5Hz  ");
scanf("%d",&con);
printf("\n");
if (con==1){
  T2=4   , T1=10   ,  /* 100KHz */ }
if (con==2){
  T2=4   , T1=20   ,  /* 50KHz */ }
if (con==3){
  T2=4   , T1=100  ,  /* 10KHz */ }
if (con==4){
  T2=4   , T1=1000 ,  /*  1KHz */ }
if (con==5){
  T2=40  , T1=1000 ,  /*  100Hz */ }
if (con==6){
  T2=40  , T1=10000 ,  /* 10Hz */ }
if (con==7){
```

```
 T2=400  , T1=10000 ,  /* 1Hz */ }
if (conv==8){
 T2=800  , T1=10000 ;  /* 0.5Hz */ }
 switch(c1)
{
case '1'
{
  num=0x00,
}
break,
case '2'
{
 num=0x10,
}
break,
case '3'
{
 num=0x30,
}
break,
}
 lcm
 if (opengraph()==TRUE)
{
 switch(c1)
 {
              case '1'
              {
 setfillstyle(SOLID_FILL,RED),
 bar(0,0,639,67),
 setcolor(WHITE),
 outtextxy(120,15,"ECG Measuring Recording and Displaying by computer"),
 rectangle(0,68,551,169),
 rectangle(0,170,551,271),
 rectangle(0,272,551,373),
 rectangle(0,374,551,475),
 setfillstyle(SOLID_FILL,12),
 bar(552,68,640,475),
 outtextxy(560,70,"Channel 1"),
 outtextxy(560,85,"Heart Rate"),
 gu=1,
 }break,
 case '2'
 {
 setfillstyle(SOLID_FILL,RED),
 bar(0,0,639,67),
 setcolor(WHITE),
 settextstyle(DEFAULT_FONT,HORIZ_DIR,3),
 outtextxy(120,15,"ECG Measuring Recording and Displaying by computer"),
 rectangle(0,0,639,67),
 rectangle(0,68,551,169),
 rectangle(0,170,551,271),
 rectangle(0,272,551,373),
 rectangle(0,374,551,475),
 setfillstyle(SOLID_FILL,12),
 bar(552,68,639,271),
 setfillstyle(SOLID_FILL,10),
 bar(552,272,639,475),
 outtextxy(560,70,"Channel 1"),
 outtextxy(560,85,"Heart Rate"),
 outtextxy(560,274,"Channel 2"),
 outtextxy(560,289,"Heart Rate"),
```

```
}break;
case '3':
{
setfillstyle(SOLID_FILL,RED);
bar(0,0,639,67);
setcolor(WHITE);
outtextxy(30,15,"ECG Measuring Recording and Displaying by computer");
outtextxy(455,6," Stop and Replay ECG ");
outtextxy(500,21," Press 'R' ");
outtextxy(20,40,"Channel 1");
outtextxy(120,40,"Channel 2");
outtextxy(220,40,"Channel 3");
outtextxy(320,40,"Channel 4");
outtextxy(420,40,"4 Channel");
outtextxy(530,40," Stop ");
outtextxy(20,55,"Press '1' ");
outtextxy(120,55,"Press '2' ");
outtextxy(220,55,"Press '3' ");
outtextxy(320,55,"Press '4' ");
outtextxy(420,55,"Press '5' ");
outtextxy(520,55,"Press Any key ");
rectangle(0,0,450,34);
rectangle(450,0,639,34);
rectangle(0,34,100,67);
rectangle(100,34,200,67);
rectangle(200,34,300,67);
rectangle(300,34,400,67);
rectangle(400,34,500,67);
rectangle(500,34,639,67);
rectangle(0,68,551,169);
rectangle(0,170,551,271);
rectangle(0,272,551,373);
rectangle(0,374,551,475);
if(pu==0){
setfillstyle(SOLID_FILL,12);
bar(552,68,639,169);
setfillstyle(SOLID_FILL,10);
bar(552,170,639,271);
setfillstyle(SOLID_FILL,13);
bar(552,272,639,373);
setfillstyle(SOLID_FILL,14);
bar(552,374,639,475);
setcolor(1);
outtextxy(560,70,"Channel 1");
outtextxy(560,85,"Heart Rate");
outtextxy(560,172,"Channel 2");
outtextxy(560,187,"Heart Rate");
outtextxy(560,274,"Channel 3");
outtextxy(560,289,"Heart Rate");
outtextxy(560,376,"Channel 4");
outtextxy(560,391,"Heart Rate");
}
if(pu==1){
setfillstyle(SOLID_FILL,12);
bar(552,68,639,475);
setcolor(1);
outtextxy(560,70,"Channel 1");
outtextxy(560,85,"Heart Rate");
}
if(pu==2){
setfillstyle(SOLID_FILL,10);
bar(552,68,640,475);
```

```
    setcolor(1) ;
    outtextxy(560,70,"Channel 2"),
    outtextxy(560,85,"Heart Rate"),
    }
    if(pu==3){
    setfillstyle(SOLID_FILL,13),
    bar(552,68,640,475),
    setcolor(1),
    outtextxy(560,70,"Channel 3"),
    outtextxy(560,85,"Heart Rate"),
    }
    if(pu==4){
    setfillstyle(SOLID_FILL,14),
    bar(552,68,640,475),
    setcolor(1),
    outtextxy(560,70,"Channel 4"),
    outtextxy(560,85,"Heart Rate"),
    }
    } break,
    }
    while(ib<549)
    {
     setcolor(WHITE),
     line(ib+1),169-s1[ib],(ib+2),169-s1[ib+1]),
     line(ib+1),271-s2[ib],(ib+2),271-s2[ib+1]),
     line(ib+1),373-s3[ib],(ib+2),373-s3[ib+1]),
     line(ib+1),475-s4[ib],(ib+2),475-s4[ib+1]),
     ib++,
    }
   do{
  /* Disable DMA transfer first --- ---------- ------- --- */
  outp( PORT3,0),          /* (1 1) Disable A/D DMA request*/
  outp(P[ch][4],D[ch][2]),   /* (1 2) Disable DMA transfer */

  /* Define the A/D channel MUX ---------- ---- ---------- */
  outp(PORT2,mux),          /* (2) Convert channel 0 to channel 7*/

  /*Initial 8254 Timer                                    */
  outp(PORT15,0x84),        /* (3 1) 8254 Counter 2 Mode */
  outp(PORT14,(T2%256)),     /* (3 2) 8254 Counter 2 LSB */
  outp(PORT14,(int)(T2/256)),  /* (3 3) 8254 Counter 2 MSB */
  outp(PORT15,0x74),         /*(3-4) 8254 Counter 1 Mode */
  outp(PORT13,(T1 % 256)),    /*(3 5) 8254 Counter 1 LSB */
  outp(PORT13,(int)(T1/256)),  /*(3 6) 8254 Counter 1 MSB */
   /*Initial DMA controller ------ ----- ---------- ---------- */
 if (ch>3){                 /*Calculate 8 bit memory address */
  AP=BS,                    /*Page register   A19-A16 */
  AM=(int)(BA/256),              /*Base address MSB A15-A08  */
  AL=BA % 256, }                 /*Base address LSB A07-A00  */
 if (ch<4){                 /*Calculate 16 bit memory address */
 AP=BS,                    /*Page register   A23-A16 A16 don't care*/
 AM=(BS % 2)*128+(int)(BA/512),     /*Base address MSB A16-A09 */
 AL=BA % 512, }             /*Base address LSB A08-A01 */
 outp(P[ch][6],0),           /*(4-1) Clear DMA F-F */
 outp(P[ch][1],AL),           /*(4-2) DMA address A00-A07 or A01-A08*/
 outp(P[ch][1],AM),           /*(4-3) DMA address A08-A15 or A09-A16*/
 outp(P[ch][0],AP),           /*(4-4) DMA address A16-A19 or A17-A23*/
 outp(P[ch][2],0x07),       /*(5-1) DMA word W00-W07*/
 outp(P[ch][2],0x00);       /*(5-2) DMA word W08-W15*/
 outp(P[ch][5],0x47),         /*(6) Write DMA mode   */
 outp(P[ch][4],D[ch][1]),       /*(7) Enable DMA      */
```

```
/*Enable PA-MA12 DMA transfer and 8254 Timer Trigger ----------*/
outp(PORT3,0x0C);          /*(8) Enable PM-MA12 DMA */

/*Check conversion status----------*/
do
{
b=0;
b=inp(P[ch][3]);
b=(b & D[ch][3]);          /*(R-1) Read DMA status */
outp(P[ch][6],0);          /*(R-2) Clear DMA F-F   */
} while (b==0);
rmem=MK_FP(0x8000,0x0000);
switch(c1)
{
 case '1'
 {
   pu=1;
              for(q=0;q<=count;q++)
 {

 rmem=rmem+1;
 c11=*rmem;
 fprintf(outfile1,"%c11",c11);
 m1[q]= (int)(c11);
 m1[q]= ((m1[q]*100)/256);
 rmem=rmem+1;
 }
 if(pop==0) {
              plotgraph1(count,m1);
 }
 if(pop==1){
  plotgraph1(count,m2);
 }
 if(pop==2) {
              plotgraph1(count,m3);
 }
 if(pop==3) {
              plotgraph1(count,m4);
 }
 }
 break;
 case '2'
 {
  for(q=0;q<(count/2);q++)
 {
 rmem=rmem+1;
 c11=*rmem;
 fprintf(outfile1,"%c11",c11);
 m1[q]= (int)(c11);
 m1[q]= ((m1[q]*100)/256);
 rmem=rmem+2;
 c12=*rmem;
 fprintf(outfile2,"%c12",c12);
 m2[q]= (int)(c12);
 m2[q]= ((m2[q]*100)/256);
 rmem=rmem+1;
 }
 if(pop==0) {
  plotgraph2(count,m1,m2);}
 if(pop==1){
  plotgraph2(count,m3,m4);
 }
```

```
 }
 break;
 case '5':
 {
  for(q=0;q<(count/4);q++)
 {
 num=num+1;
 c11=*num;
 fprintf(pnfile1,"%c11",c11);
 m1[q]= (int)(c11);
 m1[q]= ((m1[q]*100)/256);
 num=num+2;
 c12=*num;
 fprintf(pnfile2,"%c12",c12);
 m2[q]= (int)(c12);
 m2[q]= ((m2[q]*100)/256);
 num=num+2;
 c13=*num;
 fprintf(pnfile3,"%c13",c13);
 m3[q]= (int)(c13);
 m3[q]= ((m3[q]*100)/256);
 num=num+2;
 c14=*num;
 fprintf(pnfile4,"%c14",c14);
 m4[q]= (int)(c14);
 m4[q]= ((m4[q]*100)/256);
 num=num+1;
 }
  if(pu==0)
  {
  plot(graph1,count,p1,p2,p3,p4);
  }
  else
  {
   if(pop==0) {
              plot(graph1,(count/4),p1);
                          }
   if(pop==1) {
              plot(graph1,(count/4),p2);
                          }
  if(pop==2)  {
              plot(graph1,(count/4),p3);
                          }
   if(pop==3) {
              plot(graph1,(count/4),p4);
                          }
  }
 }
 break;
 }
} while(kbhit());
 c2=getch();
 {
  if(c2== 'r')
  {
  cleargraph();
  pu=1;
  sec=pop=m=num=0;
  while(sec<330)
             {
  n1[n] = 50;
  n2[n] = 50;
```

```
s3[sa] = 50.
s4[sa] = 50.
sa++.
            }
        goto learn.
}
if(c2=='2')
{
closegraph().
pu=2.
sco=pop=su=sum=0.
while(sa<550)
            {
s1[sa] = 50.
s2[sa] = 50.
s3[sa] = 50.
s4[sa] = 50.
sa++.
            }
        goto learn.
}
if(c2=='3')
{
closegraph().
pu=3.
sco=pop=su=sum=0.
while(sa<550)
            {
s1[sa] = 50.
s2[sa] = 50.
s3[sa] = 50.
s4[sa] = 50.
sa++.
            }
        goto learn.
}
if(c2=='4')
{
closegraph().
pu=4.
sco=pop=su=sum=0.
while(sa<550)
            {
s1[sa] = 50.
s2[sa] = 50.
s3[sa] = 50.
s4[sa] = 50.
sa++.
            }
        goto learn.
}
if(c2=='5')
{
closegraph().
pu=0.
sco=su1=su2=su3=su4=sum=pop=0.
while(sa<550)
            {
s1[sa] = 50.
s2[sa] = 50;
s3[sa] = 50.
s4[sa] = 50.
```

```
  i++;
            }
          goto loop;
 }
if(c2=='r')
{
/*Disable DMA transfer first ---------------------------------*/
outp(PORT3,0);              /*(3-1) Disable A/D command  */
outp(P[ch][4],D[ch][2]);          /*(3-2) Disable DMA transfer */
 closegraph();
 clscr();
 printf("Enter: memorize that you want to show  ");
 scanf("%d",&q);
 printf("Enter: ECG channel (1 ,2 ,3 ,4 ,5 for 4 channel) ");
c3=getch();
if(c3=='1')
{ size=fsall(jrnfile1);
          size= (size - (q*60*100));
          fseek(jrnfile1,size,0);
 }
if(c3=='2')
{ size=fsall(jrnfile2);
          size= (size  (q*60*100));
          fseek(jrnfile2,size,0);
 }
if(c3=='3')
{ size=fsall(jrnfile3);
          size= (size  (q*60*100));
          fseek(jrnfile3,size,0);
}
if(c3=='4')
{ size=fsall(jrnfile4);
          size= (size - (q*60*100));
          fseek(jrnfile4,size,0);
}
if(c3=='5')
{
          size1=fsall(jrnfile1);
          size1= (size1  (q*60*100));
          fseek(jrnfile1,size1,0);
          size2=fsall(jrnfile2);
          size2= (size2  (q*60*100));
          fseek(jrnfile2,size2,0);
          size3=fsall(jrnfile3);
          size3= (size3  (q*60*100));
          fseek(jrnfile3,size3,0);
          size4=fsall(jrnfile4);
          size4= (size4  (q*60*100));
          fseek(jrnfile4,size4,0);
}
pop=0;
if (opengraph()==TRUE)
{
setfillstyle(SOLID_FILL,RED);
bar(0,0,639,67);
setcolor(WHITE);
outtextxy(120,15,"ECG Measuring Recording and Displaying by computer");
rectangle(0,0,639,67);
rectangle(0,68,551,169);
rectangle(0,170,551,271);
rectangle(0,272,551,373);
rectangle(0,374,551,475);
```

```
if(c3=='1'){
setfillstyle(SOLID_FILL,12),
bar(552,68,640,475),
setcolor(1),,
outtextxy(560,70,"Channel 1"),
outtextxy(560,85,"Heart Rate"),
pu=1,
}
if(c3=='2'){
setfillstyle(SOLID_FILL,10),
bar(552,68,640,475),
setcolor(1),'
outtextxy(560,70,"Channel 2"),
outtextxy(560,85,"Heart Rate"),
pu=2,
}
if(c3=='3'){
setfillstyle(SOLID_FILL,13),
bar(552,68,640,475),
setcolor(1),
outtextxy(560,70,"Channel 3"),
outtextxy(560,85,"Heart Rate"),
pu=3,
}
if(c3=='4'){
setfillstyle(SOLID_FILL,14),
bar(552,68,640,475),
setcolor(1),
outtextxy(560,70,"Channel 4"),
outtextxy(560,85,"Heart Rate"),
pu=4,
}
if(c3=='5')
{
setfillstyle(SOLID_FILL,12),
bar(552,68,639,169),
setfillstyle(SOLID_FILL,10),
bar(552,170,639,271),
setfillstyle(SOLID_FILL,13),
bar(552,272,639,373),
setfillstyle(SOLID_FILL,14),
bar(552,374,639,475),
setcolor(1),
outtextxy(560,70,"Channel 1"),
outtextxy(560,85,"Heart Rate"),
outtextxy(560,172,"Channel 2"),
outtextxy(560,187,"Heart Rate"),
outtextxy(560,274,"Channel 3"),
outtextxy(560,289,"Heart Rate"),
outtextxy(560,376,"Channel 4"),
outtextxy(560,391,"Heart Rate"),
pu=0,
}
if(c3=='1')
{
        do
        {
         for(i=0,i<1,i++)
          {
           fread(&c11,1,1,infile1),
           m1[i]=(int)(c11),
           m1[i]=m1[i]*100/256,
```

```
                }
                if(pop==0) {
                 plotgraph1(1,s1),
                 delay(2),
                                }
 if(pop==1) {
                 plotgraph1(1,s2),
                 delay(2),
                                }
 if(pop==2) {
                 plotgraph1(1,s3),
                 delay(2),
                                }
 if(pop==3) {
                 plotgraph1(1,s4),
                 delay(2),
                                }
 }while (!feof(infile1)),
 }
 if(c3=='2')
 {
                do
                {
                 for(i=0,s<1,s++)
                  {
                   fread(&c11,1,1,infile2),
                   s1[i]=(int)(c11),
                   s1[i]=s1[i]*100/256,
                  }
                if(pop==0) {
                 plotgraph1(1,s1),
                 delay(2),
                                }
 if(pop==1) {
                 plotgraph1(1,s2),
                 delay(2),
                                }
 if(pop==2) {
                 plotgraph1(1,s3),
                 delay(2),
                                }
 if(pop==3) {
                 plotgraph1(1,s4),
                 delay(2),
                                }
 }while (!feof(infile2))
 }
 if(c3=='3')
 {
                do
                {
                 for(i=0,s<1,s++)
                  {
                   fread(&c11,1,1,infile3),
                   s1[i]=(int)(c11),
                   s1[i]=s1[i]*100/256,
                  }
                if(pop==0) {
                 plotgraph1(1,s1),
                 delay(2),
                                }
 if(pop==1) {
```

```
                plotgraph1(1,s2).
                delay(2).
                                }
    if(pop==2) {
                plotgraph1(1,s3).
                delay(2).
                                }
    if(pop==3) {
                plotgraph1(1,s4).
                delay(2).
                                }
    }while (!feof(infile3)).
    }
    if(c3=='4')
    {
                do
                {
                for(i=0;i<1;i++)
                 {
                  fread(&c11,1,1,infile4).
                  m1[i]=(int)(c11).
                  m1[i]=m1[i]*100/256.
                 }
                if(pop==0) {
                plotgraph1(1,s1).
                delay(2).
                                }
    if(pop==1) {
                plotgraph1(1,s2).
                delay(2).
                                }
    if(pop==2) {
                plotgraph1(1,s3).
                delay(2).
                                }
    if(pop==3) {
                plotgraph1(1,s4).
                delay(2).
                                }
    }while (!feof(infile4)).
    }
    if(c3==5)
    { do
                {
                fread(&c11,1,1,infile1).
                m1[i]=(int)(c11).
                m1[i]=m1[i]*100/256.
                fread(&c12,1,1,infile2).
                m2[i]=(int)(c12).
                m2[i]=m2[i]*100/256.
                fread(&c13,1,1,infile3).
                m3[i]=(int)(c13).
                m3[i]=m3[i]*100/256.
                fread(&c14,1,1,infile4).
                m4[i]=(int)(c14).
                m4[i]=m4[i]*100/256.
                plotgraph3(count,s1,s2,s3,s4).
                }while(!feof(infile1)).
    }
    }
    fclose(infile1).
    fclose(infile2).
```

```
    fclose(stdin3);
    fclose(stdin4);
    printf("time to simulate = ");
    delay(1000);
   }
  }
  }
   outp(PORT3,0);              /*(3-1) Disable A/D command  */
   outp(Pich[4],D[ch][2]);          /*(3-2) Disable DMA transfer */
   closegraph();
   ics=0;
   pu=0;
}
BOOL opengraph(void)
{
 int gdriver = DETECT,gmode,errorcode;
 initgraph(&gdriver, &gmode, "");        /* initialize graphics mode */
 errorcode = graphresult();          /* read result of initialization */
 if (errorcode != grOk)                  /* an error occurred */
              return FALSE;
 return TRUE;
}
void plotgraph1(int account,unsigned int *s1)
{
  int a=0,color;
  int i,x,y,ans,j;
  while(account)
  {
   setcolor(BLACK);
   line((ics+1),169+(102*pop)-s1[ics],(ics+2),169+(102*pop)-s1[ics+1]);
              if(pu==1){
              s1[ics]=as1[a];
              color=12;
              }
              if(pu==2){
              s1[ics]=as2[a];
              color=10;
              }
              if(pu==3){
              s1[ics]=as3[a];
              color=13;
              }
              if(pu==4){
              s1[ics]=as4[a];
              color=14;
              }
   a++;
  if (ics==0)
   {
    putpixel(ics+1,169+(102*pop)-s1[ics],color);
   }
  else
   {
    setcolor(color);
    line(ics,169+(102*pop)-s1[ics-1],ics+1,169+(102*pop)-s1[ics]);
   }
   ics++;
  if(ics==549)
  {
   ans++;
   ics=0;
   pop++;
```

```
    if(pop==4)
    { pop=0. }
    b=0.
    z=0;
    x=70;
    cnt=0;
    for (j=0;j<330;j++)
    {
              if(b1[j]>z && z==0)
              {
                b=1.
              }
              if(b1[j]<z && z==1)
              {
                b=0.
                z=0.
              }
              if(b==1 && z==0)
              {
               cnt++.
               z=1;
              }
    }
    re=(re+cnt) .
    setlinestyle(DEFAULT_FONT,HORIZ_DIR,3).
    setcolor(color).
    outtextxy(575,119,hr1);
    sprintf(hr1,"%d",hsrt1).
    sprintf(hr2,"%d",hsrt2).
    sprintf(hr3,"%d",hsrt3).
    sprintf(hr4,"%d",hsrt4).
    hsrt1= (int)(re*11)/cnn;
    setcolor(1);
    sprintf(hr1,"%d",hsrt1);
    outtextxy(575,119,hr1).
    if(cnn==11)
    {
     cnn=0.
     re=0.
    }
  }
 }
}
void plotgraph2(int count,unsigned int *s1,unsigned int *s2)
{
  int a=0.
  while(a<(count/2))
  {
   setcolor(BLACK).
   line((xx+1),160+(102*pop)-s1[xx],(xx+2),169+(102*pop)-s1[xx+1]).
   line((xx+1),373+(102*pop)-s2[xx],(xx+2),373+(102*pop)-s2[xx+1]).
   s1[xx]=s1[a].
   s2[xx]=s2[a].
   a++.
  if (xx==0)
   {
   putpixel(xx+1,169+(102*pop)-s1[xx],12);
   putpixel(xx+1,373+(102*pop)-s2[xx],10);
   }
  else
   {
   setcolor(12);
```

```
    line(xco,169+(102*pop)-s1[xco-1],xco+1,116+(102*pop)-s1[xco]),
    setcolor(10);
    line(xco,373+(102*pop)-s2[xco-1],xco+1,348+(102*pop)-s2[xco]);
    }
    xco++;
  if(xco==549)
  {
    xco=0,
    pop++;
    if(pop==2)
    { pop=0,
    }
  }
  }
}
void plotgraph3(int count,unsigned int *s1,unsigned int *s2,unsigned int *s3,unsigned int *s4)
{
  int n=0,i=0,
  int t1,t2,t3,t4,z1,z2,z3,z4,x,state1,state2,state3,state4,
  while(n<(count/4))
  {
    setcolor(BLACK),
    line((xco+1),169-s1[xco],(xco+2),169-s1[xco+1]),
    line((xco+1),271-s2[xco],(xco+2),271-s2[xco+1]),
    line((xco+1),373-s3[xco],(xco+2),375-s3[xco+1]),
    line((xco+1),475-s4[xco],(xco+2),475-s4[xco+1]),
    s1[xco]=s1[n],
    s2[xco]=s2[n],
    s3[xco]=s3[n],
    s4[xco]=s4[n],
    n++,
  if (xco==0)
    {
    putpixel(xco+1,169-s1[xco],12),
    putpixel(xco+1,271-s2[xco],10),
    putpixel(xco+1,373-s3[xco],13),
    putpixel(xco+1,475-s4[xco],14),
    }
  else
    {
    setcolor(12),
    line(xco,169-s1[xco-1],xco+1,169-s1[xco]),
    setcolor(10),
    line(xco,271-s2[xco-1],xco+1,271-s2[xco]),
    setcolor(13),
    line(xco,373-s3[xco-1],xco+1,373-s3[xco]),
    setcolor(14),
    line(xco,475-s4[xco-1],xco+1,475-s4[xco]),
    }
    xco++;
  }
  if(xco==549)
  {
    xco=0;
    t1=t2=t3=t4=0;
    z1=z2=z3=z4=0;
    x=70;
    cnt++;
    state1=state2=state3=state4=0,
    for (j=0;j<600;j++)
    {
                    if(t1>x && z1==0)
```

```
        {
          t1=1;
        }
        if(b1[j]<x && z1==1)
        {
         t1=0;
         z1=0;
        }
        if(t1==1 && z1==0)
        {
         cnto1++;
         z1=1;
        }
        if(b2[j]>x && z2==0)
        {
          t2=1;
        }
        if(b2[j]<x && z2==1)
        {
         t2=0;
         z2=0;
        }
        if(t2==1 && z2==0)
        {
         cnto2++;
         z2=1;
        }
        if(b3[j]>x && z3==0)
        {
          t3=1;
        }
        if(b3[j]<x && z3==1)
        {
         t3=0;
         z3=0;
        }
        if(t3==1 && z3==0)
        {
         cnto3++;
         z3=1;
        }
        if(b4[j]>x && z4==0)
        {
          t4=1;
        }
        if(b4[j]<x && z4==1)
        {
         t4=0;
         z4=0;
        }
        if(t4==1 && z4==0)
        {
         cnto4++;
         z4=1;
        }
 }
 zn1=(zn1+cnto1);
 zn2=(zn2+cnto2);
 zn3=(zn3+cnto3);
 zn4=(zn4+cnto4);
 settextstyle(DEFAULT_FONT,HORIZ_DIR,3);
 sprintf(zn1,"%d",heart1);
```

```
    sprintf(hr2,"%d",hsat2);
    sprintf(hr3,"%d",hsat3);
    sprintf(hr4,"%d",hsat4);
    setcolor(12);
    outtextxy(375,119,hr1);
    setcolor(10);
    outtextxy(375,221,hr2);
    setcolor(13);
    outtextxy(375,325,hr3);
    setcolor(14);
    outtextxy(375,425,hr4);
    hsat1= (int)(ra1*11)/num;
    hsat2= (int)(ra2*11)/num;
    hsat3= (int)(ra3*11)/num;
    hsat4= (int)(ra4*11)/num;
    setcolor(1);
    sprintf(hr1,"%d",hsat1);
    sprintf(hr2,"%d",hsat2);
    sprintf(hr3,"%d",hsat3);
    sprintf(hr4,"%d",hsat4);
    outtextxy(375,119,hr1);
    outtextxy(375,221,hr2);
    outtextxy(375,325,hr3);
    outtextxy(375,425,hr4);
    if(num==11)
    {
     num=0;
     ra1=ra2=ra3=ra4;
    }
  }
}
```

สรุปและวิจารณ์ผลการทดลอง

ผลการทดลองที่ทำการทดลองมาเป็นที่น่าพอใจในระดับหนึ่ง คือเราสามารถนำสัญญาณที่เป็น analog เข้ามาทำการ วิเคราะห์หรือประมวลผลของสัญญาณแบบ digital โดยใช้ computer เป็นตัวประมวลผล แสดงว่าการ Interface การ์ด PA-MA12(-H) กับ computer เป็นไปด้วยดีและสามารถเขียนโปรแกรม support การ transfer ข้อมูลในแบบที่เราเลือกได้ (โครงการนี้เราใช้การ transfer ข้อมูลแบบ DMA) แต่เนื่องจากการ ที่เรา จะ Interface การ์ด และทำการ transfer ข้อมูลนั้นเราจะต้องเสียเวลาส่วนหนึ่งในการทำงานในส่วนของการ เก็บข้อมูลลงใน file และ เสียเวลาในการ plot ค่าข้อมูลที่นำมาแสดงผลบนจอภาพ ดังนั้น ภาพที่ปรากฏบนจอภาพ เราไม่สามารถทำให้มันเป็น Real-time ได้จริง ๆ ดังนั้นถ้าเราใช้ค่าการ Sampling (Sampling Rate)ที่ ต่ำ ๆ และสัญญาณที่เข้ามาในการ์ดมีค่าสูง ๆ จะเกิด error ขึ้นที่ข้อมูลที่ทำการ Sampling จะเกิดการผิดพลาด ดังนั้นเราจะต้องทำการปรับปรุงการทำงานของโปรแกรม เพื่อให้มีความรวดเร็วมากขึ้นโดยให้มีความรวดเร็วที่สุดเท่าที่จะทำได้ โดยคณะผู้จัดทำจะเสนอแนวทางการแก้ไขดังต่อไปนี้

1. การปรับปรุงทางด้าน Hardware การปรับปรุงให้มีการ set ค่า configuration ของการ์ดให้ถูกต้องและ เลือกใช้การ transfer ที่เหมาะสม เช่น การเลือก DMA channel การเลือกช่องทางการ conversion ที่ถูกต้อง การเลือก IRQ ที่ว่าง การ set address ของ การ์ด การเลือกใช้การ transfer ข้อมูลแบบ word หรือ byte ฯลฯ เราจะต้อง set ค่าต่าง ๆ นี้อย่างรอบคอบ และเลือกวิธีที่ถูกต้องที่สุด

2. การปรับปรุงทางด้าน Solfware ทำการปรับปรุงทางด้านการทำการ Interface ที่มี Routine ที่รวดเร็วไม่ยุ่งยาก ซับซ้อน รวมถึงการปรับปรุง Routine การ plot แสดงผลบนจอภาพที่มีความรวดเร็ว ซึ่งจะทำให้การแสดงผลบนจอภาพมีความใกล้เคียงความจริงมากที่สุดเท่าที่จะทำได้ เพราะโปรแกรม 1 คำสั่งในภาษา C นั้นใช้เวลาในการทำงานมากถ้าเราสามารถใช้การเขียนโปรแกรมการ plot ที่ไม่ต้องเรียกใช้การทำงานของ function ในภาษา C กล่าวคือถ้าสามารถเขียนโปรแกรม การ plot ที่เรียกใช้ function ในภาษา C น้อยที่สุดแล้วโปรแกรมการ plot ก็จะมีการตอบสนองต่อการ plot ที่รวดเร็วขึ้น เช่นถ้าเราสามารถ สร้างการ plot ที่ใช้ภาษา Assembly คือสามารถเขียนโปรแกรมโดยที่ไม่ต้องใช้การลบภาพ บนจอภาพ แต่ใช้การสลับ การแสดงผลเป็นแบบ Page สลับกันขึ้นมาแสดงผล แล้วใช้การเขียนข้อมูลทับ Page ที่ไม่ได้นำขึ้นมาแสดง เราก็จะสามารถสร้าง Routine ที่รวดเร็วกว่าที่ใช้การ ลบ แล้ววาดใหม่ที่กระทำอยู่ในโครงการนี้

กิตติกรรมประกาศ

ปริญญานิพนธ์เล่มนี้สำเร็จได้ด้วยดี เนื่องด้วยได้รับความช่วยเหลือจากผู้มีพระคุณต่าง ๆ คณะผู้จัดทำจึงขอแสดงความขอบคุณทุกท่าน อันได้แก่ อาจารย์พิชัย ดุศิริวานิชกร อาจารย์ที่ปรึกษา ,อาจารย์ท่านอื่น ๆ ที่ให้คำแนะนำ , รุ่นพี่ทุกท่านที่ให้คำแนะนำ , เพื่อนซึ่งให้ความช่วยเหลือและกำลังใจ ปริญญานิพนธ์ชิ้นนี้ได้รับความช่วยเหลือจากท่านเหล่านี้มาด้วยดี.

บรรณานุกรม

1. การเขียนชุดคำสั่งภาษาซี , รองศาสตราจารย์ มัณฑนา ปราการสมุทร , บริษัท ควงกมลสมัย จำกัด , พิมพ์ครั้งที่สาม มิถุนายน 2534

2. หนังสือเรียนอิเลคทรอนิคส์ทางการแพทย์ (Bio-medical Electronic Engineering) , สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ฯ ลาดกระบัง

3. PA-MA12(-H) DMA Transferred Multifunction Card Instruction Manual , Acqutek Corporation , May 1 ,1992

4. การเขียนโปรแกรมใช้งาน EGA/VGA , วีรวัฒน์ ตัณฑวังกูร สราวุธ เอื้องอุทัยวัฒน์

5. PC SYSTEM PROGRAMMING , Michael Tisher

ภาคผนวก

A/D Register and Callbration

A/D Registers and Calibration

This chapter covers the information for A/D registers, A/D signal connection and the procedures for calibration of the PA-MA12(-H). Included are :

- Registers information
- Signal connections
- Calibration

Detailed information for the A/D operation theory and programming procedures can be found in chapter 3 and chapter 4.

I/D Registers List

This section covers the detailed register information. There are two types of registers invloved in the DMA operation :

- The PA-MA12(-H) control register
- The DMA controllers

The DMA controllers are accessed by 34 I/O ports. They are provided by the personal computer. The 16 bit DMA controller and 16 bit DMA page registers are not available for the PC/XT (8088) compatible computer.

- 8 bit DMA controller (15 I/O ports)
- 16 bit DMA controller (12 I/O ports)
- 8 bit DMA page registers (4 I/O ports)
- 16 bit DMA page registers (3 I/O ports)

PA-MA12(-H) registers

Port	Default	Direction	Function
Base+0	02E0H	I/O read	A/D LSB data
		I/O write	S/W start A/D conversion
Base+1	02E1H	I/O read	A/D MSB data
		I/O write	none
Base+2	02E2H	I/O read	A/D channel select
		I/O write	
Base+3	02E3H	I/O read	A/D status
		I/O write	A/D command
Base+12	02ECH	I/O read	8254 counter 0 data
		I/O write	
Base+13	02EDH	I/O read	8254 counter 1 data
		I/O write	
Base+14	02EEH	I/O read	8254 counter 2 data
		I/O write	
Base+15	02EFH	I/O read	8254 read-back status
		I/O write	8254 command

System DMA registers

There are two different controllers on the system mother board - The DMA controller (8237) and the DMA page controller. The port addresses are listed below :

8 bit transferred DMA registers

Port	DMA CH	Direction	Function
0000H	0	I/O R/W	Base address register
0001H	0	I/O R/W	Base word count register
0002H	1	I/O R/W	Base address register
0003H	1	I/O R/W	Base word count register
0004H	2	I/O R/W	Base address register
0005H	2	I/O R/W	Base word count register
0006H	3	I/O R/W	Base address register
0007H	3	I/O R/W	Base word count register
0008H	0-3	I/O Read	DMA status
0008H	0-3	I/O Write	DMA command
0009H	0-3	I/O Write	Software DREQ request
000AH	0-3	I/O Write	Write single DMA mask reg
000BH	0-3	I/O Write	DMA mode register
000CH	0-3	I/O Write	DMA reset pointer flip-flop
000FH	0-3	I/O Write	Write all DMA mask register
0087H	0	I/O Write	DMA channel 0 page register
0083H	1	I/O Write	DMA channel 1 page register
0081H	2	I/O Write	DMA channel 2 page register
0082H	3	I/O Write	DMA channel 3 page register

Note :

The DMA channel 2 is reserved for the floppy disk controller. Also, the DMA channel 0 is not available for 8088 CPU computer (PC/XT).

16 bit transferred DMA registers

Port	DMA CH	Direction	Function
00C4H	5	I/O R/W	Base address register
00C6H	5	I/O R/W	Base word count register
00C8H	6	I/O R/W	Base address register
00CAH	6	I/O R/W	Base word count register
00CCH	7	I/O R/W	Base address register
00CEH	7	I/O R/W	Base word count register
00D0H	5-7	I/O Read	DMA status
00D0H	5-7	I/O Write	DMA command
00D2H	5-7	I/O Write	Software DREQ request
00D4H	5-7	I/O Write	Write single DMA mask reg
00D6H	5-7	I/O Write	DMA mode register
00D8H	5-7	I/O Write	DMA reset pointer flip-flop
00DEH	5-7	I/O Write	Write all DMA mask register
008BH	5	I/O Write	DMA channel 5 page register
0089H	6	I/O Write	DMA channel 6 page register
008AH	7	I/O Write	DMA channel 7 page register

Note :

The DMA channel 5, 6 and 7 are not available for 8088 CPU computer (PC/XT).

egister Information

Base + 0 : S/W start conversion register

Base + 0		Write	A/D Start Conversion (S/W Trig)				
7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x
Bit	Symbol		Function				
7-0	x		don't care				

Base + 0 and Base + 1 : A/D data registers

Base + 0		Read	A/D LSB Data Register				
7	6	5	4	3	2	1	0
AD03	AD02	AD01	AD00	CH3	CH2	CH1	CH0
Bit	Symbol		Function				
7 - 4	AD03-AD00		A/D LSB data				
3 - 0	CH3-CH0		A/D Channel number				

Base + 1		Read	A/D MSB Data Register				
7	6	5	4	3	2	1	0
AD11	AD10	AD09	AD08	AD07	AD06	AD05	AD04
Bit	Symbol		Function				
7 - 0	AD11-AD04		A/D MSB data				

Port	Base + 1								Base + 0			
Bit	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4
AD	11	10	9	8	7	6	5	4	3	2	1	0

Base+2 : Channel register

Base+2		R/W	A/D Channel Select				
7	6	5	4	3	2	1	0
CH3	CH2	CH1	CH0	CL3	CL2	CL1	CL0

Bit	Symbol	Function
7 - 4	CH3-CH0	A/D channel MUX high limit CH3 CH2 CH1 CH0 Channel No 0 0 0 0 Channel 0 0 0 0 1 Channel 1 0 0 1 0 Channel 2 0 0 1 1 Channel 3 0 1 0 0 Channel 4 0 1 0 1 Channel 5 0 1 1 0 Channel 6 0 1 1 1 Channel 7 1 0 0 0 Channel 8 1 0 0 1 Channel 9 1 0 1 0 Channel 10 1 0 1 1 Channel 11 1 1 0 0 Channel 12 1 1 0 1 Channel 13 1 1 1 0 Channel 14 1 1 1 1 Channel 15
3 - 0	CL3-CL0	A/D Channel MUX low limit CL3 CL2 CL1 CL0 Channel No 0 0 0 0 Channel 0 0 0 0 1 Channel 1 0 0 1 0 Channel 2 0 0 1 1 Channel 3 0 1 x x Channel 4 to 7 1 0 x x Channel 8 to 11 1 1 x x Channel 12 to 15

Base+3 : Command register

Base+3		Write	A/D Command				
7	6	5	4	3	2	1	0
x	x	x	x	DMAE	S1	S0	INTEN

Bit	Symbol	Function
7-4	x	don't care
3	DMAE	A/D data DMA transfer enable 0 DMA transfer disable 1 DMA transfer enable
2 - 1	S1-S0	A/D conversion trig source S1 S0 Conversion singal 0 0 S/W trig 0 1 External clock trig 1 0 8254 clock trig 1 1 8254 clock trig (Paced by external clock)
0	INTEN	Interrupt enable 0 Interrupt disable 1 Interrupt enable

Base+3 : Status register

Base+3		Read	A/D Channel Select				
7	6	5	4	3	2	1	0
EOC	U/B	S/D	ID	DMAE	S1	S0	INTEN
Bit	Symbol		Function				
7	EOC		End of A/D conversion 1 A/D conversion is processing 0 A/D convsrsion is completed				
6	U/B		A/D Input voltage mode 0 A/D Input is bipolar mode 1 A/D Input is unipolar mode				
5	S/D		A/D Input type 0 Differential Input 1 Single-ended Input				
4	ID		1 Always 1				
3	DMAE		A/D data DMA transfer enable 0 DMA transfer disable 1 DMA transfer enable				
2 - 1	S1-S0		A/D conversion control S1 S0 Conversion singal 0 0 S/W trig 0 1 External clock trig 1 0 8254 timer trig 1 1 8254 timer trig (Paced by external clock)				
0	INTEN		Interrupt enable 0 Interrupt disable 1 Interrupt enable				

8254 timer/counter registers

Please refer chapter 3 for programming information.

8 bit DMA base and current address register

00, 02, 04, 06		R/W	8 bit DMA base and current address				
7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0
A15	A14	A13	A12	A11	A10	A9	A8
Bit	Symbol		Function				
7-0	A7-A0 A15-A8		Base and current address Each DMA channel has a 16-bit current address register. This register holds the value of the address used during the DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the addresses are stored in the curent address register during the transfer. This register is written or read by the CPU in successive 8-bit bytes. A clear flip-flop register command (write 00H to I/O port 0CH) can be used to initlalizes the flip-flop to a known state so that subsequent accesses to register contents by the CPU will address lower and upper bytes in the correct sequence. Channel 0 : 00H Channel 1 : 02H Channel 2 : 04H Channel 3 : 06H				

16 bit DMA base and current address register

C0,C4,C8,CC		R/W	16bit DMA base and current address				
7	6	5	4	3	2	1	0
A8	A7	A6	A5	A4	A3	A2	A1
A16	A15	A14	A13	A12	A11	A10	A9
Bit	Symbol		Function				
7-0	A8-A1 A16-A9		Base and current address Each DMA channel has a 16-bit current address register. This register holds the value of the address used during the DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the addresses are stored in the curent address register during the transfer. This register is written or read by the CPU in successive 8-bit bytes. A clear flip-flop register command (write 00H to I/O port D8H) can be used to initializes the flip-flop to a known state so that subsequent accesses to register contents by the CPU will address lower and upper bytes in the correct sequence. Channel 4 : C0H Channel 5 : C4H Channel 6 : C8H Channel 7 : CCH				

8 bit DMA base and current word register

01,03,05,07		R/W	8 bit DMA base and current word				
7	6	5	4	3	2	1	0
W7	W6	W5	W4	W3	W2	W1	W0
W15	W14	W13	W12	W11	W10	W9	W8
Bit	Symbol		Function				
7-0	W7-W0 W15-W8		Base and current word Each DMA channel has a 16-bit current word register. This register determines the number of transfer to be performed. The word counter is automatically decremented after each transfer and the intermediate values of the address are stored in the current address register during the transfer. This register is written or read by the CPU in successive 8-bit bytes. A clear flip-flop register command (write 00H to I/O port 0CH) can be used to initializes the flip-flop to a known state so that subsequent accesses to register contents by the CPU will address lower and upper bytes in the correct sequence. Channel 0 : 01H Channel 1 : 03H Channel 2 : 05H Channel 3 : 06H				

16 bit DMA base and current word register

C2,C6,CA,CE		R/W	16 bit DMA base and current word				
7	6	5	4	3	2	1	0
W7	W6	W5	W4	W3	W2	W1	W0
W15	W14	W13	W12	W11	W10	W9	W8
Bit	Symbol		Function				
7-0	W7-W0 W15-W8		Base and current word Each DMA channel has a 16-bit current word register. This register determines the number of transfer to be performed. The word counter is automatically decremented after each transfer and the intermediate values of the address are stored in the current address register during the transfer. This register is written or read by the CPU in successive 8-bit bytes. A clear flip-flop register command (write 00H to I/O port D8H) can be used to initializes the flip-flop to a known state so that subsequent accesses to register contents by the CPU will address lower and upper bytes in the correct sequence. Channel 4 : C2H Channel 5 : C6H Channel 6 : CAH Channel 7 : CEH				

8 bit DMA page register

80H-87H		Write	8 bit DMA page register				
7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16
Bit	Symbol		Function				
7-4	0		should be 0				
3-0	A19-A16		The 8 bit DMA controller only provides 16 bit address buses. Therefore only 64KB can be addressed. The DMA page register determines the address bus A16-A19 to access fully 1M byte memory. Channel 0 : 87H Channel 1: 83H Channel 2 : 81H Channel 3 : 82H				

16 bit DMA page register

89H - 8BH		Write	16 bit DMA page register				
7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	x
Bit	Symbol		Function				
7-1	A23-A17		The 16 bit DMA controller only provides 16 bit address buses. Therefore only 128KB can be addressed. The DMA page register determines the address bus A17-A23 to access fully 16M byte memory. Channel 5 : 8BH Channel 6 : 89H Channel 7 : 8AH				
x	0		don't care				

8/16 bit DMA status register

08H / D0H		Read	8/16 bit DMA Status				
7	6	5	4	3	2	1	0
DR3	DR2	DR1	DR0	DC3	DC2	DC1	DC0
Bit	Symbol		Function				
7-4	DR3-DR0		DMA request 1 DMA channel has requested 0 DMA channel has not requested Symbol 8 bit DMA 16 bit DMA DR3 Channel 3 Channel 7 DR2 Channel 2 Channel 6 DR1 Channel 1 Channel 5 DR0 Channel 0 Channel 4				
3-0	DC3-DC0		DMA completed 1 DMA channel process completed 0 DMA process does not complete Symbol 8 bit DMA 16 bit DMA DC3 Channel 3 Channel 7 DC2 Channel 2 Channel 6 DC1 Channel 1 Channel 5 DC0 Channel 0 Channel 4				
08H : 8 bit DMA D0H : 16 bit DMA							

8/16 Bit DMA command register

08H / D0H		Write	8/16 bit DMA command register				
7	6	5	4	3	2	1	0
MM	AHE	CEN	TIM	PRI	LWS	DRQ	DAK
Bit	Symbol		Function				
7	MM		Memory to memory transfer 1 Memory to memory enable 0 Memory ro memory disable				
6	AHE		DMA channel 0 hold 1 CH 0/4 address hold enable 0 CH 0/4 address hold disable				
5	CEN		DMA controller enable/disable 1 DMA controller enable 0 DMA controller disable				
4	TIM		DMA timing 1 Compress timing 0 Normal timing				
3	PRI		DMA priority 1 Rotating priority 0 Fixed priority				
2	LWS		DMA write selection 1 Extended write selection 0 Late write selection				
1	DRQ		DRQ active voltage 1 DRQn sense active low 0 DRQn sense active high				
0	DAK		DACK active voltage 1 DACKn sense active high 0 DACKn sense active low				

8/16 bit DMA software request register

09H / D2H		Write	8/16 bit DMA software request reg.				
7	6	5	4	3	2	1	0
x	x	x	x	x	RQ	C1	C0
Bit	Symbol		Function				
7-3	x		don't care				
2	RQ		Soft request command 1 Set request bit 0 Reset request bit				
1-0	C1-C0		C1 C0 Channel select (8 bit /16 bit) 1 1 Select channel 3 / 7 1 0 Select channel 2 / 6 0 1 Select channel 1 / 5 0 0 Select channel 0 / 4				

8/16 bit DMA single mask register

00AH / D4H		Write	8/16 bit DMA single mask register				
7	6	5	4	3	2	1	0
x	x	x	x	x	MS	C1	C0
Bit	Symbol		Function				
7-3	x		don't care				
2	MS		Mask command 1 Set mask bit 0 Reset mask bit				
1-0	C1-C0		C1 C0 Channel select (8 bit /16 bit) 1 1 Select channel 3 / 7 1 0 Select channel 2 / 6 0 1 Select channel 1 / 5 0 0 Select channel 0 / 4				

8/16 bit DMA mode register

0BH / D6H		Write	8/16 bit DMA mode register				
7	6	5	4	3	2	1	0
M1	M0	ADRD	AINIT	T1	T0	C1	C0
Bit	Symbol		Function				
7-6	M1 - M0		DMA operation mode M1 M0 operation mode 1 1 CASCADE mode 1 0 BLOCK mode 0 1 SINGLE mode 0 0 DEMAND mode				
5	ADRD		DMA address selection 1 Address decrement selected 0 Address increment selected				
4	AINIT		Auto-Initialization 1 Auto-Initialization enable 0 Auto-Initialization disable				
3-2	T1 - T0		DMA transfer mode T1 T0 Mode 1 0 Read transfer 0 1 Write transfer 0 0 Verify				
1-0	C1 - C0		DMA channel select C1 C0 Channel select (8 bit /16 bit) 1 1 Select channel 3 / 7 1 0 Select channel 2 / 6 0 1 Select channel 1 / 5 0 0 Select channel 0 / 4				

Single-ended input

Single ended inputs are inputs that are referenced to ground. The digital value that is obtained represents the difference between the input voltage and ground.

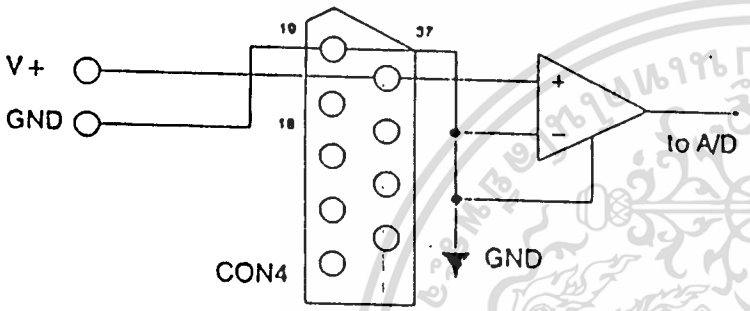


Figure 5-2 A/D Single Ended Input

Differential Input

Three leads differentilal inputs are the standard differential inputs. This configuration consists of two input leads and a ground reference. The digital value that is obtained represents the difference between the two input leads with respect to ground.

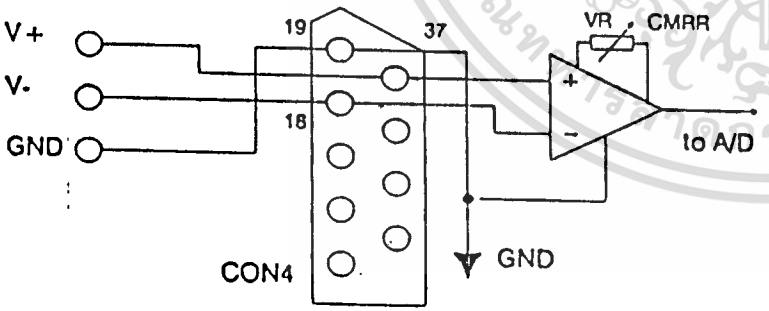


Figure 5-3 A/D Differential I/P

2 leads differential input

Two leads differential input is basically tying one of the differential inputs to ground. The result is a single ended input. The reason for this option is for a user wanting both differential and single ended inputs. The user simply sets the card jumper for differential inputs, then grounds one of the two differential leads for the inputs he wishes to be single ended.

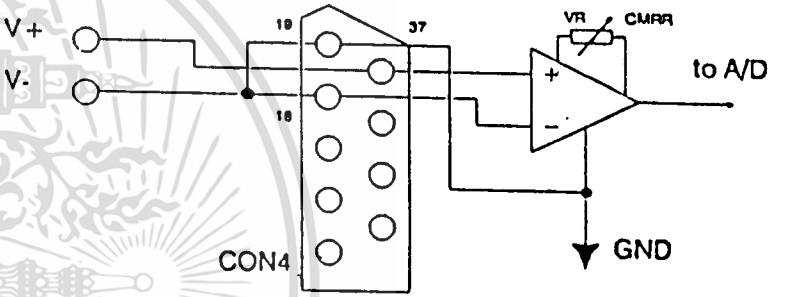


Figure 5-4 A/D 2 Leads Differential I/P

Calibration

Before using the PA-MA12(-H) for a specific A/D task, a user should calibrate the inputs. Calibration enables the highest degree of accuracy this card can provide.

Frequency of calibration

For normal use and environment, calibration every 6 months is recommended. If used in harsh or frequently changing environments, a user is advised to calibrate every 2 to 3 months. The

8/16 bit DMA reset pointer flip-flop register

0CH / D8H		Write	8/16 bit DMA reset pointer flip-flop				
7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x
Bit	Symbol		Function				
7-0	x		The base address and current word are written or read by the CPU in successive 8-bit bytes. Clear the flip-flop register (write 00H to I/O port 0CH or D8H) can be used to initialize the flip-flop to a known state so that subsequent accesses to register contents by the CPU will address lower and upper bytes in the correct sequence.				

8/16 bit DMA all mask register

0FH / DEH		Write	8/16 bit DMA all mask register				
7	6	5	4	3	2	1	0
x	x	x	x	M3	M2	M1	M0
Bit	Symbol		Function				
7-4	x		don't care				
3	M3		1 Set DMA channel 3/7 mask 0 Reset DMA channel 3/7 mask				
2	M2		1 Set DMA channel 2/6 mask 0 Reset DMA channel 2/6 mask				
1	M1		1 Set DMA channel 1/5 mask 0 Reset DMA channel 1/5 mask				
0	M0		1 Set DMA channel 0/4 mask 0 Reset DMA channel 0/4 mask				

Signal Connection

Connecter diagram

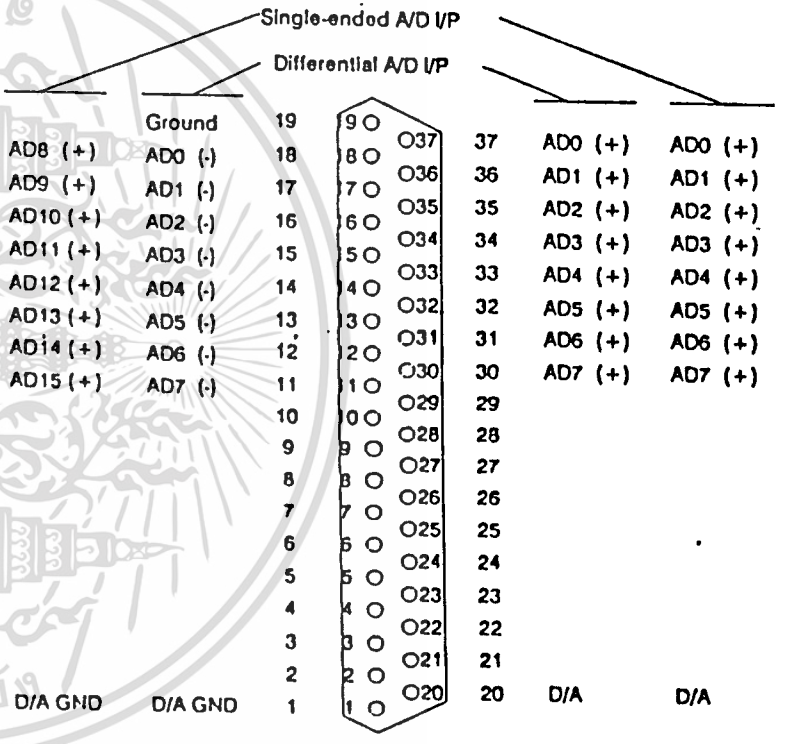


Figure 5-1 A/D Signals Connecter

Single-ended input

Single ended inputs are inputs that are referenced to ground. The digital value that is obtained represents the difference between the input voltage and ground.

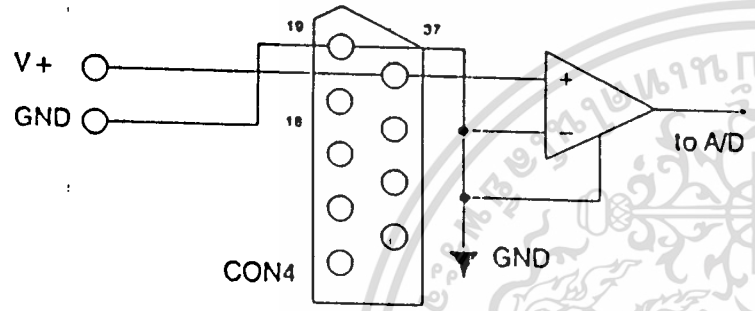


Figure 5-2 A/D Single Ended Input

Differential input

Three leads differentilal inputs are the standard differential inputs. This configuration consists of two input leads and a ground reference. The digital value that is obtained represents the difference between the two input leads with respect to ground.

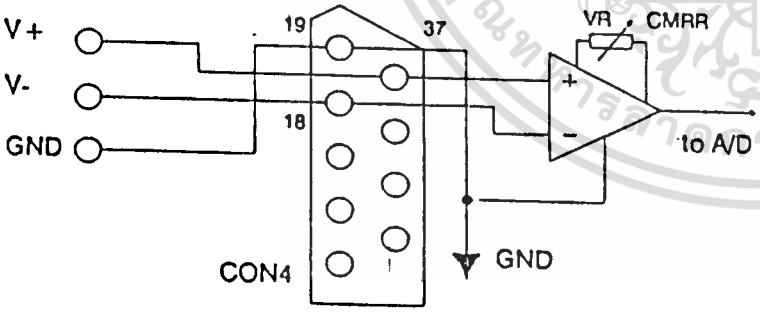


Figure 5-3 A/D Differential I/P

2 leads differential input

Two leads differential input is basically tying one of the differential inputs to ground. The result is a single ended input. The reason for this option is for a user wanting both differential and single ended inputs. The user simply sets the card jumper for differential inputs, then grounds one of the two differential leads for the inputs he wishes to be single ended.

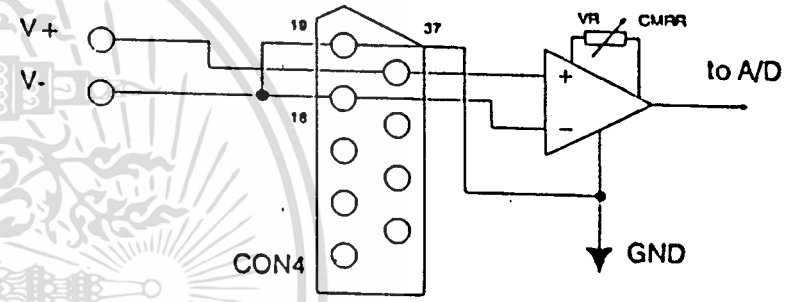


Figure 5-4 A/D 2 Leads Differential I/P

Calibration

Before using the PA-MA12(-H) for a specific A/D task, a user should calibrate the inputs. Calibration enables the highest degree of accuracy this card can provide.

Frequency of calibration

For normal use and environment, calibration every 6 months is recommended. If used in harsh or frequently changing environments, a user is advised to calibrate every 2 to 3 months. The

PA-MA12(-H) should also be recalibrated if the input voltage range for the inputs is changed.

Potentiometer (VR) list

Name	Type	Function
VR1	Potentiometer	A/D CMRR adjust
VR2	Potentiometer	A/D offset adjust
VR3	Potentiometer	A/D range adjust

Equipment requirements

Three connector cables are included with the PA-MA12(-H) for calibration purposes. The additional equipment requirements are a small flat-blade screwdriver - to adjust potentiometers, a precision voltmeter and a precision voltage power supply.

Accuracy of calibration

The accuracy of calibration depends on the accuracy of the equipment used to calibrabte the PA-MA12(-H). For instance, if a 3 1/2 digit digital voltmeter is used to calibrate and a 4 1/2 digit digital voltmeter is used to measure output, output readings may have variations compared to expected values.

Accuracy is also dependent on the accuracy of power supply used in calibration. If the power supply is not stable, the accuracy is poor, the calibration is not possible.

Calibration

There are three adjustments made to calibrate the PA-MA12(-H). The adjustments are, Common Mode Rejection, Offset adjust, and Gain adjust. The calibration is done by putting a known voltage on the inputs and looking at the digital value equivalent.

CMRR (common mode rejection) adjust

The common mode rejection adjust is for differential inputs.

Adjustment is done by tying the two leads for the differential input together and referencing them to ground. No matter what the input voltage is, the differential output (TP2-1) and Ground (TP2-3) should always be zero. If not, the CMRR potentiometer (VR1) should be adjusted to make the output zero. An example is shown here.

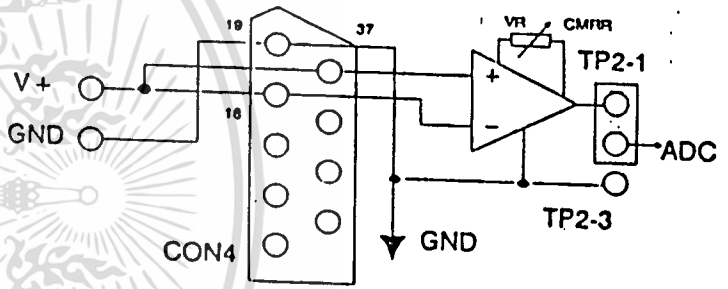


Figure 5-5 A/D CMRR Adjust

Important :

CMRR adjust must be made under bipolar input mode. If the user wishes to config the A/D input range to unipolar input, the user must config the PA-MA12(-H) as bipolar input first to adjust the CMRR then re-config it to unipolar input to continue calibrating the OFFSET and RANGE.

Offset adjust

The offset adjust enables a user to set the zero point for A/D conversions(i.e. a user can set the 0V point to exactly 000H for unipolar or 800H for bipolar, as shown in the table). The procedure to do so is as follows.

1. Ground Inputs.

2. Adjust potentiometer (VR2) with flat-blade screwdriver until 0V input gives the desired value like 000H or 800H.

Full scale adjust (gain)

The gain potentiometer (or range potentiometer) is labled VR3. This variable resistor adjustement enables a user to adjust the full scale range voltage to the FFFH data value, as is shown in the table above. The procedure for adjustment is as follows.

1. Put a known full scale voltage on the inputs.

2. Adjust the range potentiometer until the full scale value gives the desired FFFH data result.

Adjust voltage

Input Ranges		A/D Data Results(Low To High)		
Low	High	000H	800H	FFFH
0V	+10V	0V	+5.000V	+9.9976V
0V	+5V	0V	+2.500V	+4.9988V
-10V	(+10V)9.999V	-10.000V	0V	+9.9951V
-5V	(+5V)4.999V	-5.000V	0V	+4.9976V

Note:

Since there is an odd number (4095) of A/D counts, the full scale voltages are just below their range settings. For example, 0V to 5V range is truncated to 0V to 4.9988V.