ชุดแสดงผลหน้าปัดรถยนต์บนจอคอมพิวเตอร์

Car panel on computer

โดย

นายโยธิน คชวงศ์

นายอาทิตย์ จุลคณานุศาสตร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ชุดแสดงผลหน้าปัดรถยนต์บนจอคอมพิวเตอร์

Car panel on computer

โดย

นายโยธิน คชวงศ์ 47010609

นายอาทิตย์ จุลคณานุศาสตร์ 47010976

อาจารย์ที่ปรึกษา

รศ.จิรวัฒน์ ปานกลาง

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ชุดแสดงผลบนหน้าปัดรถยนต์บนหน้าจอคอมพิวเตอร์ (Car panel on computer)

ผู้จัดทำ

1.นายโยธิน คชวงศ์ รหัส 47010609

2.นายอาทิตย์ จุลคณานุศาสตร์ รหัส 47010976

.....อาจารย์ที่ปรึกษา

(รศ.จิรวัฒน์ ปานกลาง)

ชุดแสดงผลหน้าปัดรถยนต์บนจอคอมพิวเตอร์

นายโยธิน คชวงศ์ รหัส 47010609
นายอาทิตย์ จุลคณานุศาสตร์ รหัส 47010976
รศ.จิรวัฒน์ ปานกลาง อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

โครงงานนี้ได้ทำการออกแบบชุดแสดงผลการทำงานของหน้าปัดรถยนต์ทั่วไปบนหน้าจอของเครื่องคอมพิวเตอร์ส่วนบุคคลให้สามารถแสดงผลข้อมูล ความเร็วของรถยนต์ ความเร็วรอบ และอุณหภูมิของเครื่องยนต์ ค่าระดับเชื้อเพลิงในถังเชื้อเพลิง โดยพยายามเลียนแบบให้เหมือนการแสดงผลแบบเชิงเส้น เช่นเดียวกับหน้าปัดรถยนต์ ปกติซึ่งมีค่าความผิดพลาดต่ำและแสดงผลแบบเวลาจริง

CAR PANEL ON COMPUTER

Mr. Yothin Khotchawong ID.47010609

Mr. Arthit Julkananusart ID.47010976

Assoc Prof. Jirawath Parnklang Advisor

Educational Year 2007

Abstract

This project provides a design of general car panel .The panel which illustrates details of car velocity engine speed, engine temperature and fuel level is displayed on a computer screen. By imitating a analog display, similar to normal car panel, the system are accurate and real-time.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือจากหลายบุคคล โดยเฉพาะอย่างยิ่ง รศ.จิรวัฒน์ ปานกลาง (อาจารย์ที่ปรึกษา) และ อาจารย์อีกหลายท่านที่ได้ประสิทธิประสาทความรู้ให้ จนทำให้โครงงานนี้สำเร็จโดยสมบูรณ์ได้ และที่สำคัญยิ่งก็คือ คุณพ่อและคุณแม่ของผู้จัดทำที่ให้โอกาสและให้กำลังใจแก่ผู้จัดทำมาตลอด ผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย

คณะผู้จัดทำ

นาย โยธิน คชวงศ์

นาย อาทิตย์ จุลคณานุศาสตร์

สารบัญ

สารบัญ (ต่อ)

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

สารบัญตาราง

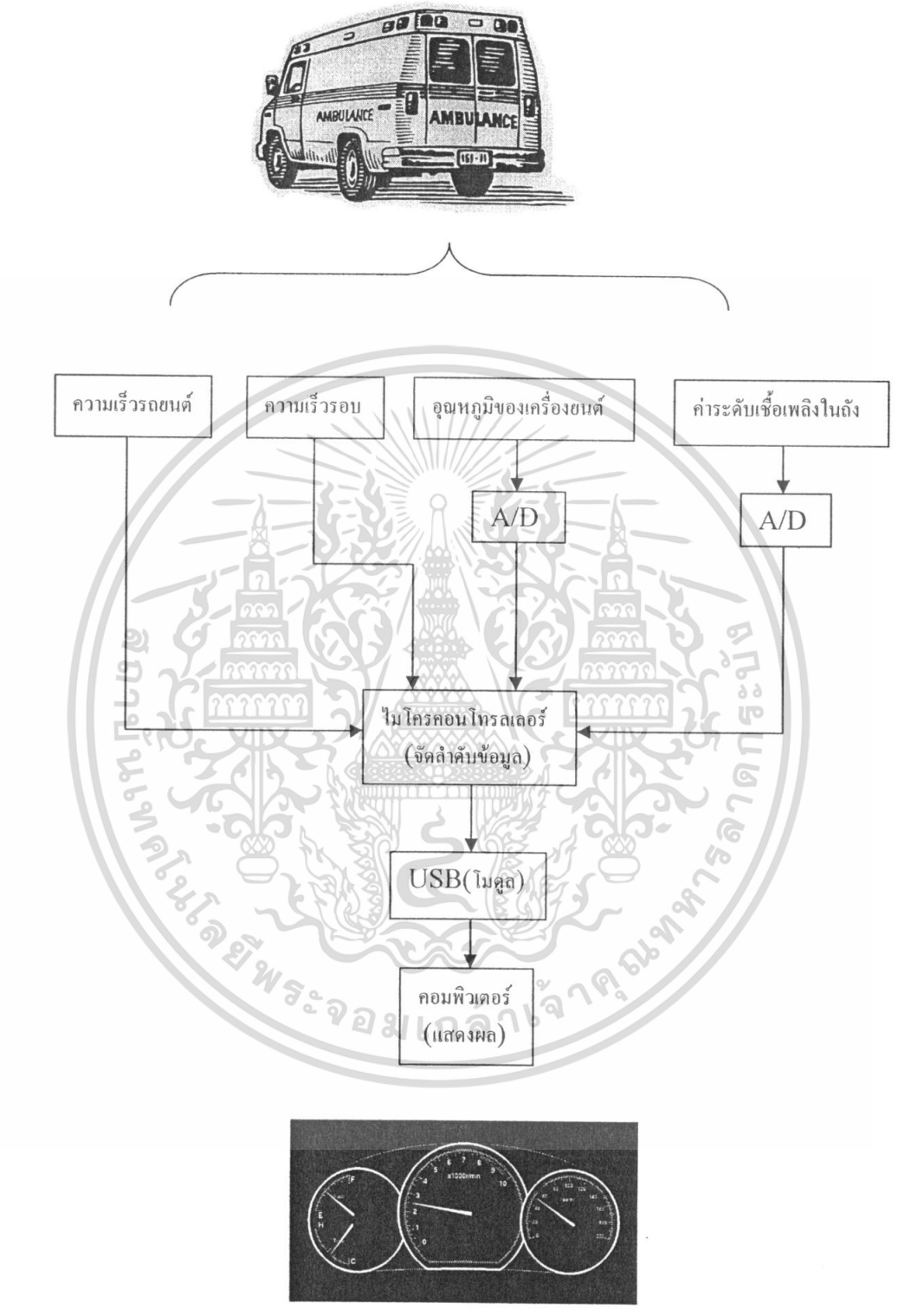
บทที่ 1
บทนำ

เนื่องจากเทคโนโลยีทางด้านการประมวลผลข้อมูลด้วยคอมพิวเตอร์ได้มีการพัฒนาอย่างมาก ซึ่งข้อดีของมัน คือ ใช้งานง่ายและมีความยืดหยุ่นสูง จึงเป็นทางเลือกหนึ่งที่อุตสาหกรรมรถยนต์จะนำเทคโนโลยีนี้มาใช้ งานวิจัยนี้จึงได้ทำการศึกษาระบบการทำงานต่างๆ บนรถยนต์และนำมาแสดงผลบนหน้าจอคอมพิวเตอร์ แนวการทางศึกษา คือ นำเอาสัญญาณต่างๆ บนรถยนต์ที่มาจากอุปกรณ์ตรวจจับเซ็นเซอร์ (sensor) หลักๆ ซึ่งประกอบด้วย ความเร็วรถยนต์ ความเร็วรอบ อุณหภูมิ และค่าระดับเชื้อเพลิงในถังเชื้อเพลิง มาเข้ายังอุปกรณ์ควบคุมไมโครคอนโทรลเลอร์ (microcontroller) ซึ่งเป็นตัวจัดลำดับข้อมูลและแสดงผลผ่านเครื่องคอมพิวเตอร์ โดยใช้ระบบบัสยูเอสบี (USB) ในการส่งผ่านข้อมูลและศึกษาการเขียนโปรแกรมแสดงผลข้อมูล (ซึ่งแสดงผลเป็นหน้าปัดรถยนต์บนหน้าจอคอมพิวเตอร์)

สำหรับโครงงานนี้จะสาธิตการทำงานของไมโครคอนโทรลเลอร์ การส่งข้อมูลผ่านระบบบัส USB และแสดงผลข้อมูลรถยนต์บนหน้าจอคอมพิวเตอร์

วัตถุประสงค์โครงงาน

1. เพื่อสาธิตการทำงานการจัดส่งแบบ USB
2. เพื่อศึกษาหลักการติดต่อกับอุปกรณ์ภายนอกเครื่องคอมพิวเตอร์
3. เพื่อศึกษาหลักการและแนวคิดในการเขียนโปรแกรม เพื่อใช้ในการควบคุมและแสดงผล ซึ่งสามารถนำมาประยุกต์ใช้ในงานอื่นๆ ต่อไป



รูปที่ 1.1 ภาพรวมของการทำงานทั้งหมดในการออกแบบ

บทที่ 2
ทฤษฎี

2.1 ไมโครคอนโทรลเลอร์ MCS 51

ไมโครคอนโทรลเลอร์ที่ใช้ในโครงงานนี้จะอ้างอิงถึงไมโครคอนโทรลเลอร์ ตระกูล MCS 51 ซึ่งมีหน่วยความจำแบบแฟลชของ Atmel Corporation เบอร์ที่ขึ้นต้นด้วย AT89

2.1.1 จัดขาของไมโครคอนโทรลเลอร์ 8051

V_{cc}: สำหรับแหล่งจ่ายไฟฟ้า (+5v.)

V_{ss}: สำหรับต่อกราวด์

P0 : เป็นขาพอร์ต 0 ของ 8051 ที่มีขนาด 8 บิตชนิดสองทิศทาง ซึ่งแต่ละบิตสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไปหากต้องการให้เป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังบิตนั้น โดยแต่ละบิตเมื่อเป็นเอาต์พุตจะสามารถต่อพ่วงกับอุปกรณ์ TTL แบบ LS ได้ 8 ตัว อีกทั้งยังเป็นขาให้สัญญาณ Multiplex ระหว่างสัญญาณข้อมูลกับสัญญาณ Address 8 บิตแรก ในกรณีที่ใช้หน่วยความจำภายนอก

P1 : เป็นขาพอร์ต 1 ของ 8051 ขนาด 8 บิต ชนิดสองทิศทางแบบ Quasi bi-directional ซึ่งแต่ละบิตสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไปหากต้องการให้เป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังบิตนั้น และสามารถต่อพ่วงกับอุปกรณ์ LS TTL ได้ 4 ตัว

P2 : เป็นขาพอร์ต 2 ของ 8051 ขนาด 8 บิต ชนิดสองทิศทางแบบ Quasi bi-directional เช่นเดียวกับพอร์ต 1 นอกจากนี้พอร์ต 2 ยังทำหน้าที่ให้สัญญาณ Address 8 บิตบน ในกรณีที่ใช้หน่วยความจำภายนอก ในกรณีอ้าง Address หน่วยความจำขนาด 16 บิต ดังนั้นขณะที่ใช้หน่วยความจำภายนอกจะต้องไม่มีการเขียนข้อมูลใด ๆไปที่พอร์ต 2 ซึ่งจะทำให้เกิดความผิดพลาดการทำงานได้

P3 : เป็นขาพอร์ต 3 ของ 8051 ขนาด 8 บิต ชนิดสองทิศทางแบบ Quasi bi-directional เช่นเดียวกับขาพอร์ต 1 และพอร์ต 2 แต่พอร์ต 3 นี้จะมีหน้าที่พิเศษดังตารางที่ 2.1

ตารางที่ 2.1 หน้าที่พิเศษของขาต่าง ๆ ของ PORT 3

ขาพอร์ต	หน้าที่พิเศษ
P3.0	R x D (สำหรับรับข้อมูลแบบอนุกรม)
P3.1	T x D (สำหรับส่งข้อมูลแบบอนุกรม)
P3.2	INT0 (ขาอินเตอร์รัพท์ภายนอก 0)
P3.3	INT1 (ขาอินเตอร์รัพท์ภายนอก 1)
P3.4	T0 (ขาอินพุตของ Timer 0)
P3.5	T1 (ขาอินพุตของ Timer 1)
P3.6	WR (สำหรับสัญญาณเขียนหน่วยความจำข้อมูลภายนอก)
P3.7	RD (สำหรับสัญญาณอ่านหน่วยความจำข้อมูลภายนอก)

ดังนั้นเมื่อมีการใช้สัญญาณดังกล่าว จึงไม่ควรเขียนข้อมูลไปที่พอร์ต 3 จะทำให้การทำงานของ 8051 ผิดพลาดได้

RST : เป็นขาสำหรับรีเซตการทำงานของ 8051 โดยการให้ลอจิก 1 เป็นเวลาอย่างน้อย 2 ช่วง Machine Cycle

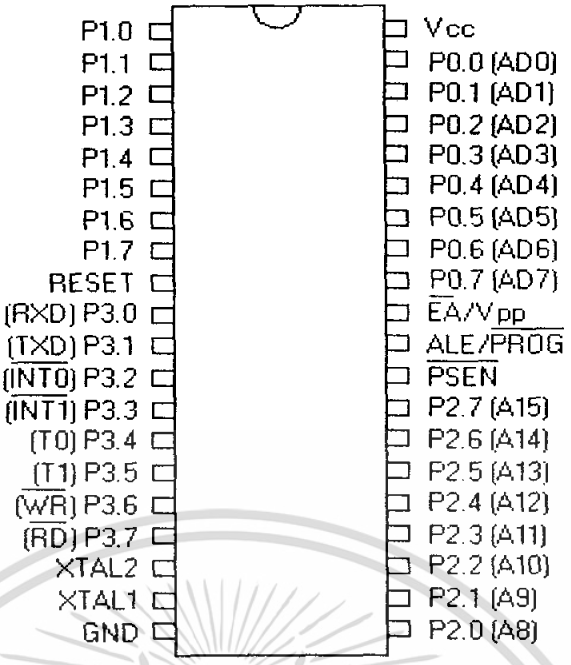
ALE : เป็นขาที่ใช้ในการควบคุมการแลตช์ของขา พอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก

PSEN : เป็นขาสัญญาณเพื่อร้องขอติดต่อหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านหน่วยความจำโปรแกรมภายนอก

EA : เป็นขาสำหรับเลือกการติดต่อหน่วยความจำโปรแกรมภายนอก หรือภายในไมโครคอนโทรลเลอร์ โดยที่ให้ลอจิก 0 จะอ่านหน่วยความจำโปรแกรมภายนอก และลอจิก 1 จะอ่านหน่วยความจำโปรแกรมภายใน

XTAL1 : ขาเข้าของวงจรกำเนิดความถี่อ้างอิงภายในของ 8051

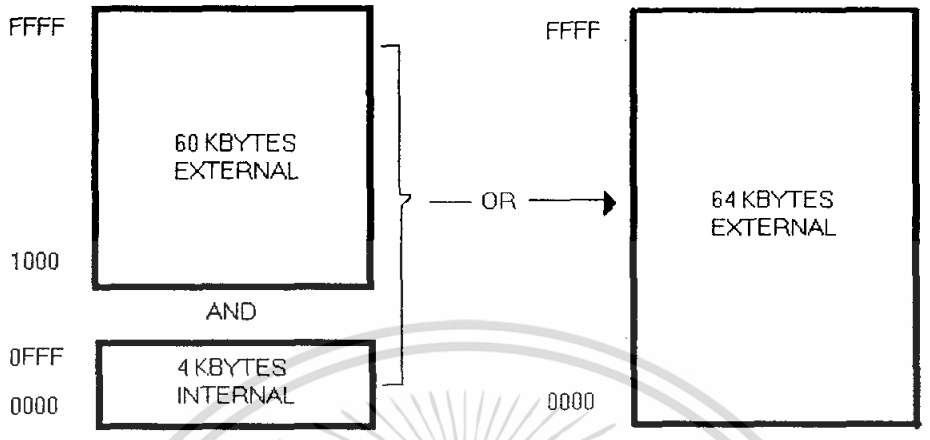
XTAL2 : ขาออกของวงจรกำเนิดความถี่อ้างอิงภายในของ 8051



รูปที่ 2.1 การจัดขาของ 8051

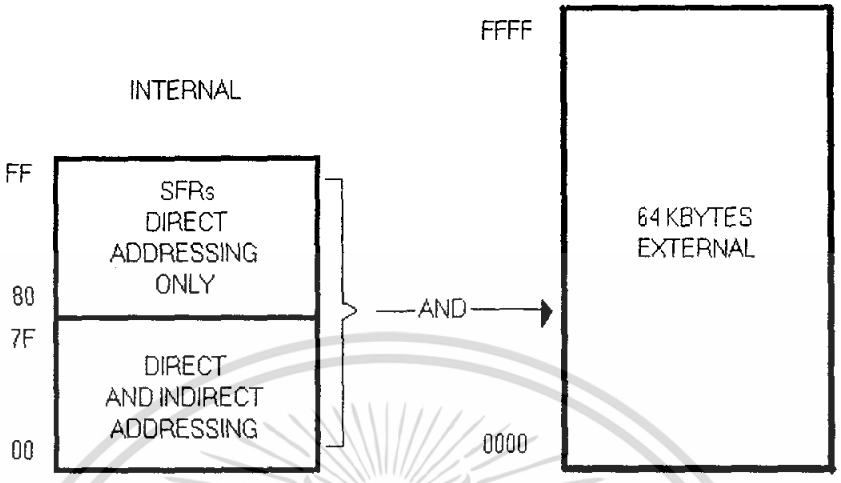
2.1.2 โครงสร้างหน่วยความจำของ 8051

ดังที่กล่าวมาแล้ว 8051 จะแบ่งหน่วยความจำออกเป็นสองส่วน ได้แก่ หน่วยความจำสำหรับโปรแกรมและหน่วยความจำสำหรับเก็บข้อมูล โดยมีขนาดของแต่ละส่วนเท่ากับ 64 กิโลไบต์ ในส่วนของหน่วยความจำโปรแกรมจะเป็นส่วนหน่วยความจำสำหรับอ่านอย่างเดียว โดยที่ 8051 จะใช้สัญญาณ PSEN ในการอ่านเท่านั้น แต่หน่วยความจำข้อมูลของ 8051 จะสามารถอ่านและเขียนได้โดยใช้สัญญาณ RD และ WR ตามลำดับ แต่อย่างไรก็ตาม ผู้ใช้สามารถรวมหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลเข้าด้วยกันได้ โดยนำสัญญาณ RD และ PSEN มาต่อเข้าวงจรแอนเกท สำหรับสร้างสัญญาณในการอ่านหน่วยความจำ นอกจากนี้หน่วยความจำโปรแกรมยังแบ่งออกเป็นภายนอกและภายในของ 8051 ดังแสดงใน รูปที่ 2.1 และรูปที่ 2.2 โดยรูปที่ 2.1 แสดงหน่วยความจำโปรแกรมในกรณีที่เลือกใช้หน่วยความจำภายนอกและภายใน ในด้านซ้ายมือเป็นส่วนหนึ่งของหน่วยความจำโปรแกรมภายในที่มีขนาด 4 กิโลไบต์ของ 8051 ส่วนที่เหลือจะเป็นหน่วยความจำภายนอก ส่วนด้านขวามือแสดงหน่วยความจำโปรแกรมเมื่อเลือกให้ติดต่อหน่วยความจำภายนอกทั้งหมด



รูปที่ 2.2 หน่วยความจำโปรแกรมของ 8051

สำหรับหน่วยความจำข้อมูลของ 8051 สามารถแบ่งออกเป็นภายนอกและภายในโดยหน่วยความจำภายนอกแสดงไว้ด้านขวามือของรูปที่ 2.2 ซึ่งมีขนาด 64 กิโลไบต์ ส่วนหน่วยความจำข้อมูลภายในแสดงไว้ด้านซ้ายของรูปที่ 2.2 โดยหน่วยความจำภายในของ 8051 แบ่งออกเป็น 2 ส่วน ได้แก่ ส่วนของหน่วยความจำข้อมูลที่สามารถอ้างอิงแบบ Direct และ Indirect ซึ่งมีขนาด 128 ไบต์ กับหน่วยความจำที่อ้างอิงได้เฉพาะแบบ Direct หรือในส่วนนี้จะเรียกอีกแบบหนึ่งว่า SFR (Special Function Register) โดยสามารถแบ่งได้ดังนี้

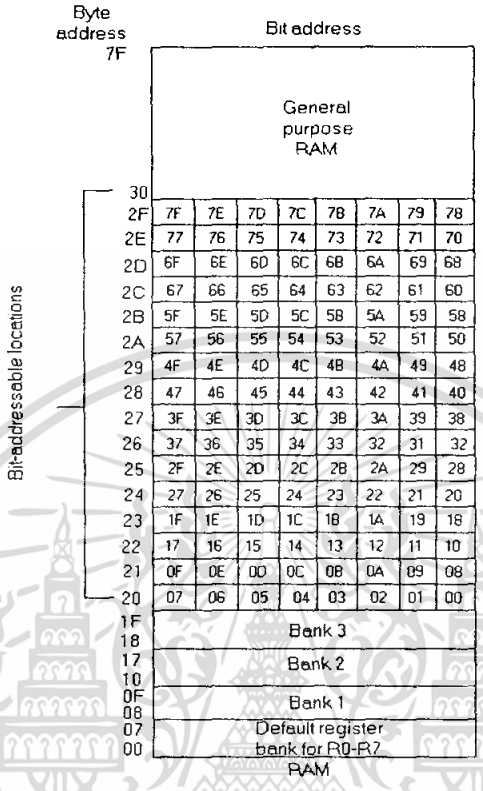


รูปที่ 2.3 หน่วยความจำข้อมูลของ 8051

ส่วนของหน่วยความจำข้อมูลภายในที่อ้างอิงแบบ direct และ Indirect นั้นจะสามารถแบ่งออกได้ 3 ส่วน ดังแสดงในรูปที่ 2.3 โดยมีรายละเอียดดังนี้

- ส่วนที่ 1 เรียกว่า Register Banks 0-3 ซึ่งอยู่ที่ตำแหน่งความจำข้อมูลภายในตั้งแต่ 00H ถึง 1FH จำนวน 32 ไบต์ โดยแบ่งออกเป็นชุดละ 8 ไบต์ จำนวน 4 ชุด ซึ่งแต่ละชุดจะมีชื่อเรียกเป็น R0 ถึง R7 จะเป็น Register ที่ใช้งาน โดยเมื่อ 8051 ถูกรีเซต Register Bank 0 จะถูกเลือกใช้

- ส่วนที่ 2 เรียกว่า Bit Addressable Area ซึ่งมีขนาด 16 ไบต์ที่ตำแหน่งหน่วยความจำข้อมูล 20H ถึง 2FH ในส่วนนี้สามารถอ้างอิงข้อมูลได้ถึงระดับ 128 บิต โดยการอ้างอิงตำแหน่งโดยตรงในลักษณะบิต ตั้งแต่ตำแหน่ง 00H ถึง 7FH



รูปที่ 2.4 หน่วยความจำข้อมูลภายใน

- <u>ส่วนที่ 3</u> เรียกว่า Scratch Pad Area ซึ่งตำแหน่ง 30 H ถึง 7 FH โดยเป็นบริเวณหน่วยความจำข้อมูลภายในเอนกประสงค์ที่ผู้ใช้สามารถใช้ได้โดยตรง นอกจากนี้ยังสามารถใช้หน่วยความจำข้อมูลบริเวณนี้สำหรับการเก็บข้อมูลแบบ Stack ได้ด้วยในส่วนของหน่วยความจำข้อมูลภายในที่ใช้อ้างอิงแบบ Direct เพียงอย่างเดียวหรือที่เรียกว่า SFR ซึ่งเป็นส่วนสำหรับเก็บหรือกำหนดการทำงานภายในของ 8051 ดังแสดงในรูปที่ 2.4

ในส่วนของบริเวณนี้จะมีขนาด 128 ไบต์แต่ในการใช้งานนั้นใช้ได้เฉพาะตำแหน่งซึ่งแสดงไว้ในรูปที่ 2.4 เท่านั้น หากผู้ใช้อ้างตำแหน่งนอกเหนือจากนี้นั้นจะได้ข้อมูลที่คาดเดาไม่ได้ โดยแต่ละตำแหน่งจะมีหน้าที่ดังนี้

ACC : Accumulator ซึ่งเป็นรีจีสเตอร์สำหรับการประมวลผลทางคณิตศาสตร์และลอจิก โดยผู้ใช้สามารถอ้างอิงได้ในรูปแบบของไบต์หรือระดับบิตได้

B : รีจีสเตอร์พิเศษสำหรับใช้กับคำสั่งในการคูณหรือหาร นอกจากนี้ยังใช้เป็นรีจีสเตอร์สำหรับเก็บพักข้อมูลได้

PSW : เป็นรีจีสเตอร์ Program Status Word หรือแฟลกจะแสดงสถานการณ์ทำงานของ 8051 สำหรับการตรวจสอบจะกล่าวถึงรายละเอียดในภายหลัง

8 Bytes

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW(1)							D7
C8	T2CON(1)(2)	T2MOD(2)	RCAP2L(2)	RCAP2H(2)	TL2(2)	TH2(2)		CF
C0								C7
B8	IP(1)							BF
B0	P3							B7
A8	IE(1)							AF
A0	P2							A7
98	SCON(1)	SBUF						9F
90	P1							97
88	TCON(1)	TMOD(1)	TL0	TL1	TH1			8F
80	P0	SP	DPL	DPH			PCON(1)	87

↑ Bit Addressable

Notes: 1. SFRs converting mode or control bits
2. AT89C52 only

รูปที่ 2.5 รายละเอียดของ Special Function Register

SP : เป็นรีจีสเตอร์สำหรับชี้หน่วยความจำข้อมูลภายในสำหรับการเก็บแบบ Stack

DPTR : เป็นรีจีสเตอร์ขนาด 16 บิต โดยแบ่งเป็น 8 บิตบนและ 8 บิตล่าง ใช้สำหรับชี้ตำแหน่งของหน่วยความจำข้อมูลภายนอก หรือสำหรับอ่านตารางข้อมูลของหน่วยความจำโปรแกรม

P0 : รีจีสเตอร์สำหรับพอร์ต 0 ของ 8051

P1 : รีจีสเตอร์สำหรับพอร์ต 1 ของ 8051

P2 : รีจีสเตอร์สำหรับพอร์ต 2 ของ 8051

P3 : รีจีสเตอร์สำหรับพอร์ต 3 ของ 8051

IP : รีจีสเตอร์สำหรับกำหนดลำดับความสำคัญของการอินเตอร์รัพท์ของ 8051

IE : รีจีสเตอร์สำหรับกำหนดการรับหรือไม่รับการอินเตอร์รัพท์ของ 8051

TMOD : รีจีสเตอร์สำหรับควบคุมหน้าที่ของ Timer/Counter ของ 8051

TCON :รีจีสเตอร์สำหรับควบคุมการทำงานของ Timer/Counter ของ 8051

T2CON : รีจีสเตอร์สำหรับควบคุมการทำงานของ Timer/Counter 2 ของ 8052

TH0 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 0 8บิตบน

TL0 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 0 8บิตล่าง

TH1 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 1 8บิตบน

TL1 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 1 8บิตล่าง

TH2 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 2 8บิตบนของ 8052

TL2 : รีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 2 8บิตล่างของ 8052

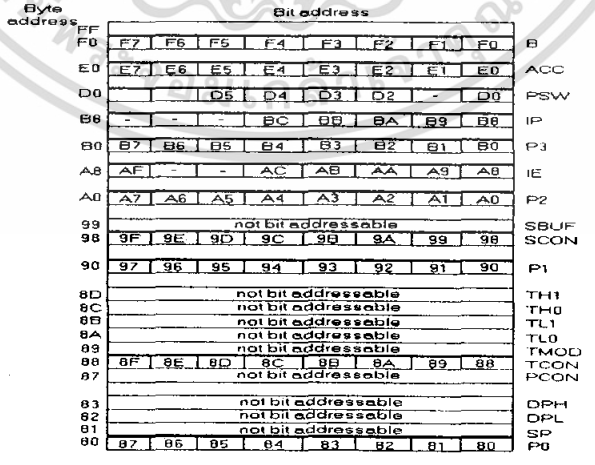
RCAP2H: Capture Register ของ Timer/Counter 2 8บิตบนของ 8052

SCON : รีจีสเตอร์สำหรับควบคุมการรับส่งข้อมูลแบบอนุกรมของ 8051

SBUF : รีจีสเตอร์สำหรับเก็บพักข้อมูลที่ได้จากการรับส่งข้อมูลแบบอนุกรมของ MCS - 8051

PCON : รีจีสเตอร์สำหรับควบคุมการทำงานของ MCS-8051 ด้านเกี่ยวกับการใช้กำลังไฟฟ้า

ในส่วนของรีจีสเตอร์ SFR นี้สามารถที่จะอ้างอิงในระดับบิตได้โดยตำแหน่งการอ้างอิงระดับบิตซึ่งได้แสดงไว้ในตาราง รูปที่ 2.5



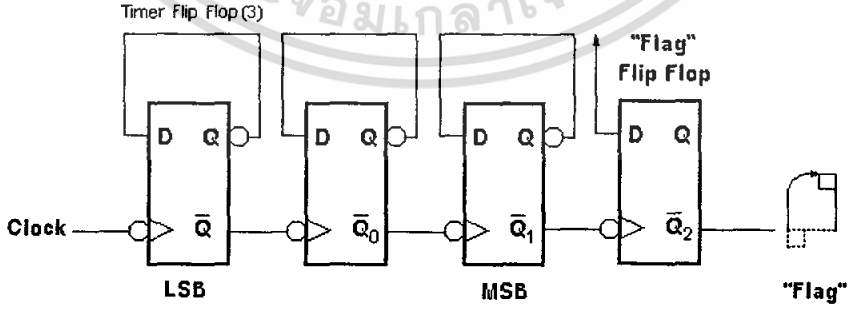
รูปที่ 2.6 ตำแหน่งการอ้างอิงระดับบิตของรีจิสเตอร์ SFR

2.1.3 TIMER

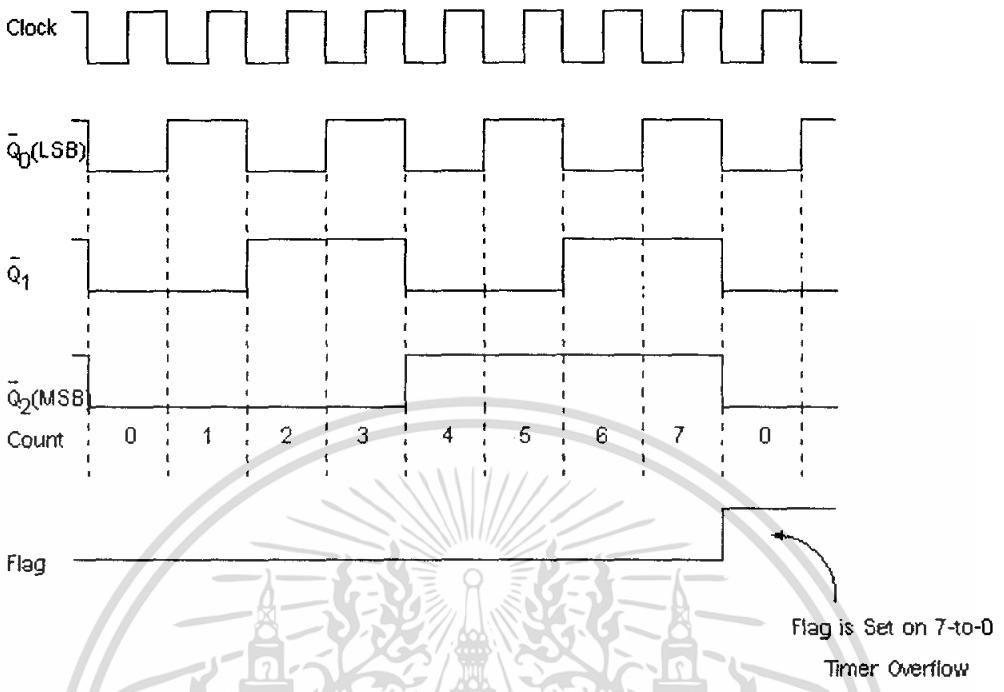
ตัว Timer อาจพิจารณาได้ง่าย ๆ ว่าเป็นตัวฟลิปฟลอปมาต่อเรียงกันโดยมี Clock เป็นอินพุท สำหรับเอาต์พุทที่ออกมาจากฟลิปฟลอปแต่ละตัวจะถูกหารด้วย 2 พิจารณาการต่อฟลิปฟลอปตามรูปที่ 2.6 ถ้าใส่ Clock เข้าไปในฟลิปฟลอปตัวแรก ความถี่ของ Clock ที่ออกจากเอาต์พุทตัวแรกจะถูกหารด้วย 2 และเอาต์พุทนี้จะต่อกับฟลิปฟลอปตัวที่สอง สัญญาณที่ออกมาจะถูกหารด้วย 2 อีก ดังนั้นถ้ามีฟลิปฟลอปต่ออยู่ n Stages จะหารสัญญาณนาฬิกาได้ 2^n ถ้าให้เอาต์พุท Stage สุดท้ายของ Timer เป็น Overflow Flip-Flop หรือ Flag และจะให้เอาต์พุทออกมาเมื่อการนับเป็น Overflow เช่น ถ้าเป็นตัวนับแบบ 16 บิต (มีฟลิปฟลอปต่ออยู่ 16 ตัว) วงจรจะนับตั้งแต่ 0000H ถึง FFFFH เมื่อฟลิปฟลอปเปลี่ยนจาก FFFFH เป็น 0000H จะให้บิต Overflow ออกมา

พิจารณารูป 2.6 (ก) เป็น 3-bit Timer โดยฟลิปฟลอปแต่ละตัวจะนำขา Q มาต่อกับ D ซึ่งอาจเรียกว่า เป็นการใช้ฟลิปฟลอปแบบ Divide-by-two Mode โดยความถี่ของสัญญาณที่ได้จากฟลิปฟลอปแต่ละตัวจะมีค่าหารสองจากสัญญาณนาฬิกาที่เข้ามา เมื่อนับไปถึงค่า 111 (หรือ $Q_2 = 1$, $Q_1 = 1$, $Q_0 = 1$) และเปลี่ยนกลับมาเป็น 000 จะให้บิต Flag ออกมา ดังแสดงในรูปที่ 2.6 (ข)

ใน MCS – 51 จะมีตัวจับเวลาอยู่ภายในชิพ ถ้าเป็นเบอร์ 8051 หรือ 8031 จะมี 2 ตัว คือ Timer 0 และ Timer 1 แต่ถ้าเป็นเบอร์ 8052 จะมีเพิ่มอีกหนึ่งตัวคือ Timer 2 รีจีสเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการใช้ Timer แสดงได้ดังตารางที่ 2.2 ซึ่งจะเห็นว่ามีรีจีสเตอร์บางตัวสามารถเข้าถึงข้อมูลระดับบิตได้ด้วย นอกจากนี้ตัว Timer สามารถใช้เป็นตัวนับ (Counter) ได้อีกด้วย โดยการโปรแกรมในรีจีสเตอร์ TMOD



(ก)



(ข)

รูปที่ 2.7 รีจีสเตอร์ที่ใช้เป็น Timer

ตารางที่ 2.2 รีจีสเตอร์ที่ใช้เป็น Timer

รีจีสเตอร์	หน้าที่	ตำแหน่ง	สามารถอ้างอิงตำแหน่งบิต
TCON	Control	88H	Yes
TMOD	Mode	89H	No
TL_0	Timer 0 Low-byte	8AH	No
TL_1	Timer 1 Low-byte	8BH	No
TH_0	Timer 0 High-byte	8CH	No
TH_1	Timer 1 High-byte	8DH	No
T_2CON*	Timer 2 Control	C8H	Yes
$RCAP_2L$*	Timer 2 Low-byte Capture	CAH	No
$RCAP_2H$*	Timer 2 High-byte Capture	CBH	No
TL_2*	Timer 2 Low-byte	CCH	No
TH_2*	Timer 2 High-byte	CDH	No

* มีในเบอร์ 8032 / 8052

2.1.3.1 Timer Mode Register (TMOD)

ตัวรีจีสเตอร์ TMOD เป็นรีจีสเตอร์ควบคุม Timer จะแบ่งออกเป็น 2 กลุ่ม กลุ่มละ 4 บิต โดย 4 บิตบนจะเป็นการควบคุม Timer 1 ส่วน 4 บิตล่างจะเป็นการควบคุม Timer 0 ความหมายของแต่ละบิตดูในตารางที่ 2.2 ซึ่งตัวรีจีสเตอร์นี้เป็นตัวเลือกการทำงานว่าจะให้ตัว Time/Counter ทำงานในโหมดใด และเป็น Timer หรือ Counter รีจีสเตอร์ TCON ไม่สามารถจะโปรแกรมเข้าไปในระดับบิตได้ (Not Bit-Addressable) ซึ่งการใช้งานมักจะโปรแกรมเข้าไปครั้งเดียวในตำแหน่งเริ่มต้นของโปรแกรม

ตารางที่ 2.3 รีจีสเตอร์ TMOD (Timer Mode)

บิต	ชื่อ	Timer	ความหมาย
7	GATE	1	Gate bit ถ้าบิตนี้เซตวงจรจะทำงาน เมื่อ INT1 เป็น High
A	C/T	1	เป็นบิตเลือก Counter / Timer 1 = ใช้เป็น Counter 0 = ใช้เป็น Timer
5	M1	1	Mode bit 1 (ดูตาราง 5-3)
4	M0	1	Mode bit 0 (ดูตาราง 5-3)
3	GATE	0	บิต Gate ของ Timer 0
2	C/T	0	บิตเลือก Counter / Timer ของ Timer 0
1	M1	0	Timer 0 M1 bit
0	M0	0	Timer 0 M0 bit

ตารางที่ 2.4 การใช้ Timer โหมดต่าง ๆ

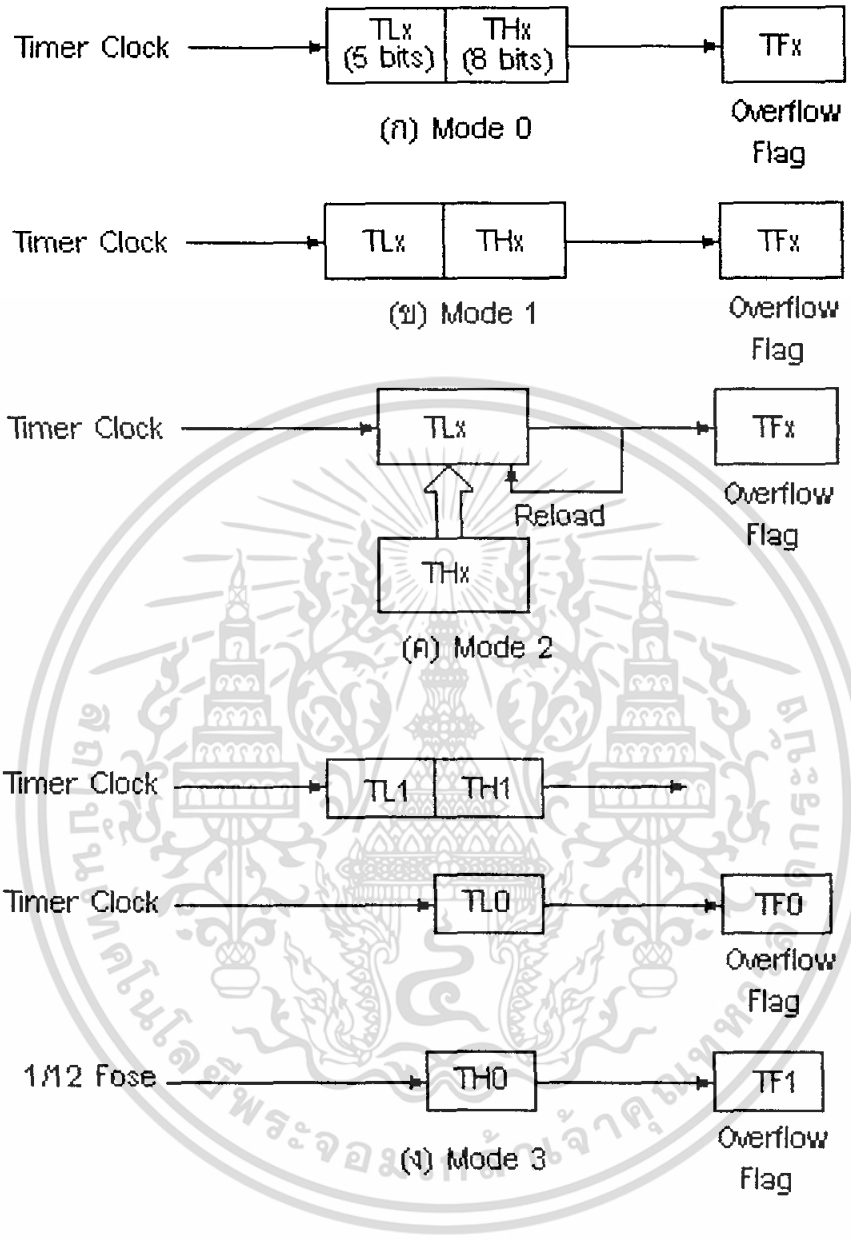
M1	M0	Mode	ความหมาย
0	0	0	ใช้เป็น Timer แบบ 13-bit (8048 Mode)
0	1	1	ใช้เป็น Timer แบบ 16-bit
1	0	2	ใช้เป็น Timer แบบ 8-bit Auto-reload Mode
1	1	3	Split Timer Mode : แยก Timer 0 ออกเป็น Timer 8 บิตสองตัวคือ TL0 และ TH0 โดยไม่ใช้ Timer 1

2.1.3.2 Timer Control Register (TCON)

รีจีสเตอร์ TCON เป็นรีจีสเตอร์ที่บอกสถานะและควบคุมบิต Timer 0 และ Timer 1 ซึ่งดูได้จากตารางที่ 2.5 รีจีสเตอร์นี้สามารถเข้าถึงข้อมูลระดับบิตได้

ตารางที่ 2.5 ความหมายแต่ละบิตของรีจีสเตอร์ TCON (Timer Control)

บิต	ชื่อ	ตำแหน่งบิต	ความหมาย
TCON.7	TF1	8FH	บิตแฟลกแสดงการโอเวอร์โฟลว์ของ Timer 1 จะ Set โดย Hardware และ Clear โดย Software
TCON.6	TR1	8EH	บิตควบคุมการปิด-เปิด Timer 1 Set และ Clear โดย Software
TCON.5	TF0	8DH	แฟลกแสดงการโอเวอร์โฟลว์ของ Timer 0
TCON.4	TR0	8CH	บิตควบคุมการปิด-เปิด Timer 0
TCON.3	IE1	8BH	บิตแฟลกแสดงการอินเทอร์รัพท์จาก INT1 จะ Set โดย Hardware และสามารถ Clear ได้ด้วย Software
TCON.2	IT1	8AH	บิตเลือกชนิดของสัญญาณอินเทอร์รัพท์จากอินเทอร์รัพท์ภายนอก INT1 สามารถ Set และ Clear ได้ด้วย Software
TCON.1	IE0	89H	บิตแฟลกแสดงการอินเทอร์รัพท์จาก INT0
TCON.0	IT0	88H	บิตเลือกชนิดของสัญญาณอินเทอร์รัพท์จากอินเทอร์รัพท์ภายนอก INT0



รูปที่ 2.8 การทำงานของ Timer ในโหมดต่าง

2.1.3.3 Timer Mode And Overflow Flag

เมื่อใช้ Timer 0 และ Timer 1 จะต้องใช้รีจีสเตอร์คู่ TLx และ THx โดยค่า x จะเป็นตัวบอกว่าเป็น Timer 0 หรือ Timer 1 การใช้ Timer สามารถใช้งานได้หลายโหมด ดังแสดงในรูปที่ 2.7 ซึ่งเราสามารถเซตค่าโหมดการทำงานได้ โดยการโปรแกรมในรีจีสเตอร์ TMOD

1. 13-Bit Timer Mode (Mode 0)

การทำงานในโหมด 0 นี้จะเป็นการใช้ Timer แบบ 13 บิต ดังแสดงในรูป 2.7(ก) ซึ่งจะใช้ 5 บิตล่างของ TLx โดยไม่สนใจ 3 บิตที่เหลือ และ 8 บิต ของ THx การทำงานในโหมดนี้เมื่อบิตของ TLx นับไปจนเป็น "1" ทุกบิตจะส่ง Clock 1 ลูกให้ หนึ่งลูกให้ THx นับต่อและเมื่อนับเป็น "1" ทุกบิต และเปลี่ยนกลับเป็น "0" จะเกิด Overflow Flag เกิดขึ้น

2. 16-Bit Timer Mode (Mode 1)

การทำงานในโหมดนี้จะเหมือนกับการทำงานในโหมด 0 แต่เป็น Timer แบบ 16 บิต ซึ่งการนับจะเริ่มตั้งแต่ 0000H,0001H,0002H ไปเรื่อย ๆ และจะเกิด Overflow ขึ้น เมื่อมีการเปลี่ยนจาก FFFFH เป็น 0000H ดังรูปที่ 2.7(ข) ซึ่งเป็นการเซต Overflow Flag และค่านี้จะเกิดขึ้นในบิต TFx ของรีจีสเตอร์ TCON ซึ่งสามารถอ่านและเขียนด้วยโปรแกรม

การใช้ตัว Timer นี้ค่าของบิตสูงสุด (MSB) คือค่าบิต 7 ของ THx ส่วนบิตต่ำสุด (LSB) คือบิต 0 ของ TLx บิต LSB จะเป็น Toggles เมื่อมีสัญญาณอินพุตเข้ามา ถูกหารด้วย 2 ดังนั้นจะพบว่าบิต MSB จะ Toggles ด้วยค่าความถี่ของสัญญาณอินพุทหารด้วย 65,536 (2^{16}) และค่า Timer รีจีสเตอร์นี้ (TLx/THx) สามารถอ่านและเขียนได้ด้วยการโปรแกรม ดังนั้นสามารถนำไปประยุกต์ใช้งานได้ตามต้องการ

3. 8-Bit Auto – Reload Mode (Mode 2)

การทำงานในโหมด 2 เรียกอีกอย่างหนึ่งว่า 8-bit Auto – reload Mode โดยใช้ Timer ไบต์ต่ำ (TLx) เป็น Timer แบบ 8 บิต เมื่อไบต์ต่ำเกิด Overflows หรือเกิดการเปลี่ยนแปลงจาก FFH เป็น 00H จะมีการโหลดค่าที่เก็บไว้ในไบต์สูง (THx) ไปเก็บไว้ในไบต์ต่ำ (TLx) ซึ่งจะเป็นค่าเริ่มต้นของการนับครั้งต่อไป นิยมใช้สร้างเป็นฐานเวลาที่สามารถโปรแกรมได้ การทำงานในโหมดนี้แสดงดังรูปที่ 2.7 (ค)

4. Split Timer Mode (Mode 3)

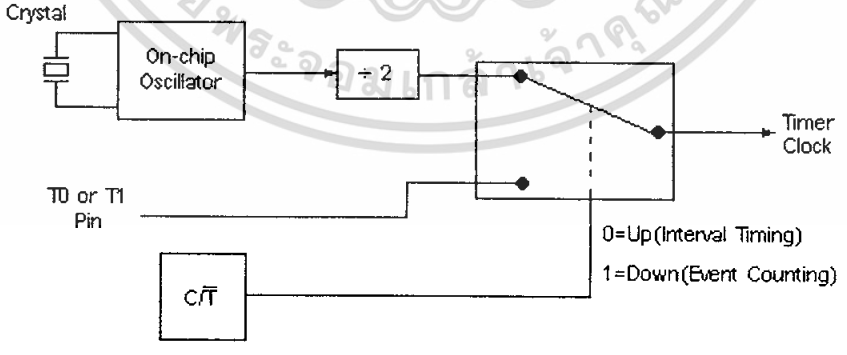
การทำงานในโหมด 3 นี้ ตัว Timer 1 จะไม่ทำงาน ตัว Timer 0 จะแยกเป็น 2 ตัว ตัวละ 8 บิต คือ TL0 และ TH0 เมื่อ Timer เกิด Overflows จะมีการเซตบิต TF0 และ TF1 ดังแสดงในรูปที่ 2.7(ง) การทำงานในโหมด 3 นี้ Timer 1 จะไม่ถูกใช้งานแต่เราสามารถสวิตช์ให้ Timer 1 ไปทำงานในโหมดอื่นได้ แต่การทำงานของ Timer 1 จะไม่มีการอินเทอร์รัพท์เกิดขึ้น เพราะบิต TF1 ถูกใช้ในการนับของ TH0 ในการทำงานของโหมด 3 ไปแล้ว เราอาจมองว่าถ้าให้ Timer ทำงานใน โหมด 3 ทำให้เรามี Timer เพิ่มขึ้น คือ TH0 และ TL0 ใน Timer 0 โหมด 3 และโปรแกรมให้ Timer 1 ไปทำงานในโหมดอื่น ๆ

5. Clocking Source

ในรูปที่ 2.7 ไม่ได้แสดงว่า Timer Clock นำมาจากที่ใดซึ่งการใช้ Timer นี้ สามารถใช้ได้ 2 หน้าที่ คือ เป็นตัวจับเวลา (Timer) และเป็นตัวนับ (Counter) ซึ่งสามารถโปรแกรมได้โดยการเซตหรือรีเซตบิต C / Tในรีจีสเตอร์ TMOD

6. การใช้เป็นตัวจับเวลา (Timer)

ถ้าบิต C / Tใน TMOD เป็นลอจิก "0" จะเป็นการเลือกให้ Timer นำ Clock มาจากวงจร Oscillator ในชิพ ซึ่งสัญญาณนาฬิกาจะเข้ามาทุก ๆ Machine Cycle หรืออาจกล่าวได้ว่าค่าใน THx และ TLx จะมีค่าเพิ่มขึ้นด้วยอัตราการนับแต่ละครั้งใช้เวลาเท่ากับ 1/12 ของความถี่ของสัญญาณนาฬิกาที่ใช้บนชิพ ดังแสดงในรูปที่ 16 ถ้า MCS – 51 ใช้สัญญาณนาฬิกา 12 MHz การนับจะมีความถี่เท่ากับ 1 MHz



รูปที่ 2.9 ความถี่ของสัญญาณนาฬิกาที่เข้าขา Timer

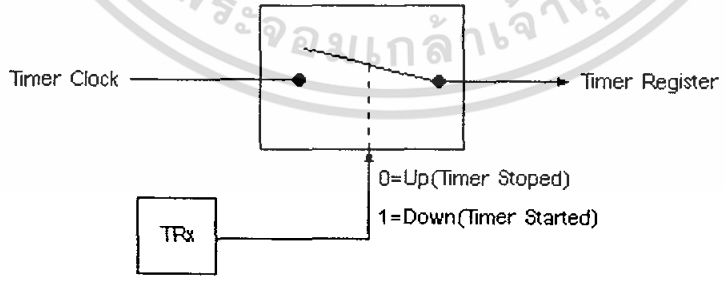
7. การใช้เป็นตัวนับ (Counter)

ถ้าบิต C / T เป็น "1" ตัว Timer จะนำ Clock มาจากภายนอกโดยใช้ขา P3.4 หรือ T0 เป็นขา Input Clock ให้กับ Timer 0 และใช้ขา P3.5 หรือ T1 เป็น Input Clock ให้กับ Timer 1 ดังรูปที่ 2.8 หรืออาจมองว่าถ้าจะให้นับอะไรสัญญาณที่จะนับให้ต่อกับขา T0 และ T1 ในการใช้เป็น Counter สัญญาณที่เข้ามามีการเปลี่ยนแปลงจาก "1" เป็น "0" จะทำให้วงจรนับ TLx มีค่าเพิ่มขึ้น 1 ภายใน MCS – 51 นี้จะตรวจสอบขาอินพุต T0 และT1 ในช่วงเวลาเฟส 2 ของ State 5 (S5P2) ถ้าพบว่ามีค่าเป็น "1" ต่อมาในอีกหนึ่ง Machine Cycle ที่เฟส 2 ของ State 5 (S5P2) ลอจิกอินพุตเปลี่ยนเป็น "0" จะทำให้ค่าใน Timer เพิ่มขึ้น 1 ดังนั้น จะเห็นได้ว่าการนับ 1 ครั้ง จะต้องใช้เวลา 2 Machine Cycles ดังนั้นความถี่สูงสุดที่จะให้ Timer ทำงานเป็น Counter นับได้จะมีค่ามากที่สุด 500 kHz ถ้า MCS – 51 ทำงานที่ความถี่สัญญาณนาฬิกา 12 MHz

2.1.3.4 การเริ่ม , หยุด และการควบคุม Timer

ในรูปที่ 2.7 แสดงลักษณะของ Timer Registers ซึ่งเห็นได้ว่าประกอบด้วย TLx และ THx และเมื่อเกิด Overflow จะเกิดเอาต์พุทที่บิต TFx สำหรับสัญญาณนาฬิกาที่จะเข้าไปใน Time จะมาจาก 2 ส่วนดังแสดงในรูปที่ 2.8 ต่อไปจะกล่าวถึงว่าเราจะควบคุมให้เริ่มหรือหยุดตัว Timer ได้อย่างไร

วิธีเริ่มและหยุดตัว Timers สามารถควบคุมได้ที่บิต TRx ในรีจีสเตอร์ TCON โดยปกติแล้ว TRx จะเคลียหลังจากที่ระบบถูกรีเซต ซึ่งจะเป็นการให้ Timer ไม่นับและ TRx นี้จะเซตได้จากชุดคำสั่ง หรือการโปรแกรม พิจารณารูปที่ 2.10



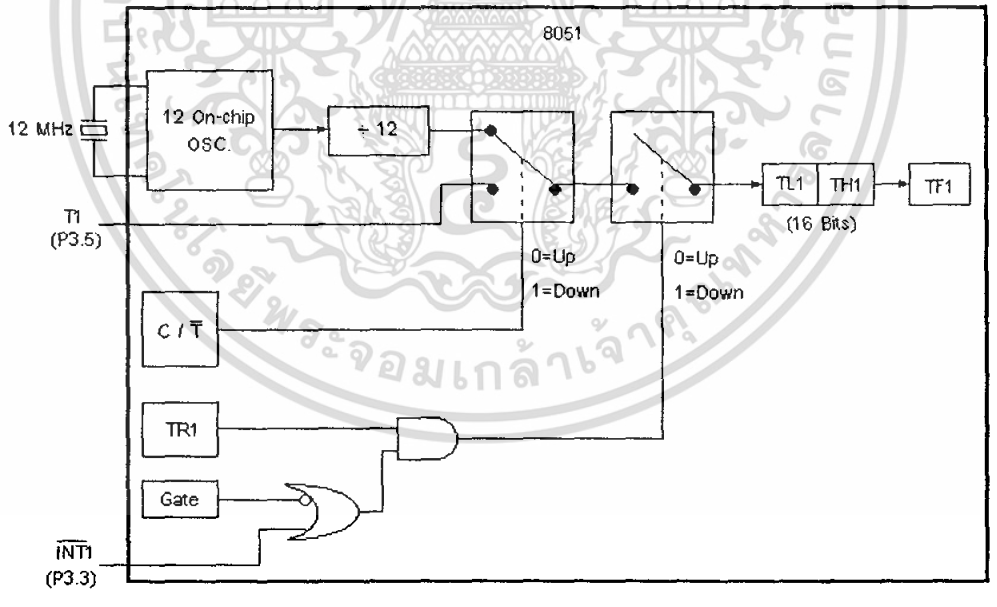
รูปที่ 2.10 การใช้บิตควบคุม TR

ตัวบิต TRx จะเป็นส่วนที่สามารถเข้าถึงข้อมูลในระดับบิตได้ (Bit Addressable)ในรีจีสเตอร์ TCON ถ้าจะให้ TIMER 0 เริ่มทำงานจะเขียนคำสั่งเป็น **SETB TR0**

ถ้าจะหยุดทำงานเขียนคำสั่งเป็น **CLR TR0**

ในการเขียนโปรแกรมภาษาแอสแซมบลี สามารถใช้สัญลักษณ์ TR0 ในคำสั่ง SETB TR0 เลยได้ เพราะตัวแอสแซมบลีจะตีความ TR0 เป็น Bit Address ตำแหน่ง 8 CH วิธีควบคุม Timer สามารถควบคุมได้ที่บิต GATE ใน TMOD และขาอินเตอร์รัพท์จากภายนอก INTx ถ้า INT0 เป็นลอจิก "0" และโปรแกรมให้ Timer 0 ทำงานในโหมด 2 เมื่อ TL0/TH0 = 0000H, GATE = 1 และ TR0 = 1 เมื่อ INT0 ขึ้นเป็นลอจิก "1" ตัว Timer จะ "Gate On" และจะให้สัญญาณนาฬิกาความถี่ 1 MHz เมื่อ INT0 ลงเป็น "0" ตัว Timer " Gate Off " สัญญาณที่ได้จะมีความกว้างของสัญญาณนาฬิกา 1 μS ส่งเข้าไปใน TL0/TH0

รูปที่ 2.10 เป็นระบบที่สมบูรณ์ของ Timer 1 เมื่อทำงานในโหมด 1 ซึ่งเป็น 16-bit Timer โดยใช้รีจีสเตอร์ TL1 / TH1 และ Overflow Flag TF1 ในรูปจะเห็นถึงการควบคุมแหล่งกำเนิด Clock การเริ่มทำงาน และการหยุดทำงาน



รูปที่ 2.11 ระบบทั้งหมดของ Timer 1

2.1.3.5 Intializing And Accessing Timer Register

การใช้งาน Timer เริ่มแรกจะต้องโปรแกรมเพื่อเลือกโหมดการทำงานของ Timer ก่อน เมื่อเริ่มใช้งานก็โปรแกรมให้ เริ่มทำงาน, หยุดทำงาน, อ่าน และ เคลียร์ค่า Flag Bits อ่านค่า Timer Registers ตามลำดับ เพื่อนำไปประยุกต์การใช้งานต่อไป

TMOD คือ รีจีสเตอร์ที่ต้องโปรแกรม โดยเซตโหมดการทำงานก่อน ตัวอย่างเช่น ถ้าให้ Timer 1 เป็น 16-bits Timer (โหมด 1) นับสัญญาณนาฬิกาบนชิพ สามารถเขียนคำสั่งได้ดังนี้

MOV TMOD, #00010000B

ผลที่ได้จากคำสั่งข้างบนคือ เซตบิต $M_1 = 0$ และ $M_0 = 1$ ซึ่งเป็นการเลือกโหมด 1 และให้ C / T= 0 และ GATE = 0 ซึ่งเป็นการใช้สัญญาณนาฬิกาจากภายในหรือใช้เป็น Timer และตัว Timer นี้จะยังไม่ทำงาน ถ้าบิตควบคุม TR_1 ยังไม่ได้เซต

ถ้าให้ Timer นี้นับขึ้นโดยใช้รีจีสเตอร์ TL1 /TH1 และจะเซตบิต Overflow Flag เมื่อรีจีสเตอร์เปลี่ยนจาก FFFFH เป็น 0000H โดยให้นับเวลาไป 100 μS หรือให้ TL1/TH1 นับสัญญาณนาฬิกาได้ 100 ลูก ดังนั้นค่าเริ่มต้นของ TL1 / TH1 จะไม่เริ่มที่ 0000H จะต้องเริ่มที่ FFFFH ลบด้วย 100 ลูก หรือ FF9CH เพื่อให้นับไปถึง FFFFH และเปลี่ยนเป็น 0000H ได้สัญญาณนาฬิกา 100 ลูกพอดี สามารถเขียนคำสั่งได้ดังนี้

MOV TL1 , # 9CH

MOV TH1 , # 0FFH

ถ้าให้ Timer เริ่มทำงานก็ให้บิตควบคุมดังนี้

SETB TR1

จากนั้นบิต Overflow Flag จะส่งออกมาหลังเวลาผ่านไป 100 μS ซึ่งเราสามารถเขียนโปรแกรมเป็นโปรแกรมวนลูป 100 μS ได้ โดยตรวจสอบบิต TF1 ว่าถูกเซตหรือไม่ ถ้าไม่เซตก็ให้วนลูปต่อไปดังนี้

CLR TR1

CLR TF1

การใช้แบบ Readind a Timer "On the Fly"

การใช้งานแบบประยุกต์บางงานจะต้องอ่านค่าจาก Timer Register เนื่องจากตัว Timer Register มีขนาด 2 ไบต์ ถ้าหากไบต์ต่ำเกิด Overflow จะทดเข้าไบต์สูง ถ้าหากเขียนโปรแกรมให้อ่านค่าจากไบต์ต่ำก่อน แล้วจึงอ่านไบต์สูงข้อมูลที่ได้อาจเกิดข้อผิดพลาดได้เนื่องจากไบต์ต่ำมีการเปลี่ยนแปลงเร็วกว่าไบต์สูง การอ่านข้อมูลควรอ่านจากไบต์สูงก่อน แล้วจึงกลับมาอ่านไบต์ต่ำ จากนั้นอ่านข้อมูลไบต์สูงอีกครั้ง ถ้าค่าไบต์สูงที่อ่านได้ไม่มีการเปลี่ยนแปลงให้ใช้ค่านั้นได้เลย

แต่ถ้ามีการเปลี่ยนแปลงให้อ่านอีกครั้ง ถ้าต้องการอ่านข้อมูลจาก TL1 / TH1 เข้าในรีจีสเตอร์ R6 / R7 อาจเขียนโปรแกรมได้ดังนี้

```
AGAIN :  MOV A, TH1
MOV R6, TL1
CJNE A, TH1, AGAIN
MOV R7, A
```

2.1.3.6 Short Intervals And Long Intervals

ถ้า MCS – 51 ทำงานที่ความถี่สัญญาณนาฬิกา 12 MHz ถ้าให้ Timers ใช้วงจร Oscillator บนชิพ สัญญาณนาฬิกาจะถูกหารด้วย 12 และ Timer จะทำงานด้วยความถี่ 1 MHz ถ้าต้องการใช้โปรแกรมสร้างสัญญาณนาฬิกาออกมาอาจทำได้โดยง่าย ซึ่งพิจารณาจากการทำงานชุดคำสั่งต่าง ๆ ของ MCS – 51 ใน 1 Machine Cycle จะใช้เวลา 1μS ในตารางที่ 2.6 จะแสดงความกว้างของสัญญาณที่สร้างขึ้นจาก MCS – 51 ที่ทำงานด้วย Crystal ความถี่ 12 MHz

ตารางที่ 2.6 ค่าสูงสุดของการใช้ Timer โหมดต่าง ๆ

Maximum Interval in Microseconps	**Technique**
≈10	Software Tuing
256	8 – bit Timer with Auto-reload
65536	8 – bit Timer
No Limit	16 – bit Timer Plus Software Loops

2.1.4 การอินเตอร์รัพท์

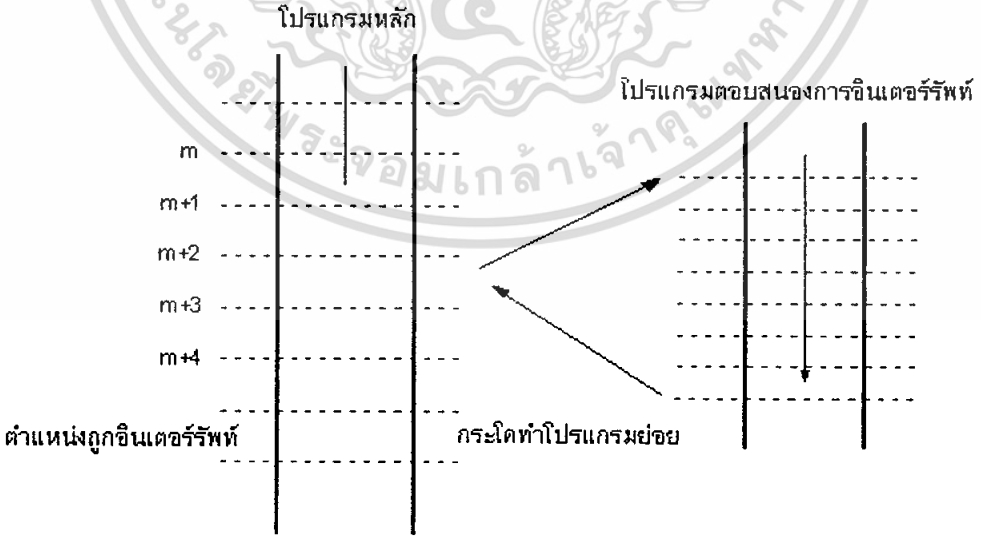
การทำงานของระบบคอมพิวเตอร์โดยทั่วไปมักมีอุปกรณ์ภายนอกต่อร่วมอยู่ ถ้าคอมพิวเตอร์ต้องการทำงานกับอุปกรณ์ภายนอกจะต้องคอยตรวจสอบอุปกรณ์เหล่านั้นเสมอ ตัวอย่างเช่น ถ้าหากให้คอมพิวเตอร์พอร์ทหนึ่งต่ออยู่กับหลอด LED 7 ส่วน อีกพอร์ทหนึ่งต่อกับสวิตช์ ถ้าระบบของเราทำงานเป็นนาฬิกาเดินไปให้คอยตรวจสอบสวิตช์ด้วยว่ามีการกดหรือยัง การทำงานแบบนี้เรียกว่า Polling Method คือ ตัวไมโครโปรเซสเซอร์จะต้องคอยตรวจสอบอุปกรณ์อินพุตตลอดเวลาว่ามีข้อมูลเข้ามาหรือยัง การทำงานแบบนี้ถ้ามีอุปกรณ์ภายนอกหลายตัวระบบต้องตรวจสอบอุปกรณ์ภายนอกหลายตัว ทำให้เสียเวลาในการทำงานหลักไป การทำงานอีกแบบหนึ่งจะให้ CPU ทำงานหลัก ถ้ามีการกดสวิตช์เมื่อไรให้นาฬิกาหยุดเดินทันที การทำงาน

ในลักษณะนี้ CPU ไม่ต้องเสียเวลาในการตรวจอุปกรณ์ภายนอก ถ้าอุปกรณ์ภายนอกต้องการติดต่อกับ CPU อุปกรณ์ภายนอกจะส่งสัญญาณมาบอก CPU เอง ระบบนี้เรียกว่า การอินเทอร์รัพท์ (Interrupt)

2.1.4.1 ขบวนการเกิดอินเตอร์รัพท์

ถ้าหากคอมพิวเตอร์กำลังทำงานโปรแกรมหลักอยู่ เมื่อมีการอินเทอร์รัพท์เข้ามา คอมพิวเตอร์จะละทิ้งโปรแกรมหลัก แต่ไปทำงานโปรแกรมตอบสนองการอินเตอร์รัพท์ (Interrupt Service Routine) เมื่อทำโปรแกรมตอบสนองอินเตอร์รัพท์เสร็จ คอมพิวเตอร์จะกลับมาทำโปรแกรมเดิม พิจารณารูปที่ 2.11

ถ้า CPU กำลังทำงานโปรแกรมหลักอยู่ เช่นกำลังทำคำสั่งในตำแหน่งของหน่วยความจำที่ m, m+1, m+2 ไปเรื่อย ๆ โดย PC จะชี้ที่ตำแหน่งที่จะอ่านค่าคำสั่งถัดมา เมื่อโปรแกรมทำงานมาถึงตำแหน่งที่ m+3 แล้วเกิดการอินเตอร์รัพท์ขึ้น (ขณะนั้น PC อยู่ที่ m+4) โปรแกรมจะต้องทำงานโปรแกรมตอบสนองการอินเตอร์รัพท์ โดยย้าย PC ไปที่ตำแหน่งที่เก็บโปรแกรมตอบสนองการอินเตอร์รัพท์ จากนั้นจะเก็บค่า PC เดิมลงในหน่วยความจำสแตค เมื่อคอมพิวเตอร์ทำงานโปรแกรมตอบสนองการอินเตอร์รัพท์เสร็จสิ้นลง จะคืนค่าใน สแตค (m+4) ให้กับ PC ทำโปรแกรมหลักต่อไป



รูปที่ 2.12 ขั้นตอนการทำงานของโปรแกรมเมื่อถูกอินเตอร์รัพท์

2.1.4.2 สัญญาณอินเตอร์รัพท์

แหล่งกำเนิดสัญญาณอินเตอร์รัพท์ที่ใช้กับ MCS – 51 มีสองชนิดคือ อินเตอร์รัพท์ภายในและภายนอก โดยอินเทอร์รัพท์ภายในจะเกิดขึ้นจากภายในตัว MCS – 51 เอง ได้แก่ สัญญาณจากไทเมอร์แฟลก 0 (TF0) ไทเมอร์แฟลก 1 (TF1) และพอร์ทอนุกรม สำหรับอินเตอร์รัพท์ภายนอกเกิดจากสัญญาณที่กระตุ้นเข้ามาทางขา INT0 และ INT1 เมื่อมีสัญญาณอินเตอร์รัพท์จากแหล่งต่างๆ เข้ามา เราสามารถโปรแกรมได้ว่าจะให้ MCS – 51 ยอมให้มีการอินเตอร์รัพท์ได้หรือไม่ โดยการโปรแกรมไปที่ รีจีสเตอร์ IE (Interrupt Enable) และถ้ามีสัญญาณอินเตอร์รัพท์มากจากแหล่งต่าง ๆ หลายแหล่งพร้อมกันเราสามารถจัดลำดับได้ว่า จะให้อินเตอร์รัพท์ใดเกิดก่อน โดยการโปรแกรมไปที่ อินเตอร์รัพท์ไพออริตี้ IP (Interrupt Priority) รีจีสเตอร์ทั้งสองตัวมีรายละเอียดดังนี้

1. Interrupt Enables

เป็นรีจีสเตอร์ที่สามารถเข้าถึงข้อมูลระดับบิตได้ ใช้สำหรับกำหนดค่าว่าถ้าเกิดการอินเตอร์รัพท์จากแหล่งต่าง ๆ จะทำอินเตอร์รัพท์เหล่านั้นหรือไม่ โดยรายละเอียดของบิตต่าง ๆ มีดังตารางที่ 2.7

ตารางที่ 2.7 บิตต่าง ๆ ของรีจีสเตอร์ IE

บิต	ชื่อบิต	ตำแหน่งบิต	รายละเอียด
IE.7	EA	AFH	ถ้าเซตยอมให้มีการอินเตอร์รัพท์
IE.6	-	AEH	ไม่ใช้งาน
IE.5	ET2	ADH	Enable อินเตอร์รัพท์จาก Timer 2 (ใช้กับ 8052)
IE.4	ES	ACH	Enable อินเตอร์รัพท์จากพอร์ทอนุกรม
IE.3	ET1	ABH	Enable อินเตอร์รัพท์จาก Timer 1
IE.2	EX1	AAH	Enable อินเตอร์รัพท์จาก INT1
IE.1	ET0	A9H	Enable อินเตอร์รัพท์จาก Timer 0
IE.0	EX0	A8H	Enable อินเตอร์รัพท์จาก INT0

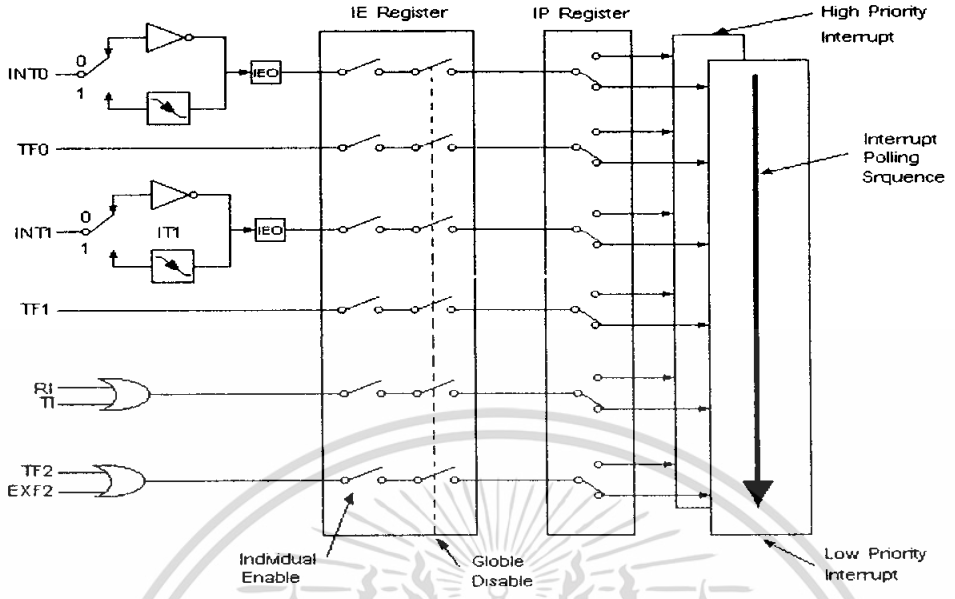
2. Interrupt Priority

เป็นรีจีสเตอร์ที่สามารถเข้าถึงข้อมูลระดับบิตได้ใช้ในการจัดลำดับความสำคัญของการอินเทอร์รัพท์ซึ่งสามารถจัดได้สองลำดับ ถ้าเป็น "1" หมายความว่ามีความสำคัญสูงสุด ถ้าเป็น "0" หมายความว่ามีลำดับความสำคัญต่ำสุด ความหมายของบิตต่าง ๆ แสดงได้ดังตารางที่ 8 ถ้าหากกำหนดให้มีความสำคัญเป็น "1" เหมือนกันหมด MCS – 51 จะจัดลำดับความสำคัญใหม่ดังนี้

ลำดับ	อินเตอร์รัพท์
1 (สูงสุด)	IE0
2	TF0
3	IE1
4	TF1
5 (ต่ำสุด)	Serial Port

ตารางที่ 2.8 บิตและหน้าที่ต่าง ๆ ของรีจีสเตอร์ IP

บิต	ชื่อบิต	ตำแหน่งบิต	รายละเอียด
IP.7	-	-	ไม่ใช้งาน
IP.6	-	-	ไม่ใช้งาน
IP.5	PT2	0BDH	ใช้กับ Timer 2 (8052)
IP.4	PS	0BCH	ใช้กับพอร์ทอนุกรม
IP.3	PT1	0BBH	ใช้กับ Timer 1
IP.2	PX1	0BAH	ใช้กับอินเทอร์รัพท์จาก INT1
IP.1	PT0	0B9H	ใช้กับ Timer 0
บิต	ชื่อบิต	ตำแหน่งบิต	รายละเอียด
IP.0	PX0	0B8H	ใช้กับอินเทอร์รัพท์จาก INT0



รูปที่ 2.13 รีจิสเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการอินเตอร์รัพท์

จากรูปที่ 2.12 แสดงการอินเตอร์รัพท์จากแหล่งต่าง ๆ ที่มีผลกับ MCS – 51 ถ้าเป็นเบอร์ 8051 8031 จะถูกอินเตอร์รัพท์ได้ 5 แหล่ง ถ้าเป็นเบอร์ 8052,8032 จะถูกอินเตอร์รัพท์ได้ 6 แหล่ง โดยเพิ่มอินเตอร์รัพท์จาก Timer 2 ในรูปที่ 2.17 จะแสดงให้เห็นว่า ถ้า MCS – 51 จะถูกอินเตอร์รัพท์ได้จะต้องเซตค่า Global Enable ในรีจีสเตอร์ IE นอกจากนี้ยังกำหนดได้ว่าจะให้อินเตอร์รัพท์ใดเกิดได้ โดยการเซตค่า Interrupt Enable ของอินเตอร์รัพท์จากแหล่งต่าง ๆ ในรีจีสเตอร์ IE จากรูปยังแสดงให้เห็นอีกว่าเมื่อมีการอินเทอร์รัพท์เข้ามาจะมีผลต่อแฟลกใด เช่น ถ้า INT0 เป็น "1" บิต IE0 จะเป็น "1" หมายความว่าถูกอินเตอร์รัพท์ โดยแฟลกต่าง ๆที่มีผลจากการถูกอินเตอร์รัพท์แสดงได้ดังตารางที่ 2.9

ตารางที่ 2.9 แฟลกที่จะทำงานเมื่อถูกอินเตอร์รัพท์

อินเตอร์รัพท์	แฟลก	ประกอบอยู่ในรีจีสเตอร์
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 1	TF1	TCON.7
Timer 0	TF0	TCON.5
Serial port	T1	SCON.1
Serial port	RI	SCON.0
Timer 2	TF2	T2CON.7 (8052)
Timer 2	EXF2	T2CON.6 (8052)

จากตารางจะเห็นว่า ถ้ามีการอินเตอร์รัพท์จากภายนอกเข้ามา ตัวที่จะอินเตอร์รัพท์ MCS – 51 คือ บิตแฟลก IE0 ซึ่งอยู่ในรีจีสเตอร์ TCON ถ้ามีการสื่อสารแบบอนุกรม เมื่อข้อมูลถูกส่งไปหมดแล้วจะอินเตอร์รัพท์ MCS –51 ทางบิตแฟลก TI ถ้ารับข้อมูลหมดแล้วจะอินเตอร์รัพท์ MCS – 51 ทางบิตแฟลก RI ซึ่งอยู่ในรีจีสเตอร์ SCON และถ้าใช้ Timer 0 ในการนับเมื่อเกิด Overflow สามารถอินเตอร์รัพท์ MCS – 51 ได้ทางบิต TF0

2.1.4.3 การทำงานของระบบหลังถูกอินเตอร์รัพท์

เมื่อ MCS – 51 ถูกอินเทอร์รัพท์จะต้องกระโดดไปทำโปรแกรมตอบสนองการอินเทอร์รัพท์โดยตำแหน่งที่จะกระโดดไปเรียกว่า อินเตอร์รัพท์เวกเตอร์ (Interrupt Vectors) เมื่อทำโปรแกรมตอบสนองการอินเตอร์รัพท์เรียบร้อยแล้ว MCS – 51 จะกระโดดมาทำงานยังตำแหน่งเดิม โดยก่อนที่จะกระโดดไปทำโปรแกรมตอบสนองการอินเทอร์รัพท์จะต้องเก็บค่าตำแหน่งเดิมไว้ โดยเก็บค่า PC ลงหน่วยความจำสแตคซึ่งอยู่ที่หน่วยความจำที่ถูกชี้โดยรีจีสเตอร์ SP เมื่อทำโปรแกรมตอบสนองการอินเทอร์รัพท์เสร็จแล้วจะคืนค่าในหน่วยความจำสแตคให้ PC ตามเดิม ค่าอินเทอร์รัพท์เวกเตอร์ของ MCS – 51 แสดงได้ดังตารางที่ 2.10

ตารางที่ 2.10 อินเตอร์รัพท์เวกเตอร์ของอินเตอร์รัพท์ต่าง ๆ

อินเทอร์รัพท์	อินเทอร์รัพท์เวกเตอร์
System Reset	0000H
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial Port	0023H
Timer 2	002BH

จากตารางจะเห็นว่าถ้าระบบถูกอินเตอร์รัพท์จากภายนอกทาง INT0 ตัว MCS – 51 จะกระโดดไปทำงานที่ตำแหน่ง 0003H ถ้าระบบถูกอินเตอร์รัพท์จาก Timer 0 จะกระโดดไปทำงานตำแหน่ง 000BH

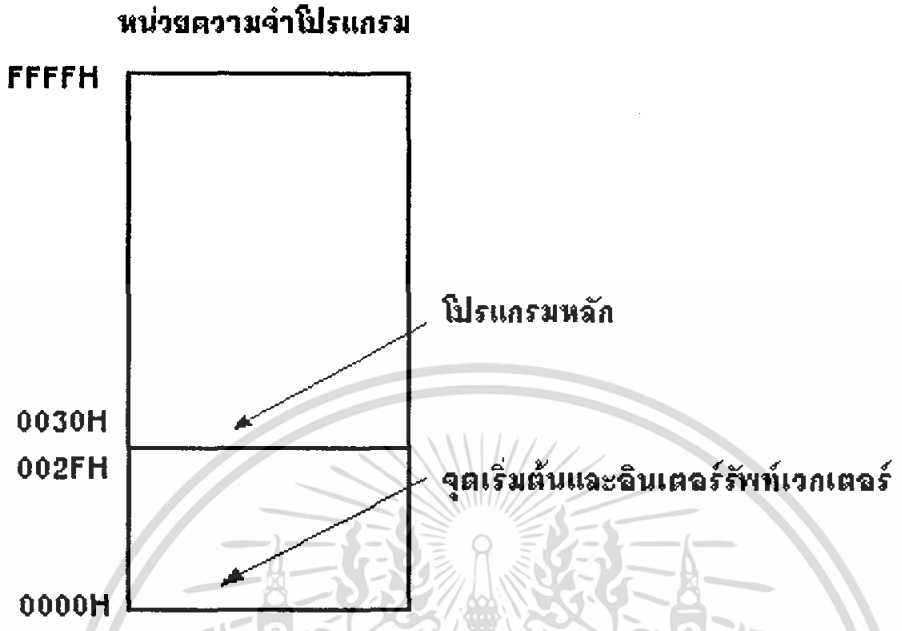
2.1.4.4 การออกแบบโปรแกรมอินเตอร์รัพท์

ในการเขียนโปรแกรมหลัก (Main Program) จะต้องกำหนดค่าว่าจะให้ MCS – 51 ถูกอินเตอร์รัพท์ด้วยอะไร และจะให้ MCS – 51 ถูกอินเตอร์รัพท์ได้หรอไม่ โดยการโปรแกรมค่าต่าง ๆ ใน IE รีจีสเตอร์ ถ้ามีการอินเทอร์รัพท์จากสองแหล่งขึ้นไปควรมีการจัดลำดับความสำคัญในรีจีสเตอร์ IP ดังนั้นในโปรแกรมหลักจะต้องมีการโปรแกรมต่อไปนี้

1. โปรแกรมค่าในรีจีสเตอร์ IE

สำหรับโปรแกรมตอบสนองการอินเทอร์รัพท์ถือว่าเป็นโปรแกรมย่อยโปรแกรมหนึ่ง แต่จะต้องจบโปรแกรมย่อยด้วยค่า RETI (Return From Interrupt) จากตารางอินเตอร์รัพท์เวกเตอร์จะเห็นว่าถ้ากด Reset หรือให้ระบบเริ่มทำงาน โปรแกรมจะเริ่มทำงานที่ตำแหน่ง 0000H และจะเห็นว่า ตำแหน่งที่เก็บโปรแกรมหลักมีโอกาสอย่างมากที่จะทับกับหน่วยความจำโปรแกรมที่เก็บค่าอินเทอร์รัพท์เวกเตอร์ที่ตำแหน่ง 0003H ถ้าโปรแกรมยาวมากอาจจะไปทับตำแหน่ง 000BH ได้ซึ่งเป็นตำแหน่งของอินเทอร์รัพท์เวกเตอร์ของ Timer 0 ดังนั้นในการเขียนโปรแกรมหลัก ภายใน 3 ตำแหน่งแรก คือ 0000H, 0001H, 0002H จะต้องกระโดดไปที่อื่นก่อนเพื่อให้ข้ามอินเทอร์รัพท์เวกเตอร์ไป ซึ่งอาจเขียนโปรแกรมได้ดังนี้

```
ORG    0000H          ; เริ่มต้นโปรแกรม
LJMP   MAIN           ; กระโดดไปโปรแกรมหลัก
.......               ; เพื่อหนีอินเทอร์รัพท์เวกเตอร์
.......
ORG    0030H          ; ตำแหน่งเริ่มต้นของโปรแกรม
MAIN : .......        ; เริ่มต้นโปรแกรมหลัก
```



รูปที่ 2.14 การจัดตำแหน่งโปรแกรมในหน่วยความจำ

จากตัวอย่างโปรแกรมจะเห็นว่า เมื่อเริ่มต้นโปรแกรมหรือระบบถูกรีเซต ระบบจะทำงานตำแหน่งแรก คือ คำสั่งกระโดดไปโปรแกรมหลัก ซึ่งอยู่ต่อจากโปรแกรมตอบสนองการอินเตอร์-รัพท์ที่อยู่ที่ตำแหน่ง 0030H

โปรแกรมตอบสนองการอินเตอร์รัพท์แบบสั้น

จากตารางอินเตอร์รัพท์เวกเตอร์ จะเห็นว่าที่เก็บโปรแกรมอินเตอร์รัพท์แต่ละแหล่งจะห่างกัน 8 ไบต์ ดังนั้นถ้ามีการอินเตอร์รัพท์จากแหล่งต่าง ๆ หลาย ๆ แหล่ง และโปรแกรมตอบสนองการอินเตอร์รัพท์บางโปรแกรมมีขนาดยาวเกิน 8 ไบต์ จะทำให้โปรแกรมไปทับกับตำแหน่งของโปรแกรมตอบสนองการอินเทอร์รัพท์ของอินเทอร์รัพท์ถัดไป แต่ถ้าโปรแกรมตอบสนองการอินเทอร์รัพท์ไม่ยาวมากเกินไป เราสามารถเขียนไปในตำแหน่งนั้นได้เลยดังโปรแกรมต่อไปนี้

```
ORG       0000H
LJMP      MAIN        ; กระโดดไปโปรแกรมหลัก
ORG       000BH       ; ตำแหน่งเริ่มต้นของอินเตอร์รัพท์  Timer 0
TOISR :.....
      .....
```

```
RETI                    ; กลับโปรแกรมหลัก
MAIN : ……              ; โปรแกรมหลัก
……
```

จากตัวอย่างโปรแกรมจะใช้อินเตอร์รัพท์จาก Timer 0 เมื่อระบบเริ่มทำงานจะทำตำแหน่ง 0000H โดยกระโดดไปโปรแกรมหลัก ซึ่งอยู่ที่ตำแหน่งต่อจากโปรแกรมตอบสนองการอินเทอร์รัพท์เมื่อมีการอินเทอร์รัพท์ Timer 0 ระบบจะทำโปรแกรมตำแหน่งที่ 000BH ซึ่งเป็นอินเตอร์รัพท์-เวกเตอร์ของ Timer 0 โดยโปรแกรมตอบสนองการอินเตอร์รัพท์จะจบด้วยคำสั่ง RETI เพื่อกลับสู่โปรแกรมหลักต่อไป

โปรแกรมตอบสนองการอินเตอร์รัพท์ขนาดใหญ่

ในกรณีที่มีการอินเตอร์รัพท์จากหลายแหล่ง และโปรแกรมตอบสนองการอินเตอร์รัพท์แต่ละโปรแกรมยาวเกิน 8 ไบต์ เราไม่สามารถเขียนโปรแกรมตอบสนองการอินเทอร์รัพท์ไว้ที่ตำแหน่งของอินเทอร์รัพท์เวกเตอร์ได้ ซึ่งจะแก้ปัญหานี้ได้โดยกำหนดให้ตำแหน่งของอินเทอร์รัพท์เวกเตอร์ให้ทำโปรแกรมกระโดด โดยกระโดดไปที่ตำแหน่งเก็บโปรแกรมตอบสนองการอินเทอร์รัพท์ที่เขียนไว้ที่ตำแหน่งอื่นดังตัวอย่าง ต่อไปนี้

```
ORG   0000H       ; เริ่มโปรแกรมของระบบ
LJMP  MAIN        ; กระโดดไปโปรแกรมหลัก
ORG   000BH       ; ตำแหน่งของอินเตอร์รัพท์ Timer 0
LJMP  LED1        ; กระโดดไปโปรแกรมตอบสนองการอินเตอร์รัพท์ชื่อ LED1
ORG   0030H       ; ตำแหน่งหลังอินเตอร์รัพท์เวกเตอร์
MAIN : ……        ; โปรแกรมหลัก
……
LED1 : ……        ; โปรแกรมตอบสนองการอินเตอร์รัพท์ Timer 1
……
RETI              ; กลับสู่โปรแกรมหลัก
```

จากโปรแกรมจะเห็นว่า เมื่อระบบทำงาน จะต้องทำที่ตำแหน่ง 0000H โดยกระโดดไปทำโปรแกรมหลักที่ตำแหน่งต่อจาก 0030H เพราะตำแหน่งดังกล่าวข้ามอินเตอร์รัพท์เวกเตอร์จากแหล่งต่าง ๆ ไปแล้ว เมื่อมีการอินเตอร์รัพท์จาก Timer 0 โปรแกรมจะต้องทำงานที่ตำแหน่ง 000BH แต่โปรแกรมตอบสนองการอินเตอร์รัพท์ยาวมาก ที่ตำแหน่ง 000BH จึงให้ทำโปรแกรมกระโดด โดยกระโดดไปที่โปรแกรมตอบสนองการอินเตอร์รัพท์ชื่อ LED1 ซึ่งอยู่ท้ายโปรแกรม เมื่อจบโปรแกรมจะจบด้วยคำสั่ง RETI เพื่อกลับไปโปรแกรมหลักต่อไป

2.2 ภาพรวมของ USB

2.2.1 ความหมายของพอร์ตในระบบ USB

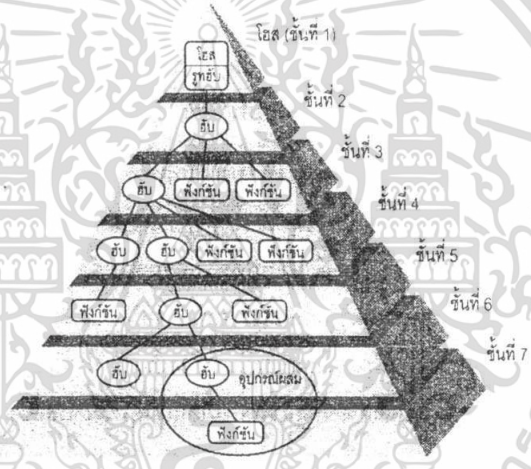
พอร์ต (port) ในระบบ USB จะมีความหมายแตกต่างจากพอร์ตอนุกรมและขนานซึ่งมีอยู่ก่อนหน้านี้ ตามความเข้าใจโดยทั่วไปนั้นพอร์ตคอมพิวเตอร์จะเป็นตำแหน่งที่มีไว้สำหรับนำวงจรอิเล็กทรอนิคส์ภายนอกมาต่อเข้าไป ซึ่งปกติวงจรอิเล็กทรอนิคส์จะมีจุดเชื่อมต่อเป็นคอนเน็คเตอร์ สำหรับเสียบสายเคเบิลเข้ากับอุปกรณ์อิเล็กทรอนิคส์รอบข้าง เช่น เมาส์, คีย์บอร์ด, จอคอมพิวเตอร์ หรือพรินเตอร์ ความจำของคอมพิวเตอร์เองก็จะมีแอดเดรสเช่นกันแต่ CPU จะเข้าถึงหน่วยความจำด้วยคำสั่งที่แตกต่างกันออกไป ซึ่งส่วนมากแล้วแอดเดรสของหน่วยความจำจะถูกต่อเข้ากับบัสข้อมูลของระบบเอง

สำหรับการเชื่อมต่อแบบอนุกรมนั้น แต่ละพอตร์จะเป็นอิสระต่อกัน เช่น ถ้าเรามีพอร์ตอนุกรม 2 พอร์ต แต่ละพอร์ตก็จะมีเส้นทางข้อมูลของตัวมันเอง และสายเคเบิลแต่ละเส้นจะมีเฉพาะข้อมูลที่ใช้กับพอร์ตของมันเองเท่านั้น ซึ่งการรับส่งข้อมูลทั้งสองนี้จะสามารถทำได้อย่างอิสระต่อกัน

ส่วนระบบบัส USB จะใช้วิธีการที่แตกต่างออกไป โดยมีโฮสคอนโทรเลอร์เป็นตัวควบคุมการสื่อสารข้อมูลบนระบบบัสทั้งหมด และโฮสคอนโทรเลอร์นี้จะสนับสนุนบัสหรือเส้นทางข้อมูลเพียงหนึ่งเส้นทางเท่านั้น คอนเน็คเตอร์แต่ละตัวที่อยู่บนระบบบัสจะทำหน้าที่เป็นพอร์ต USB โดยอุปกรณ์ทุกตัวจะแบ่งการใช้เวลาบนบัสกัน ถึงแม้ว่าจะมีพอร์ตหลายพอร์ตอยู่ในระบบบัส และแต่ละพอร์ตก็มีคอนเน็คเตอร์และสายเคเบิลเป็นของมันเอง แต่ในระบบบัส USB จะมีเส้นทางข้อมูลที่แท้จริงเพียงเส้นทางเดียวเท่านั้น ซึ่งในเวลาหนึ่ง ๆ จะมีโฮสหรืออุปกรณ์เพียงตัวเดียวเท่านั้นที่เป็นตัวส่งข้อมูล ซึ่งโฮสหนึ่งตัวอาจจะสนับสนุนโฮสคอนโทรเลอร์ได้หลายตัว อย่างไรก็ตามโฮสคอนโทรเลอร์แต่ละตัวก็จะมีบัสเป็นของมันเอง

2.2.2 โทโพโลยีของระบบบัส (Bus topology)

โทโพโลยี หรือการจัดการเชื่อมต่อบนระบบบัส USB จะเป็นแบบไทร์สตาร์(Tiered Star) หรือสตาร์แบบลำดับชั้นตามรูปที่ 2.15 โครงสร้างนี้ประกอบด้วยโครงสร้างแบบสตาร์หลาย ๆ ชุด เชื่อมต่อกันอยู่ โดยมีฮับเป็นจุดศูนย์กลางของโครงสร้างย่อย ๆ เหล่านั้นที่ส่วนบนสุดของสตาร์จะเป็นโฮสคอนโทรเลอร์ ซึ่งในระบบบัส USB จะมีโฮสเพียงตัวเดียวเท่านั้นที่ควบคุมการติดต่อสื่อสารทั้งหมดของระบบ และมีฮับเป็นตัวเพิ่มจำนวนพอร์ตที่ใช้เชื่อมต่ออุปกรณ์ได้มากขึ้น จุดปลายแต่ละจุดของโครงสร้างสตาร์จะเป็นอุปกรณ์เพื่อเพิ่มจำนวนพอร์ตให้มากยิ่งขึ้นก็ได้ ซึ่งความสามารถใช้ในการเชื่อมต่อฮับเข้าด้วยกันจึงทำให้เกิดโครงสร้างย่อยได้หลายชุดและมีลักษณะเป็นลำดับชั้น



รูปที่ 2.15 โทโพโลยีของระบบบัส USB

สตาร์แบบลำดับชั้นเป็นเพียงการอธิบายการเชื่อมต่อทางกายภาพเท่านั้น แต่ในความเป็นจริงนั้นระบบบัส USB จะมีเส้นทางข้อมูลสำหรับการสื่อสารเพียงหนึ่งเส้นทาง และโฮสกับอุปกรณ์ทุกตัวที่ต่ออยู่ในระบบจะส่งถ่ายข้อมูลโดยอาศัยเส้นทางดังกล่าวเท่านั้น

เนื่องจากตามปกติจะมีการหน่วงเวลาของสัญญาณเกิดขึ้นภายในสายเคเบิลและฮับ ดังนั้นจำนวนชั้นสูงสุดที่ยอมให้ใช้ได้ภายในโครงสร้างนี้จะมีทั้งหมด 7ชั้น (รวมชั้นทีเป็นรูทฮับด้วย) ภายในเจ็ดชั้นนี้จะมีฮับได้ทั้งหมดไม่เกิน 5 ตัวและในชันที่ 7 จะต้องเป็นฟังก์ชั่นเท่านั้น ในกรณีของอุปกรณ์ผสม (Compound Device) นั้นด้วยโครงสร้างซึ่งมีฮับอยู่ภายในตัวมัน ทำให้อุปกรณ์นี้ครอบครองโครงสร้างซึ่งมีฮับอยู่ฮับอยู่ภายในตัวมัน ทำให้อุปกรณ์แบบนี้ครอบครองโครงสร้างไว้ 2 ชั้น ดังนั้นอุปกรณ์แบบผสมไม่สามารถนำมาต่อในชั้นที่ 7 ได้

โฮส USB โฮส คือ เครื่องคอมพิวเตอร์ซึ่งประกอบไปด้วยโฮสคอนโทรเลอร์ และรูทฮับอยู่ภายใน ซึ่งทั้งสองตัวนี้จะทำงานร่วมกันเพื่อให้ระบบปฏิบัติการสามารถสื่อสารกับอุปกรณ์บนบัสได้ โฮสคอนโทรลเลอร์จะเป็นตัวจัดรูปแบบข้อมูลสำหรับส่งไปบนบัสและแปลข้อมูลที่รับเข้ามาให้อยู่ในรูปของที่ระบบปฏิบัติการสามารถเข้าใจได้ นอกจากนี้โฮสคอนโทรเลอร์ยังทำงานเกี่ยวกับฟังก์ชั่นอื่น ที่เกี่ยวข้องกับการจัดการงานทางด้านการสื่อสารบนบัสอีกด้วย ส่วนรูทฮับจะเป็นฮับที่ต่อกับโฮสคอนโทรลเลอร์โดยตรง และมีคอนเน็คเตอร์สำหรับใช้ต่อกับอุปกรณ์ USB อีกอย่างน้อยหนึ่งตัว

ตามปกติ USB โฮส จะต้องทราบว่ามีอุปกรณ์อะไรบ้างที่ต่ออยู่บนบัส และแต่ละตัวมีความสามารถในการทำงานอะไรบ้าง นอกจากนี้การทำงานของโฮสจะต้องมีเสถียรภาพมากที่สุดเพื่อให้มั่นใจได้ว่าอุปกรณ์ทั้งหมดที่ต่ออยู่บนระบบบัสสามารถส่งและรับข้อมูล เมื่ออุปกรณ์เหล่านั้นต้องการได้สำหรับหน้าที่ต่าง ๆ ของโฮสมีดังนี้

- **ตรวจจับอุปกรณ์ที่ถูกเพิ่มเข้ามาในระบบ** เมื่อทำการจ่ายไฟเข้ากับระบบ ฮับจะทำการแจ้งให้โฮสทราบถึงรายการอุปกรณ์ USB ทั้งหมดที่ถูกต่อเข้ากับระบบ โดยกระบวนการที่เรียกว่า “อินิวเมอร์เรท” (Enumerate) ซึ่งโฮสจะทำการกำหนดแอสเดรสและร้องขอข้อมูลเพิ่มเติมจากอุปกรณ์แต่ละตัว เมื่ออุปกรณ์ถูกถอดออกหรือ ต่อเข้ากับระบบภายหลังจากที่มีการจ่ายไฟให้แก่ระบบแล้ว โฮสจะสามารถรับรู้ได้และทำการอินิวเมอร์เรท อุปกรณ์ใหม่ที่ถูกต่อเข้าไป หรือทำการลบอุปกรณ์ตัวที่ถูกถอดออกไปจากระบบ
- **การจัดการการไหลของข้อมูล** โฮสจะเป็นตัวจัดการไหลของข้อมูลที่อยู่ภายในระบบบัส ซึ่งในบางครั้งอุปกรณ์รอบข้างหลายตัวอาจต้องการการส่งถ่ายข้อมูลภายในเวลาเดียวกัน ดั้งนั้นโฮสคอนโทรลเลอร์จะต้องจัดการให้อุปกรณ์ทุกตัวสามารถส่งถ่ายข้อมูลได้ โดยการแบ่งเวลาออกเป็นส่วน ๆ เรียกว่า “เฟรม หรือไมโครเฟรม” และทำการแบ่งข้อมูลแล้วส่งออกส่วนของเฟรมหรือไมโครเฟรม
- **การตรวจสอบความผิดพลาดของข้อมูล** อีกหน้าหนึ่งซึ่งสำคัญของโฮส คือ การตรวจสอบความผิดพลาดของข้อมูล โดยโฮสจะทำการใส่บิตตรวจสอบความผิดพลาดลงในข้อมูลที่ต้องการส่ง เมื่ออุปกรณ์ได้รับข้อมูลแล้วตัวอุปกรณ์จะนำข้อมูลที่ได้รับมาทำการคำนวณแล้วเปรียบเทียบผลที่ได้ของบิตทดสอบความผิดพลาด ถ้าผลที่ได้ไม่ตรงกันอุปกรณ์ก็จะไม่รับข้อมูลนั้นและโฮส จะทราบได้ว่ามันต้องทำการส่งใหม่อีกครั้ง (ระบบบัส USB ยังสนับสนุนการส่งถ่ายข้อมูลชนิดที่ไม่อนุญาติให้มีการส่งซ้ำทั้งนี้เพื่อรักษาอัตราการส่งถ่ายข้อมูลให้มีค่าคงที่ตลอดเวลา) และด้วยวิธีการเดียวกันนี้โฮสจะทำการตรวจสอบข้อมูลที่รับเข้ามาจากอุปกรณ์เช่นกัน

- **จ่ายกำลังงานให้อุปกรณ์** โฮสสามารถจ่ายกำลังงานให้แก่อุปกรณ์แบบที่ให้กำลังงานจากระบบบัส (Bus-Powered Device) ได้โดยผ่านสายเคเบิล USB แต่ในอุปกรณ์บางตัวอาจให้กำลังงานจากบัสเพียงแค่ในขณะทำการเริ่มต้นสื่อสารกับโฮสเท่านั้น
- **แลกเปลี่ยนข้อมูลกับอุปกรณ์รอบข้าง** หน้าที่ทั้งหมดที่กล่าวมานี้เป็นงานที่สนับสนุนงานหลักของโฮส ซึ่งหน้าที่หลักจริง ๆ ของโฮส คือ การแลกเปลี่ยนข้อมูลกับอุปกรณ์รอบข้าง โดยโฮสจะทำงานร่วมกับดีไวซ์ไดร์ฟเวอร์ในการรับส่งข้อมูล

2.2.3 อุปกรณ์ USB

อุปกรณ์ USB หรืออุปกรณ์รอบข้าง USB เป็นสิ่งที่เรานำเข้ามาต่อกับพอร์ต USB บนเครื่องคอมพิวเตอร์หรือฮับ โดยคำจำกัดความที่เป็นทางการของอุปกรณ์ คือ ฟังก์ชัน ซึ่งเป็นตัวเพิ่มความสามารถในการทำงานต่าง ๆ ให้แก่โฮส หรืออาจเป็นฮับก็ได้

อุปกรณ์ USB แต่ละตัวจะต้องมีวงจรอิเล็กทรอนิคส์และโค้ดโปรแกรมซึ่งให้ในการติดต่อสื่อสารกับโฮส ในการทำงานนั้นอุปกรณ์ USB จะไม่สามารถเริ่มการติดต่อสื่อการบนระบบบัส USB ได้ด้วยตัวของมันเอง แต่มันจะต้องรอและตอบสนองกับการสื่อสารที่มาจากโฮส ยกเว้นในกรณีทีมีการใช้รีโมตเวคอัพ ซึ่งอนุญาตให้อุปกรณ์สามารถร้องขอการสื่อสารจากโฮสได้ สำหรับหน้าที่ของอุปกรณ์ USB มีดังนี้

- **ตรวจจับการสื่อสารที่เกิดขึ้น** อุปกรณ์แต่ละตัวจะตรวจจับแอดเดรสที่ถูกส่งมาในการสื่อสารที่เกิดขึ้นบนบัสแต่ละครั้ง ถ้าแอดเดรสไม่ตรงกับแอดเดรสที่เก็บอยู่ในตัวอุปกรณ์ อุปกรณ์ตัวนั้นจะไม่สนใจการสื่อสารครั้งนั้น แต่ถ้าแอดเดรสตรงกันอุปกรณ์จะเก็บข้อมูลเข้าไปในบัฟเฟอร์ที่ใช้สำหรับรับข้อมูล และกำเนิดสัญญาณอินเทอร์รัพเพื่อบอกให้โฮสทราบว่าข้อมมูลนั้นได้ถูกรับไปแล้ว ชิปส่วนใหญ่จะมีส่วนซึ่งทำงานเหล่านี้สร้างอยู่ภายในฮาร์ดแวร์และถูกกระทำอย่างอัตโนมัติ
- **ตอบสนองต่อคำร้องขอ** เมื่อทำการจ่ายไฟให้ระบบ หรือเมื่ออุปกรณ์ถูกเสียบเข้ากับระบบ อุปกรณ์จะต้องตอบสนองต่อคำร้องขอที่สร้างจากโฮสในขั้นตอนการทำอนิวเมอร์เรท และบางครั้งโฮสอาจส่งคำร้องขอมาตรฐานออกมาอีกครั้งหลังจากการอินิวเมอร์เรทเสร็จสมบูรณ์ อุปกรณ์ USB ทุกตัวจะต้องตอบสนองต่อคำร้องขอมาตราฐาน ซึ่งจะสอบถามเกี่ยวกับความสามารถและสถานะของอุปกรณ์ หรือร้องขอให้อุปกรณีทำงานอย่างใดอย่างหนึ่ง เมื่อมีการรับคำร้องขอ
- **การตรวจสอบความผิดพลาด** อุปกรณ์จะใส่บิตตรวจสอบความผิดพลาดเข้าไปในข้อมูลที่มันส่งออกไปเช่นเดียวกับโฮส และสำหรับข้อมูลที่มันรับเข้านั้นจะมีการตรวจสอบ

ความผิดพลาดรวมอยู่ด้วยเช่นกัน อุปกรณ์จะทำการคำนวณเพื่อตรวจสอบความผิดพลาดของข้อมูล และรายงานให้โฮสทราบว่าจะต้องทำการส่งข้อมูลใหม่หรือไม่ การทำงานทั้งหมดเหล่านี้จะถูกสร้างอยู่ภายในฮาร์ดแวร์ของส่วนที่ใช้ในการสื่อสารกับระบบบัส USB อยู่แล้ว ซึ่งเราไม่จำเป็นต้องเขียนโปรแกรมเพิ่มเติมลงไป

- **การจัดการพลังงาน** สำหรับอุปกรณ์ที่ใช้กำลังงานจากบัส เมื่อโฮสเข้าสู่โหมดการประหยัดพลังงาน หรือซัซเพนด์ การสื่อสารทั้งหมดจะหยุดลง รวมถึงสิ่งที่โฮสส่งออกมาเป็นคาบเวลาตามปกติด้วย เมื่ออุปกรณ์ที่ติดต่ออยู่บนบัสตรวจจับได้ว่าไม่มีการกระทำใด ๆ บนบัสเป็นเวลาประมาณ 3 มิลลิวินาที มันจะต้องเข้าสู่โหมดการประหยัดพลังงานและกำจัดกระแสที่พวกมันดึงจากบัสด้วย
- **การแลกเปลี่ยนข้อมูลกับโฮส** หน้าที่ทั้งหมดที่กล่าวมานี้เป็นงานที่สนับสนุนงานหลักของอุปกรณ์ USB ซึ่งหน้าที่หลักจริงๆ คือ การแลกเปลี่ยนข้อมูลกับโฮส โดยภายหลังจากที่อุปกรณ์ถูกตั้งค่าเพื่อเริ่มการทำงานแล้ว มันจะต้องตอบสนองต่อคำร้องขอเพื่อส่งและรับข้อมูลจากโฮสอย่างถูกต้อง

2.2.4 โพรโตคอลของระบบบัส USB

การเชื่อมต่อบนระบบบัส USB จะต่างจากการเชื่อมต่อพอร์ตแบบอนุกรมหรือแบบ RS 232 ซึ่งมีการกำหนดรูปแบบของการส่งข้อมูลอย่างเคร่งครัด โดยการสื่อสารข้อมูลบนระบบบัส USB จะสร้างขึ้นจากโพรโตคอลที่ค่อนข้างซับซ้อน และมีข้อกำหนดที่เคร่งครัด แต่ในความเป็นจริงแล้วการทำงานต่าง ๆ ในส่วนของโพรโตคอลนี้จะถูกจัดการอยู่ในไอซีคอนโครเลอร์ USB อย่างสมบูรณ์แล้ว ดังนั้นผู้ออกแบบส่วนมากจึงไม่จำเป็นต้องสนใจการทำงานในส่วนนี้เท่าใดนัก

โพรโตคอลของ USB ประกอบขึ้นจากทรานเซคชัน หลายชุดซึ่งแต่ละชุดจะประกอบไปด้วยโทเค็นแพคเก็ต (Token Packet), ดาตาแพคเก็ต (Data Packet), สเตตัสแพคเก็ต (Status Packet) ซึ่งรายละเอียดจะกล่าวต่อไปภายหลัง

ดังที่ได้กล่าวไปแล้วข้างต้นว่า USB เป็นระบบบัสที่ใช้โฮสเป็นศูนย์กลาง โดยโฮสจะเป็นตัวเริ่มสร้างทรานเซ็คชันทั้งหมด สำหรับแพ็คเก็ตแรกที่โฮสสร้างขึ้นเป็นการอ่านหรือเขียนข้อมูล นอกจากนี้ยังเป็นตัวบอกแอดเดรสของตัวอุปกรณ์และระบุชนิดของเอนพอยน์ด้วย โดยทั่วไปแพ็กเก็ตถัดมาจะเป็นดาตาแพ็คเก็ตซึ่งบรรจุข้อมูลเอาไว้ภายใน และปิดท้ายด้วยแฮนด์เช็คแพ็คเก็ตเพื่อใช้ในเป็นตัวรายงานผลของการสื่อสารข้อมูล

2.2.5 เอนด์พอยน์(Endpoint)

เอนพอยน์ คือ บัฟเฟอร์ซึ่งใช้สำหรับรับ หรือจ่ายข้อมูลโดยเอนพอยน์จะเกิดขึ้นที่จุดสิ้นสุดของช่องทางการสื่อสารบนฟังก์ชัน สำหรับการทำงานของการสื่อสารข้อมูลบนเอนด์พอยน์สามารถอธิบายได้ดังนี้ ถ้าโฮสต้องการติดต่อกับอุปกรณ์ โฮสจะระบุหมายเลขเอนพอยด์ที่ต้องการติดต่อมาด้วยออกมา จากนั้นดีไวซ์ไดรเวอร์บนโฮสจะทำการส่งแพกเก็ตไปที่อุปกรณ์โดยผ่านทางเอนพอยน์ที่ได้ระบุไว้ เช่น เอนด์พอยน์ 1 (EP 1) เป็นต้น เมื่อข้อมูลไหลออกจากโฮสแล้วมันจะไปสิ้นสุดที่บัฟเฟอร์ EP1 OUT จากนั้นเฟิร์มแวร์จะทำการอ่านข้อมูลออกมา ถ้าอุปกรณ์ต้องการข้อมูลส่งกลับออกมามันจะเขียนข้อมูลไปยัง EP1 IN แต่เนื่องจากฟังก์ชันจะไม่สามารถทำการเขียนข้อมูลลงไปบนบัสโดยตรงในขณะที่บัสถูกควบคุมจากโฮสได้ ดังนั้นมันจะต้องรอจนกว่าจะถึงเวลาที่โฮสส่งแพ็คเก็ต IN ไปยังเอนพอยน์ที่มีการร้องขอข้อมูลก่อนทำการส่งข้อมูลกลับออกไป

อุปกรณ์ USB ทุกตัวจะต้องสนับสนุนเอนพอยน์ศูนย์ ซึ่งเอนพอยน์นี้จะใช้สำหรับรับข้อมูลคอนโทรล รวมถึงคำร้องขอสถานะในขณะที่ทำการอินิวเมอร์เรท และในขณะระหว่างที่อุปกรณ์กำลังทำงานอยู่บนระบบบัสตลอดเวลา

2.2.6 ชนิดของเอนด์พอยน์

ข้อกำหนด USB ได้กำหนดชนิดของเอนพอยน์ไว้ 4 ชนิดซึ่งสอดคล้องกับชนิดของการส่งถ่ายข้อมูลได้แก่

- คอนโทรลเอนด์พอยน์ (Control Endpoint)
- อินเทอร์รัพเอนด์พอยน์ (Interrupt Endpoint)
- ไอโซโครนัสเอนพอยน์ (Isochonous Endpoint)
- บัลก์เอนพอยน์ (Bulk Endpoint)

2.2.7 ไปป์ (Pipe)

ไปป์เป็นการเชื่อมต่อทางลอจิกระหว่างโฮสและเอนด์พอยน์ (การเชื่อมต่อทางลอจิก คือ การเชื่อมต่อระหว่างจุด 2 จุด ซึ่งสามารถเชื่อมต่อกันได้โดยไม่ได้ใช้ตัวกลาง เช่น สายไฟ)โดยในขณะที่อุปกรณ์ส่งหรือรับข้อมูลบนเอนพอยน์ ซอฟแวร์จะทำการรับส่งข้อมูลโดยผ่านทางไปป์ ซึ่งพารามิเตอร์ที่เกี่ยวข้องกับการใช้งานไปป์ ได้แก่ ขนาดของแบนด์วิดธ์ ชนิดของการส่งถ่ายข้อมูล ทิศทางการไหลของข้อมูล และขนาดสูงสุดของแพ็กเก็ต ในข้อกำหนด USB ได้กำหนดชนิดของไปป์ไว้ 2ชนิดได้แก่

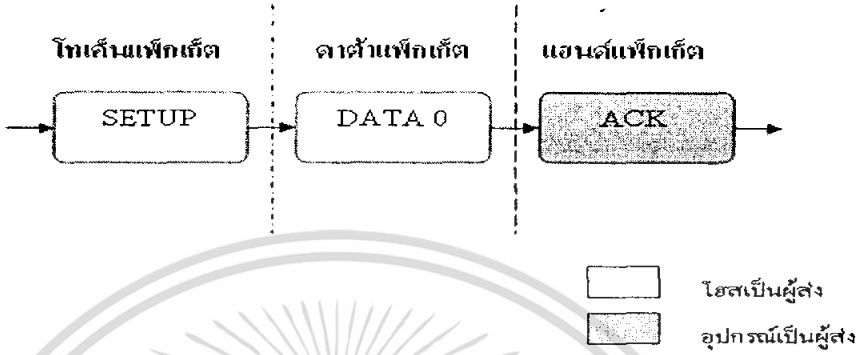
- สตรีมไปป์ (Stream Pipe) ข้อกำหนด USB ไม่ได้กำหนดรูปแบบของข้อมูลที่ถูกส่งไปบนไปป์ชนิดนี้เอาไว้ ซึ่งเราสามารถส่งข้อมูลชนิดใดก็ได้ลงไปบนสตรีมไปป์ ข้อมูลจะไหลอย่างเป็นลำดับลีทิศทางตามที่ได้ถูกกำหนดไว้ว่าให้เป็น IN หรือ OUT สตรีมไปป์จะรองรับการส่งถ่ายข้อมูลแบบบัลก์ ไอโซโครนัส และ อินเทอร์รัพท์ โดยทั้งโฮสและอุปกรณ์ สามารถควบคุมสตรีมไปป์ได้
- เมสเสจไปป์ (Message Pipe) ในข้อกำหนด USB ได้มีการกำหนดรูปแบบของข้อมูลที่ส่งไปบนไปป์ชนิดนี้เอาไว้ โดยมันจะถูกควบคุมจากโฮสซึ่งเริ่มต้นด้วยการส่งคำร้องขอจากโฮส จากนั้นข้อมูลจะถูกส่งถ่ายไปในทิศทางที่ต้องการที่ได้กำหนดไว้ในคำร้องขอเมสเสสไปป์ จะอนุญาติให้ข้อมูลไหลได้ทั้งสองทิศทางแต่ไปป์ชนิดนี้จะสนับสนุนการส่งถ่ายข้อมูลแบบคอนโทรล์เท่านั้น

2.2.8 ชนิดของการส่งถ่ายข้อมูลในระบบบัส USB

ข้อกำหนด USB ได้กำหนดรูปแบบการส่งถ่ายข้อมูลไว้ 4 ชนิด ได้แก่ ส่งถ่ายข้อมูลแบบคอนโทรล บัลก์ อินเทอร์รัพ และไอโซโครนัส ซึ่งแต่ละชนิดจะมีจุดประสงค์การใช้งานที่แตกต่างกัน

- การส่งถ่ายข้อมูลแบบคอนโทรล เป็นการส่งถ่ายข้อมูลเพียงชนิดเดียวซึ่งมีฟังก์ชันที่ถูกกำหนดโดยข้อกำหนด USB การส่งถ่ายข้อมูลแบบนี้จะทำให้โฮสสามารถอ่านข้อมูลเกี่ยวข้องกับอุปกรณ์ตั้งค่าแอดเดรสของอุปกรณ์ เลือกคอนฟิกิวเรชัน และตั้งค่าอื่นได้ อุปกรณ์ USB ทุกตัวจะต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้ การส่งถ่ายข้อมูลชนิดนี้เป็นการสื่อสารข้อมูลแบบสองทิศทางซึ่งใช้สำหรับการตั้งค่าการทำงาน การส่งคำสั่งหรือข้อมูล ชนิดนี้จะเริ่มต้นขึ้นจากโฮส และใช้ความพยายามในการส่งมากที่สุด นอกจากนี้ยังมีการตรวจสอบความผิดพลาดด้วยการใช้ CRC (Cyclic Redundancy Check) เพื่อไม่ให้ข้อมูลเกิดความผิดพลาดได้เนื่องจากข้อมูลชนิดนี้มีความสำคัญมากที่สุด ขนาดของข้อมูลซึ่งถูกใส่ลงไปในฟิลด์ข้อมูลของดาต้าแพกเก็ตหรือเรียกทับศัพท์ว่า ดาต้าเพย์โหลด (Data Payload) สำหรับการส่งถ่ายข้อมูลคอนโทรลในอุปกรณ์โลว์สปีดจะมีขนาดเท่ากับ8ไบต์ อุปกรณ์ฟูลสปีดจะมีขนาดเท่ากับ 64 ไบต์ และอุปกรณ์ไฮสปีดจะมีขนาดเท่ากับ 8, 16, 32 หรือ 64 ไบต์ ส่วนขั้นตอนการทำงานหรือสเตจ (Stage) ของมันอาจประกอบไปด้วย 2 หรือ 3 สเตจ ได้แก่ เซ็ตอัพสเตจ ดาตาสเตจ (อาจจะมีหรือไม่มีก็ได้) และสเตตัสสเตจ
  1. เซ็จอัพสเตจ เป็นขั้นตอนที่คำร้องขอถูกส่งออกไปประกอบด้วยแพ็กเก็ต 3 ชุด โดยโทเค็นแพ็กเก็ตจะถูกส่งออกไปเป็นอันดับแรก PID ของแพ็กเก็ตนี้จะเป็นเซ็ตอัพ (SET UP) จากนั้นดาต้าแพ็กเก็ตจะถูกส่งออกมาเป็นลำดับถัดไป ภายในแพ็กเก็ตนี้ประกอบไปด้วยเซ็ตอัพที่มีข้อมูลเกี่ยวกับคำร้องขอ ส่วนแพ็กเก็ตสุดท้ายคือแฮนด์

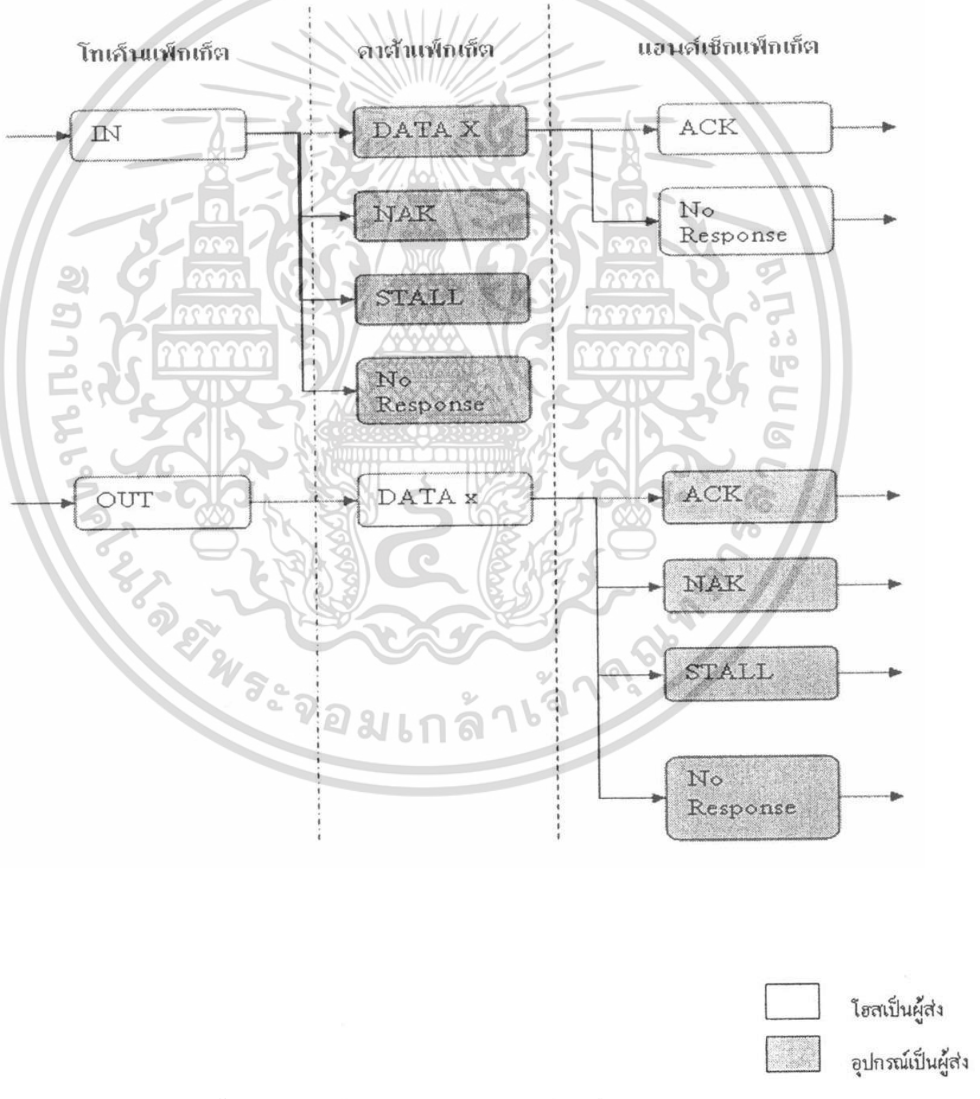
เช็กแพกเก็ตซึ่งใช้สำหรับการบอกสถานะข้อมูลว่าสำเร็จหรือไม่ ถ้าอุปกรณ์รับข้อมูลถูกต้องมันจะตอบกลับด้วย ACK เพียงอย่างเดียว



รูปที่ 2.16 เซ็ตอัพสเตจของการส่งถ่ายข้อมูลคอนโทรล

2. ดาต้าสเตจ (Data Stage) สเตจนี้อาจมีหรือไม่มีก็ได้ โดยมันจะประกอบไปด้วยการส่งถ่ายข้อมูลแบบ IN หรือ OUT อย่างใดอย่างนึงเป็นจำนวน 1 ครั้งหรือมากกว่านั้น ซึ่งจำนวนของข้อมูลที่ส่งไปในสเตจนี้จะถูกกำหนดไว้ในเซ็ตอัพแพ็กเก็ต ถ้าหากข้อมูลมีขนาดเกินขนาดสูงสุดของดาต้าเพย์โหลด ข้อมูลนั้นจะถูกส่งด้วยการส่งถ่ายข้อมูลหลายครั้ง โดยในแต่ละครั้งจะมีขนาดสูงสุดของดาต้าเพย์โหลดยกเว้นการส่งครั้งสุดท้าย ดาต้าสเตจจะมีบทบาทที่แตกต่างกัน 2 อย่างโดยขึ้นอยู่กับทิศทางของการส่งถ่ายข้อมูล ได้แก่
   - IN เกิดขึ้นในการส่งถ่ายข้อมูลแบบคอนโทรล Read โดยเมื่อโฮสพร้อมรับข้อมูลคอนโทรล โฮสจะส่งโทเค็น IN ออกมา ถ้าอุปกรณ์ได้รับโทเค็น IN ที่มีความผิดพลาด เช่น PID กับส่วนอินเวิร์ส ของมันไม่สอดคล้องกัน อุปกรณ์จะทิ้งแพ็กเก็ตนี้ไป แต่ถ้าโทเค็นที่ได้รับกลับมาถูกต้อง อุปกรณ์อาจตอบกลับมาด้วยดาตาแพ็กเก็ตซึ่งบรรจุข้อมูลคอนโทรลที่จะส่ง หรืออาจเป็นแพกเก็ต STALL เพื่อเป็นการบอกว่าเกิดการผิดพลาดขึ้นที่เอนพอยด์ หรืออาจเป็นแพ็กเกต NAK เพื่อเป็นการบอกโฮสว่าเอนด์พอยสามารถทำงานได้ตามปกติแต่ในขณะนั้นไม่มีข้อมูลที่จะส่ง
   - OUT เกิดขึ้นในการส่งถ่ายข้อมูลแบบคอนโทรล Write โดยเมื่อโฮสต้องการส่งคอนโทรลแพ็กเก็ตไปยังอุปกรณ์ มันจะส่งโทเค็น OUT ออกมาแล้วตามด้วยดาต้าแพ็กเก็ตซึ่งบรรจุดาต้าเพย์โหลดที่เป็นข้อมูลคอนโทรลเอาไว้ ถ้าบางส่วน

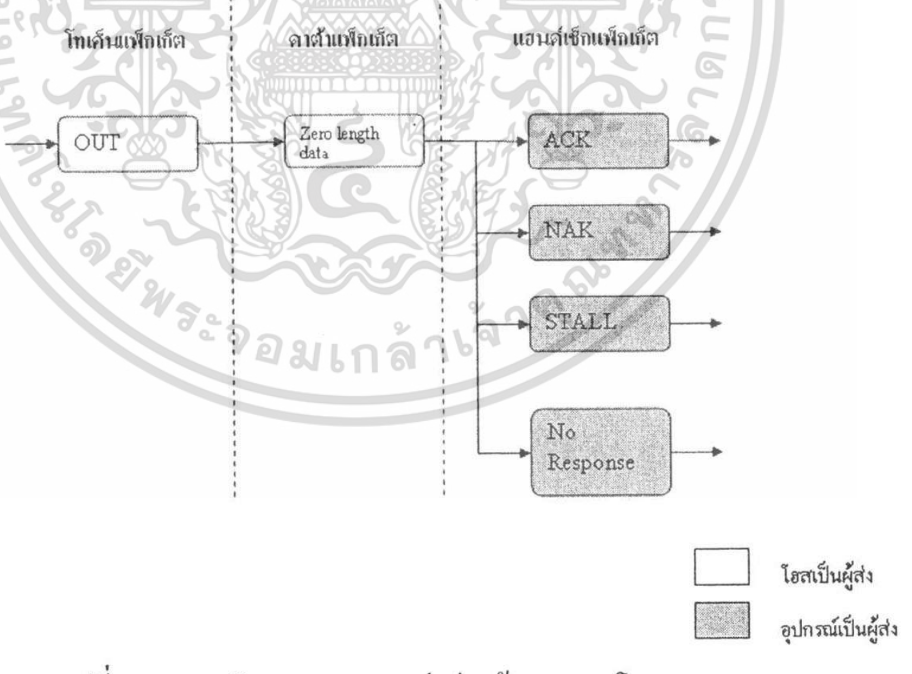
ของโทเค็น OUT หรือดาต้าแพ็กเก็ตไม่ถูกต้องอุปกรณ์จะทิ้งแพ็กเก็ตนี้ไป ถ้าเอนด์พอยน์บัพเฟอร์ของอุปกรณ์ว่างและไม่สามรถป้อนข้อมูลเข้าไปในเอนพอยน์บัพเฟอร์ได้ อุปกรณ์จะส่ง ACK ออกมาเพื่อแจ้งให้โฮสทราบว่ามันรับข้อมูลสำเร็จแล้ว ถ้าเอนด์พอยน์บัพเฟอร์ ไม่ว่างเนื่องจากการประมวลผลแพ็กเก็ตก่อนหน้านี้ไม่เสร็จสมบูรณ์ อุปกรณ์จะส่งค่า NAK กลับไป แต่อย่างไรก็ตามมันอาจส่ง STALL กลับออกมาถ้าเอนด์พอยน์มีข้อผิดพลาดและบิต HALT ถูกเซตอยู่ ซึ่งแสดงได้ดังรูป



รูปที่ 2.17 ดาต้าสเตจของการส่งถ่ายข้อมูลคอนโทรล

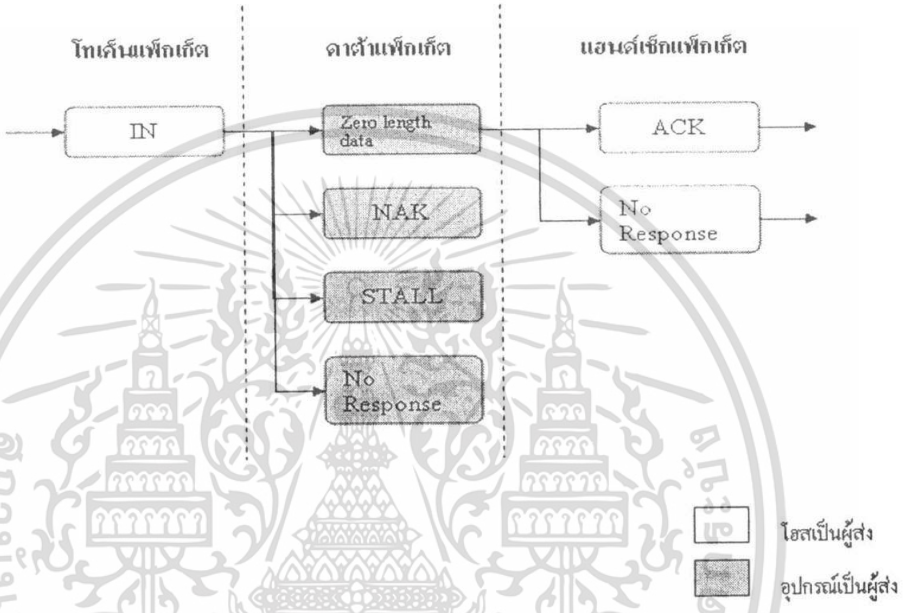
3. <u>สเตตัสสเตจ</u> (Status Stage) เป็นสเตจที่ใช้รายงานสถานะของการร้องขอทั้งหมด และทั้งมีทิศทางแปรเปลี่ยนไปตามการเปลี่ยนแปลงทิศทางของการส่งถ่ายข้อมูล ซึ่งการรายงานสถานะจะกระทำอุปกรณ์เสมอ

   - ถ้าการส่งถ่ายจข้อมูลคอนโทรลครั้งนั้น ๆ ไม่มีดาต้าสเตจอุปกรณ์จะตอบกลับไปด้วย ACK แต่ถ้าข้อมูลมีความผิดพลาดมันจะทิ้งข้อมูลนั้นและไม่ส่งแฮนด์เช็กแพ็กเก็ตกลับไปยังโฮส
   - กรณีที่เป็นคอนโทรล Read โฮสจะต้องตอบรับ ACK ว่าการรับข้อมูลนี้ทำได้สำเร็จโดยโฮสต้องทำการส่งโทเค็น OUT แล้วตามด้วยดาต้าแพ็กเก็ตที่มีความยาวเป็นศูนย์ ในขณะนี้อุปกรณ์จะสามรถรายงานสถานะของมันลงมาในแฮนด์เช็กสเตจด้วย ACK ซึ่งเป็นการบอกว่าอุปกรณ์ได้ทำตามคำสั่งโดยสมบูรณ์แล้วพร้อมที่จะรับคำสั่งถัดไปถ้าเกิดข้อผิดพลาดขึ้น ในขณะประมวลผลคำสั่งนี้อุปกรณ์จะส่ง STALL ออกมาอย่างไรก็ตาม ถ้าอุปกรณ์ยังคงประมวลผลต่อไปมันจะส่ง NAK กลับออกมาเพื่อแจ้งให้โฮสทำสเตตัสสเตจอีกครั้งในภายหลัง



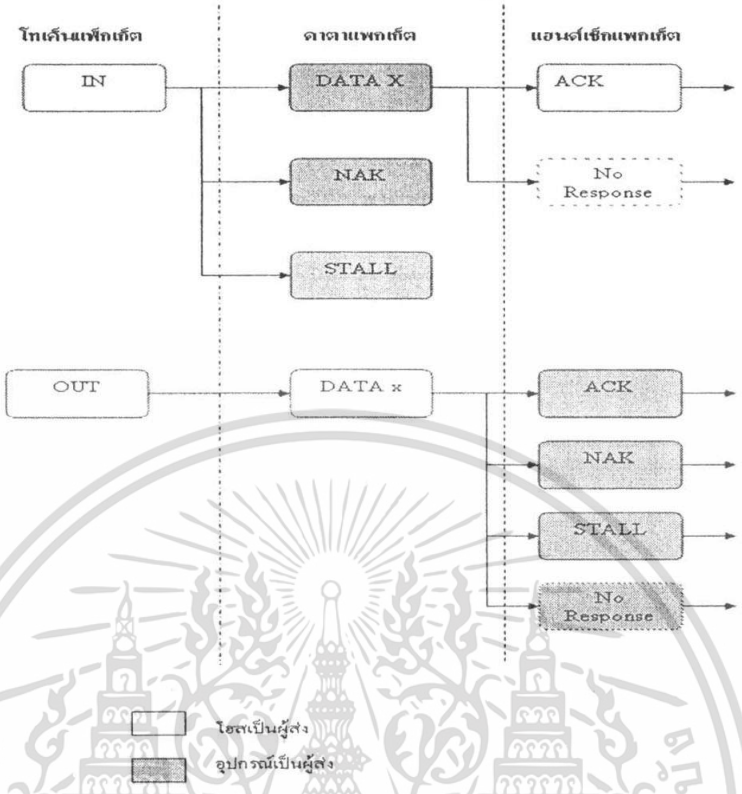
รูปที่ 2.18 สเตตัสสเตจของการส่งถ่ายข้อมูลคอนโทรล Read

- กรณีที่เป็นคอนโทรล Write อุปกรณ์จะตอบรับว่าการรับข้อมูลทำได้สำเร็จโดยการส่งแพกเก็ตที่มีความยาวเป็นศูนย์ในการตอบสนองโทเค็น IN อย่างไรก็ตาม ถ้ามีข้อผิดพลาดเกิดขึ้นมันจะส่ง STALL ออกมา หรือถ้าอุปกรณ์ยังคงอยู่ในระหว่างการประมวลผลข้อมูลอยู่มันจะส่ง NAK ออกไปเพื่อบอกโฮสให้เข้าสเตตัสอีกครั้งภายหลัง



รูปที่ 2.19 สเตตัสของการส่งถ่ายข้อมูลคอนโทรล Write

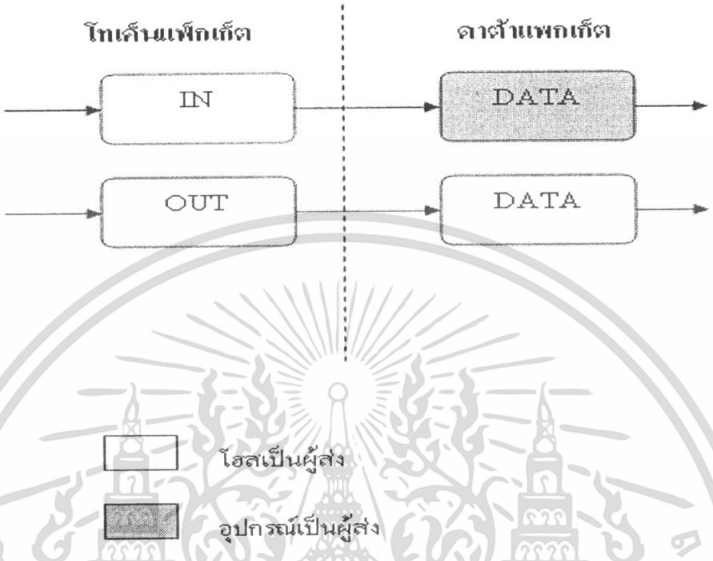
- **การส่งถ่ายข้อมูลแบบบัลก์** ถูกออกแบบมาเพื่อใช้กับงานที่ไม่คำนึงถึงอัตตราการส่งถ่ายข้อมูลเป็นหลัก เช่น การส่งไฟล์ข้อมูลไปยังพริ้นเตอร์หรือการรับข้อมูลจากสแกนเนอร์ แต่จะเป็นการที่ต้องการความถูกต้องของข้อมูลเป็นหลัก สำหรับกรณีที่จำเป็นข้อมูลที่ถูกส่งถ่ายด้วยวิธีการนี้จะต้อวสามารถหยุดเพื่อรอการส่งถ่ายข้อมูลได้ เช่น ถ้าในระบบบัสมีการขนส่งข้อมูลหนาแน่นมากเนื่องจากมีการส่งถ่ายข้อมูลแบบอื่น ๆ ซึ่งมีการรับประกันอัตราการส่งถ่ายข้อมูลอยู่ การส่งถ่ายข้อมูลบัลก์อาจต้องหยุดการส่งถ่ายข้อมูลขณะนั้นเพื่อรอให้บัสว่างลง สำหรับอุปกรณ์ที่สนับสนุนการส่งถ่ายข้อมูลบัลก์นี้จะมีเพียงแค่อุปกรณ์แบบฟูลสปีดและไฮสปีดเท่านั้น อุปกรณ์ USB ทุกตัวอาจไม่ต้องการการสนับสนุนการส่งถ่ายข้อมูลบัลก์ก็ได้



รูปที่ 2.20 เฟสของการส่งถ่ายข้อมูลแบบบัลก์

- **การส่งถ่ายข้อมูลแบบอินเทอร์รัพท์** จะใช้สำหรับอุปกรณ์ซึ่งต้องการรับสัญญาณจากโฮสหรือเรียกร้องความสนใจของอุปกรณ์เป็นช่วงเวลาที่สม่ำเสมอ นอกเหนือจากการส่งถ่ายข้อมูลแบบคอนโทรลแล้วการส่งถ่ายข้อมูลอินเทอร์รัพท์จะเป็นอีกวิธีหนึ่งซึ่งอุปกรณ์แบบโลว์สปีดสามารถใช้ในการส่งถ่ายข้อมูลได้ คีย์บอร์ดและเมาส์จะใช้ในการส่งถ่ายข้อมูลอินเทอร์รัพท์ในการส่งสัญญาณการกดจากคีย์บอร์ดและข้อมูลตำแหน่งของการเลื่อนเมาส์ การส่งถ่ายข้อมูลแบบอินเทอร์รัพท์สามารถใช้กับโหมดการสื่อสารที่ความเร็วใดก็ได้ ทั้งนี้อุปกรณ์ USB ทุกตัวไม่จำเป็นต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้
- **การส่งถ่ายข้อมูลแบบไอโซโครนัส** เป็นการส่งถ่ายข้อมูลที่มีการรับประกันเวลาการส่งแต่จะไม่มีการแก้ไขความผิดพลาดที่เกิดขึ้น ข้อมูลซึ่งเหมาะสำหรับการส่งข้อมูลชนิดนี้ได้แก่ ไฟล์ข้อมูลเสียงซึ่งต้องมีการใช้งานในแบบเวลาจริง (real time) การส่งถ่ายข้อมูลชนิดนี้เป็นเพียงชนิดดียวที่ไม่สนับสนุนการส่งซ้ำอย่างอัตโนมัติในกรณีที่มีการส่งข้อมูลเกิดความผิดพลาด ดังนั้นความผิดพลาดที่อาจเกิดขึ้นบ้างจึงเป็นสิ่งที่ต้องยอมรับได้

อุปกรณ์ที่สนับสนุนการส่งถ่ายข้อมูลชนิดนี้จะมีอุปกรณ์ แบบไฮสปีด และแบบโลว์สปีด ซึ่งอุปกรณ์ USB ทุกตัวไม่จำเป็นต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้



รูปที่ 2.21 เซ็ตอัพสเตจของการส่งถ่ายข้อมูลไอโซโครนัส

รูปแบบของทรานแซคชั่นแบบไอโซโครนัส IN และ OUT แสดงดังรูปที่ 2.21 ซึ่งทรานแซคชั่นแบบนี้ไม่มีสเตตัสสำหรับการทำแฮนด์เช็กและไม่สามารถรายงานความผิดพลาดได้ (STALL และ HALT)

2.2.9 การทำให้ระบบรู้จักอุปกรณ์ที่สร้างขึ้น (Enumeration)

ตอนนี้สิ่งที่เราต้องการคือให้วินโดวส์ทำการอินิวเมอร์เรทอุปกรณ์ที่เราสร้างขึ้น

1. เขียนโค๊ดที่จำเป็นสำหรับคอนโทรเลอร์ชิพเพื่อใช้ในการอินิวเมอร์เรท ซึ่งรายละเอียดส่วนนี้จะขึ้นอยู่กับชิปที่ใช้ โดยคอนโทรเลอร์ชิปทุกตัวต้องสามารถส่งชุดของดีสคริปเตอร์ไปยังโฮสได้ นอกจากภายในชิปยังต้องมีโปรแกรมหรือฮาร์ดแวร์ซึ่งจดจำและตอบสนองกับคำร้องขอซึ่งโฮสส่งออกมาเมื่อมันทำการอินิวเมอร์เรทอุปกรณ์รวมอยู่ภายในตัวมันด้วย โดยทั่วไปแล้วผู้ผลิตชิปจะให้โค้ดตัวอย่างมาด้วยเสมอ ซึ่งเราสามารถนำโค๊ดเหล่านี้มาทดลองแก้ไขเพื่อศึกษาการทำงานของมันได้
2. สร้างไฟร์ INF (Information) ซึ่งเป็นไฟล์ที่ทำให้วินโดวส์สามารถรู้จักกับอุปกรณ์บน ขั้นตอนของการทำอินิวเมอร์เรทได้ ไฟล์ INF จะเป็นเทกซ์ไฟล์หรือไฟล์ตัวอักษร ซึ่งสามารถสร้างได้จากการใช้โปรแกรมเทกซ์อิดิเตอร์ใดๆ ก็ได้ภายในไฟล์นี้จะมีการ

ระบุชื่อของไดร์แวอร์ซึ่งอุปกรณ์จำเป็นต้องใช้ ตามปกติผู้ผลิตชิปมักจะให้ตัวอย่างไฟล์ INF มาด้วยนอกจากนี้เราอาจไฟล์ INF ที่มาพร้อมกับวินโดวส์ได้ถ้าอุปกรณ์ที่สร้างขึ้นนั้นเป็นอุปกรณ์มาตรฐานซึ่งสนับสนุนโดยวินโดวส์

3. ออกแบบและสร้างวงจรซึ่งในบางครั้งเราอาจจะเริ่มต้นด้วยการใช้บอร์ดพัฒนาซึ่งสามารถหาซื้อได้จากผู้ผลิตชิปโดยตรง
4. โหลดโค๊ดหรือเฟิร์มแวร์เข้าไปภายตัวอุปกรณ์ แล้วเสียบอุปกรณ์เข้ากับระบบบัส หลังจากวินโดส์จะทำการอินิวเมอร์เรทอุปกรณ์และเพิ่มมันลงไปในรายงานฮาร์ดแวร์ที่คอนโทรพาเนล
5. ดีบักและทำซ้ำขั้นตอนทั้งหมดถ้าจำเป็น

2.2.10 การสร้างการแลกเปลี่ยนข้อมูล

ขั้นตอนสุดท้ายซึ่งจำเป็นขั้นตอนที่สำคัญในการสร้างอุปกรณ์ USB คือ การเขียนโค้ดและโปรแกรมสำหรับการแลกเปลี่ยนข้อมูลบนระบบบัสเพื่อทำให้อุปกรณ์สามารถทำงานตามจุดประสงค์ที่ตั้งไว้

1. เขียนเฟิร์มแวร์ให้กับคอนโทรลเลอร์ชิป
2. ถ้าอุปกรณ์ที่สร้างขึ้นนี้ใช้ไดร์ฟเวอร์ที่เขียนขึ้นเอง เราจะต้องเขียนไดร์ฟเวอร์เพื่อให้สื่อสารระหว่างแอพพลิเคชันโปรแกรมกับอุปกรณ์ แต่ถ้าอุปกรณ์ที่เราสร้างขึ้นเป็นอุปกรณ์มาตราฐานซึ่งสนับสนุนโดยวินโดวส์ เราก็สามารถให้ไดร์ฟเวอร์ที่ให้มาพร้อมกับวินโดวส์ได้
3. เขียนแอพพลิเคชั่นซอฟแวร์สำหรับติดต่อสื่อสารอุปกรณ์ที่สร้างขึ้น แต่ถ้างานของเครือการออกแบบเมาส์ คีย์บอร์ด หรืออุปกรณ์มาตรฐานอื่นๆ เราจะสามารถใช้งานอุปกรณ์ที่สร้างขึ้นนี้ร่วมกับแอพพลิเคชั่นซอฟแวร์ใดๆ ก็ได้
4. ทดสอบการทำงาน

2.2.11 การทำแฮนด์เช็ก(Handshaking)

การทำแฮนด์เช็กแบ่งเป็นการทำแฮนด์เช็กด้วยฮาร์ดแวร์ และการทำแฮนด์เช็กด้วยซอฟแวร์ สำหรับการทำแฮนด์เช็กโดยใช้ฮาร์ดแวร์จะมีสายสัญญาณเส้นหนึ่งสำหรับเป็นตัวส่งสัญญาณแฮนด์เช็ก เช่น สาย RTS และ CTS ในการเชื่อมต่อแบบ RS232 สำหรับการทำแฮนด์เช็กโดยใช้ซอฟแวร์นั้นจะมีสายสัญญาณ เส้นหนึ่งสำหรับส่งโค๊ดของแฮนด์เช็กเช่นกัน เช่นโค้ด XON และ XOFF ที่ถูกส่งไปบนสายดาต้าในการเชื่อมต่อแบบ RS 232

ระบบบัส USB จะใช้ซอฟต์แวร์ในการทำแฮนด์เช็ก โดยใช้โค้ดแฮนด์เช็กเป็นตัวบอกถึงความสำเร็จหรือความผิดพลาดของทรานแซคชัน ยกเว้นการส่งถ่ายข้อมูลแบบไอโซโครนัส นอกจากนี้ในการส่งถ่ายข้อมูลแบบคอนโทรลจะมีสเตตัสสเตจเพื่อให้ตัวอุปกรณ์จะสามารถรายงานความสำเร็จหรือความผิดพลาดของการส่งถ่ายข้อมูลออกมาได้

1. ACK (Acknowledge) เป็นสเตตัสโค้ดที่ใช้บ่งบอกว่าโฮสหรืออุปกรณ์ได้รับข้อมูลอย่างถูกต้องปราศจากความผิดพลาด
2. NAK (Negative Acknowledge) NAK หมายถึง อุปกรณ์อยู่ในสถานะไม่ว่าง (Busy) หรือไม่มีข้อมูลส่งกลับ
3. STALL แฮนด์เช็ก STALL ให้ความหมายได้ถึงสามอย่าง ได้แก่ การไม่สนับสนุนต่อคำร้องขอ คอนโทรล การร้องขอคอนโทรลไม่สำเร็จ และการเกิดความผิดพลาดขึ้นที่เอนด์พอยน์
4. NYET แฮนด์เช็ก NYET จะมีการใช้กับอุปกรณ์ไฮสปีดเท่านั้น หมายความว่าไม่มีการทำตามคำร้องขอได้ในขณะนี้
5. ERR แฮนด์เช็ก ERR จะถูกใช้กับไฮสปีดในคอมพลีตสปริตทรานแซคชั่นเท่านั้น ซึ่ง ERR หมายถึง อุปกรณ์ไม่ได้ส่งค่าแฮนด์เช็กที่ต้องการกลับมาในทรานแซคชั่นของฮับกับโฮสที่กำลังจะเสร็จสมบูรณ์
6. การไม่ตอบสนอง (No Response) โค้ดบอกสถานะตัวสุดท้ายจะเกิดขึ้นหรืออุปกรณ์คาดหมายว่าจะได้รับแฮนด์เช็ก แต่มันกลับไม่ได้รับอะไรเลย ซึ่งปกติแล้วเหตุการณ์นี้จะเป็นการบอกถึงการคำนวณความผิดพลาดของตัรับได้ตรวจพบข้อผิดพลาดและแจ้งแก่ผู้ส่งว่าให้มันพยายามส่งกลับมาใหม่อีกครั้ง หรือถ้ามีการส่งหลายครั้งแต่ยังผิดพลาดให้ไปทำงานอย่างอื่นแทน

2.3 การวาดรูปในวินโดว์

2.3.1 รู้จักกับดีไวซ์คอนเท็กซ์ และGDI

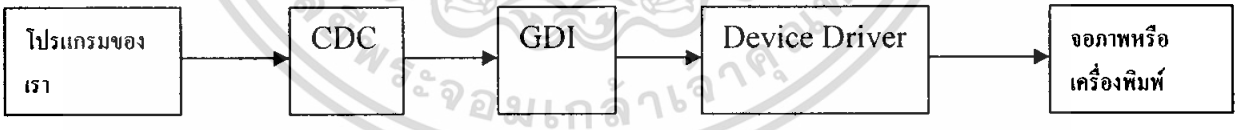
เมื่อแอพพลิเคชันที่รันบน Windows (หรือแม้แต่ Windows เองก็ตาม) จะเขียนข้อความหรือวาดรูปภาพเพื่อแสดงผล ไม่ว่าจะเป็นจอภาพ หรือเครื่องพิมพ์ โดยปกติแล้ว Windows ไม่ได้วาดโดยตรงกับฮาร์ดแวร์เหมือนกับโปรแกรมดอสทำ ทั้งนี้เพราะถูกห้ามไว้ด้วยกฎของ Windows

แอพพลิเคชันชนิดต่างๆ จะต้องใช้ตัวกลางที่เราเรียกว่า ดีไวซ์คอนเท็กซ์ เรียกย่อๆ ว่า ดีซี (DC) ทำหน้าที่วาดแทน

ดีไวซ์คอนเท็กซ์เป็นโครงสร้างข้อมูลที่กำหนดออบเจ็กต์กราฟิก และแอตทริบิวต์ (Attributes) ต่างๆรวมทั้งโหมดกราฟิกที่มีผลต่อเอาต์พุต

ดีไวซ์คอนเท็กซ์ไม่ถูกจำกัดเพียงแต่อุปกรณ์ที่มีอยู่จริงจับต้องได้ ซึ่งเรียกฟิสิกส์คอลดีไวซ์ (Physical device) เท่านั้น แต่ดีไวซ์คอนเท็กซ์สามารถอ้างถึงอุปกรณ์ของโลจิกคอลดีไวซ์ เช่น เมต้าไฟล์ (metafile) ซึ่งเป็นกลุ่มของโครงสร้างที่เก็บรูปภาพในฟอร์แมตที่อิสระต่อฮาร์ดแวร์ ดังนั้น DC จึงถูกนำมาใช้อย่างกว้างขวางใน Windows ซึ่งก็คือ Graphic Device Interface ซึ่งเรียกย่อยๆ ว่า GDI โดย GDI นั้นจะมีฟังก์ชั่น Windows API จำนวนมากให้ใช้งาน

ในการใช้งานจะผ่านค่าพารามิเตอร์ต่างๆ จาก GDI ซึ่ง GDI จะนำไปเรียกใช้ Device Driver ที่เหมาะสมกับอุปกรณ์ชนิดต่างๆ เพื่อให้แสดงกราฟิกได้อย่างถูกต้อง จากที่กล่าวมาเราสามารถแสดงได้ดังรูป



รูปที่2.22 ขั้นตอนการแสดงภาพกราฟิกบนจอคอมพิวเตอร์

Graphic Device Interface

2.3.2 คอมพิวเตอร์กราฟิก 2D โดยใช้ไลบรารี OpenGL

กราฟิกเป็นภาพที่สร้างจากเครื่องคอมพิวเตอร์ ซึ่งเราพบได้จากภาพยนตร์ที่ใช้เทคนิคการสร้างภาพจากเครื่องคอมพิวเตอร์ คอมพิวเตอร์กราฟิกจะมีการอ้างถึงเครื่องมือ (tool) ที่ใช้สำหรับการสร้างภาพ ซึ่งมีอยู่มากมายแต่ในที่นี้เราจะพูดถึงเครื่องมือของไลบรารี OpenGL (Open Graphic Library) ซึ่งเป็นของบริษัท ซิลิกอน กราฟิกอิงค์ จำกัด

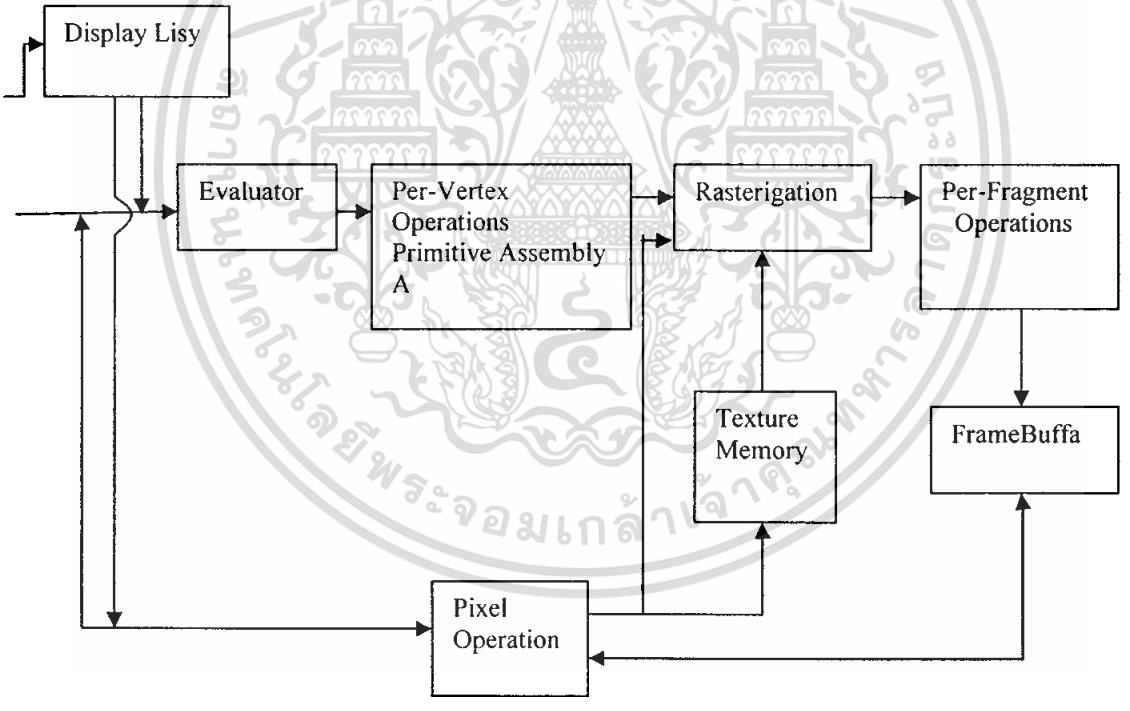
OpenGL คือ ซอฟต์แวร์อินเตอร์เฟสกับฮาร์ดแวร์กราฟิก (จอภาพกับการ์ดจอภาพ) อินเตอร์เฟสประกอบด้วยโพรซีเจอร์ และฟังก์ชั่นนับร้อยๆ ชุด ซึ่งอนุญาตให้นักเขียนโปรแกรมนำมาใช้ในการสร้างกราฟิกที่มีคุณภาพสูงโดยเฉพาะการสร้างภาพสามมิติ (3 Dimension)

ส่วนใหญ่แล้ว OpenGL จะระบุว่าการ์ดจอภาพที่ใช้ควรจะประกอบด้วยเฟรมบัฟเฟอร์ (Frame Buffer) ซึ่งจะใช้เมื่อนำภาพบิตแมปเข้ากับรูป Polygon (เรียกว่า Texturing) หรือทำการวาดรูปโดยการใช้เส้นหรือเส้นโค้งให้ราบเรียบ (Anti-aliasing) เป็นต้น

การสร้างภาพโดยใช้ไลบรารี OpenGL จะใช้ Rendering Context (RC) แทนที่จะเป็นดีไวซ์คอนเท็กซ์ อย่างไรก็ตาม RC ก็ยังทำงาน ผ่านDCอีกครั้งก่อนที่จะแสดงภาพบนหน้าจอคอมพิวเตอร์

2.3.2.1 การทำงานของ OpenGL

การทำงานของ OpenGL แสดงได้ดังรูป



รูปที่2.23 การทำงานของOpenGL

จากรูปคำสั่งจะเข้าสู่ OpenGL ทางด้านซ้ายมือ บางคำสั่งระบุรูปทรงเรขาคณิตที่จะถูกวาด ในขณะที่คำสั่งอื่นๆ ควบคุมรูปทรงเรขาคณิตหรือออบเจ็กต์อื่นที่จะถูกจัดการ โดยวิธีหลายๆ สเตท คำสั่งส่วนใหญ่จะถูกสะสมไว้ใน Display list เพื่อทำการประมวลผลโดย OpenGL ในภายหลัง คำสั่งจะถูกส่งอย่างมีประสิทธิภาพตลอดไปตามลูกศรชี้ไป

สเตทแรก(Evaluator) จะให้ความหมายประสิทธิภาพสำหรับการประมาณ เส้นโค้ง และพื้นผิวเรขาคณิต โดยการหาค่าฟังก์ชั่นโพลิโนเมียน (Polynomial Function) ของค่าอินพุทที่เข้ามา

สเตทต่อไป(Per-Vertex Operations Primitive Assembly) ที่จะทำคือ ทำรูปทรง Primitive เรขาคณิต (เช่น รูปทรงสี่เหลี่ยมสองมิติ) ซึ่งถูกวาดด้วยจุดยอดต่างๆ (Vertices) เช่น จุด ส่วนของเส้น(line segment) และรูปหลายเหลี่ยม (Polygon) ในสเตทนี้จุดยอดต่างๆ จะถูกแปลงและถูกให้แสง และรูปทรง Primitive ต่างๆ จะถูกคลิปเพื่อให้เห็น เป็นรูปทรงปริมาตร ซึ่งจะเตรียมการไว้สำหรับสเตทถัดไป ซึ่งก็คือ Rasterization

สเตท Rasterizer จะสร้างชุดของแอดเดรสเฟรมบัฟเฟอร์ และค่าต่างๆด้วยการใช้การวาดรูป 2 มิติของจุดส่วนของเส้นหรือรูปหลายเหลี่ยม แต่ละ Fragment ของรูปที่สร้างขึ้นจะถูกป้อนไปยังสเตทต่อไป ซึ่งกระทำการปฏิบัติต่อ Fragment เดียว ซึ่งเรียกว่า Per-Fragment Operation บล็อก Per-Fragment Operation เป็นบล็อกก่อนท้ายสุดมีหน้าที่กระทำกับข้อมูลก่อนที่จะบันทึกเป็นพิกเซลไว้ในบล็อก FrameBuffer (ซึ่งเป็นบล็อกท้ายสุด)

ซึ่งจากรูปนี้ จะเห็นว่าโพรเซสไม่จำเป็นต้องเข้าที่บล็อก Evaluator แต่จะมีวิธีการหลีกเลี่ยงไปยังบล็อก Pixel Operation และไปยังบล็อก Per-Fragment Operation โดยตรง ถ้าเป็นอิมเมจ (Image) ในที่สุดแล้วเป็นสาเหตุให้บล็อกของพิเซลต่างๆ ถูกบันทึกลงในเฟรมบัฟเฟอร์ ค่าต่างๆจะถูกอ่านกลับจากเฟรมบัฟเฟอร์หรือถูกก็อปปี้จากส่วนหนึ่งของเฟรมบัฟเฟอร์ไปยังส่วนอื่นๆ อีกด้วย การโอนย้ายเหล่านี้จะรวมบางชนิดของการเข้ารหัสหรือถอดรหัส การเรียงลำดับตามลูกศรนี้มีความหมายเฉพาะเมื่อเป็นเครื่องมือสำหรับอธิบาย OpenGL ไม่ใช่เป็นกฎตายตัวของวิธีการที่ OpenGL ถูกสร้าง และเราแสดงเฉพาะเมื่อหมายถึงการจัดการทำงานหลายๆ อย่างของ OpenGL รูปทรงเรขาคณิตหรือวัตถุ เช่น พื้นผิวโค้ง จะถูกแปลงก่อนเป็นรูปหลายเหลี่ยม

บทที่ 3
การออกแบบ

หลักการออกแบบวงจร แบ่งออกเป็นส่วนหลักๆ ได้ 2 ส่วนคือ

3.1 ภาคส่งข้อมูล

3.2 ภาครับข้อมูล

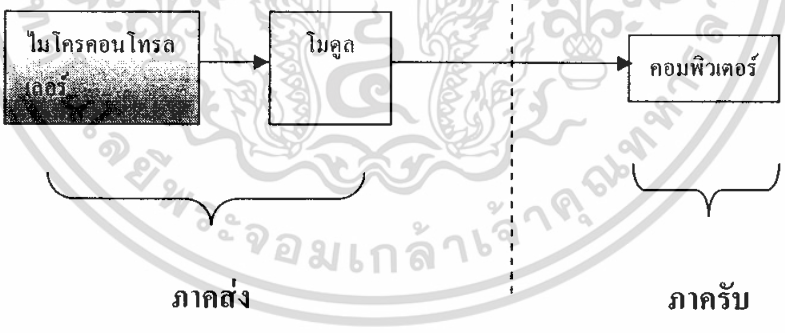
3.1 ภาคส่งข้อมูล

ในส่วนของภาคส่งนั้น จะแบ่งออกเป็น 2 ส่วน ซึ่งแบ่งออกเป็น

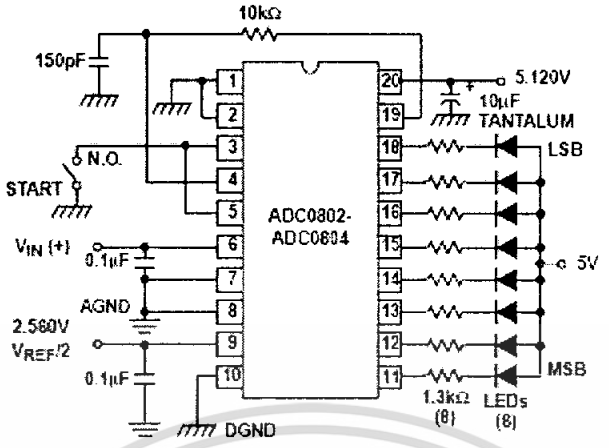
1.ส่วนของไมโครคอนโทรลเลอร์ ซึ่งจะเป็นส่วนที่ใช้จัดลำดับข้อมูล

2.ส่วนของโมดูล ซึ่งจะเป็นส่วนที่ใช้สำหรับเชื่อมต่อ USB ผ่าน ไปยังเครื่องคอมพิวเตอร์ เพื่อแสดงผลข้อมูลซึ่งโมดูลที่ใช้ คือ Ezy USB-M02 ของบริษัท แอสทรอน ลอจิก รีเสิร์ชแอนด์ ดิวิลอปเมนต์ จำกัด

3.1.1 ส่วนของไมโครคอนโทรลเลอร์



เริ่มจากการที่เราได้รับข้อมูลที่มาจากการตรวจจับของ sensor แล้ว จากนั้นสัญญาณ Pulse ที่อยู่ในรูปของความเร็วรถ และความเร็วรอบของรถนั้นจะเข้ามาสู่ microcontroller เพื่อตรวจวัดสัญญาณจากการกำหนด Timer ใน Microcontroller ให้เป็น counter และนำเอาระดับโวลต์ที่มาจาก sensor ที่แสดงถึงระดับน้ำมัน และระดับ อุณหภูมิ มาเพื่อเข้าสู่ A/D และจากนั้นสัญญาณจะอยู่ในรูปของดิจิตอล และเข้าสู่ Microcontroller A/D ที่เราได้ทำการเลือกใช้นั้น คือ เบอร์ ADC0804 ซึ่งจะให้การต่อตามรูปก่อนเข้าภาคของไมโครคอนโทรเลอร์ เพื่อทำการลำดับและเรียงข้อมูลต่อไป

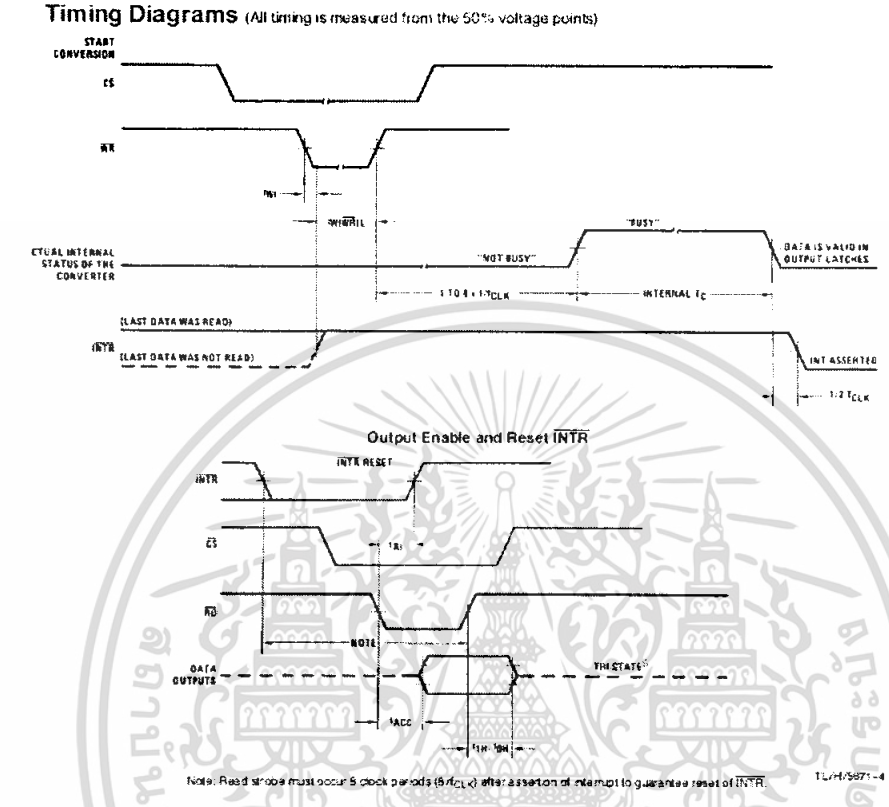


รูปที่ 3.1 ลักษณะการต่อ ADC0804 ก่อนเข้ากับไมโครคอนโทรเลอร์

3.1.2 ความหมายของขาต่างของ ADC 0804

- CS Chip Select (Active Low) ไมโครโปรเซสเซอร์จะต้องให้ค่า "0" กับขานี้ เมื่อต้องการติดต่อกับ ADC0805
- WR WRITE (Active Low) ถ้าสัญญาณเป็น "0" ADC0805 จะทำการ Sampling ค่ามาเก็บไว้
- RD READ (Active Low) ถ้าเป็นสัญญาณเป็น "0" ADC0804 จะส่งค่าที่ทำการเปลี่ยนแปลงเป็น Digital แล้วให้ไมโครคอนโทรเลอร์
- Vref/2 ต่อกับค่า Vref/2 ซึ่ง A/D จะใช้เพื่อนำไปเปรียบเทียบกับ input ซึ่ง ADC0804 จะใช้ 2.5 Vdc
- DB0-DB7 เป็นเอาท์พุทของข้อมูลต่อกับ DATA BUS ของไมโครคอนโทรเลอร์ โดย DB0 เป็น LSB (Least Significant Bit) และ DB7 เป็น MSB (Most Significant Bit)
- A GND และ D GND เป็น Ground ของวงจร
- Vin(+) และ Vin(-) เป็นขา Input
- CLK R และ CLK IN ต่อวงจรให้กำเนิดสัญญาณ Clock ของA/D

Timing Diagrm ของ ADC 0804

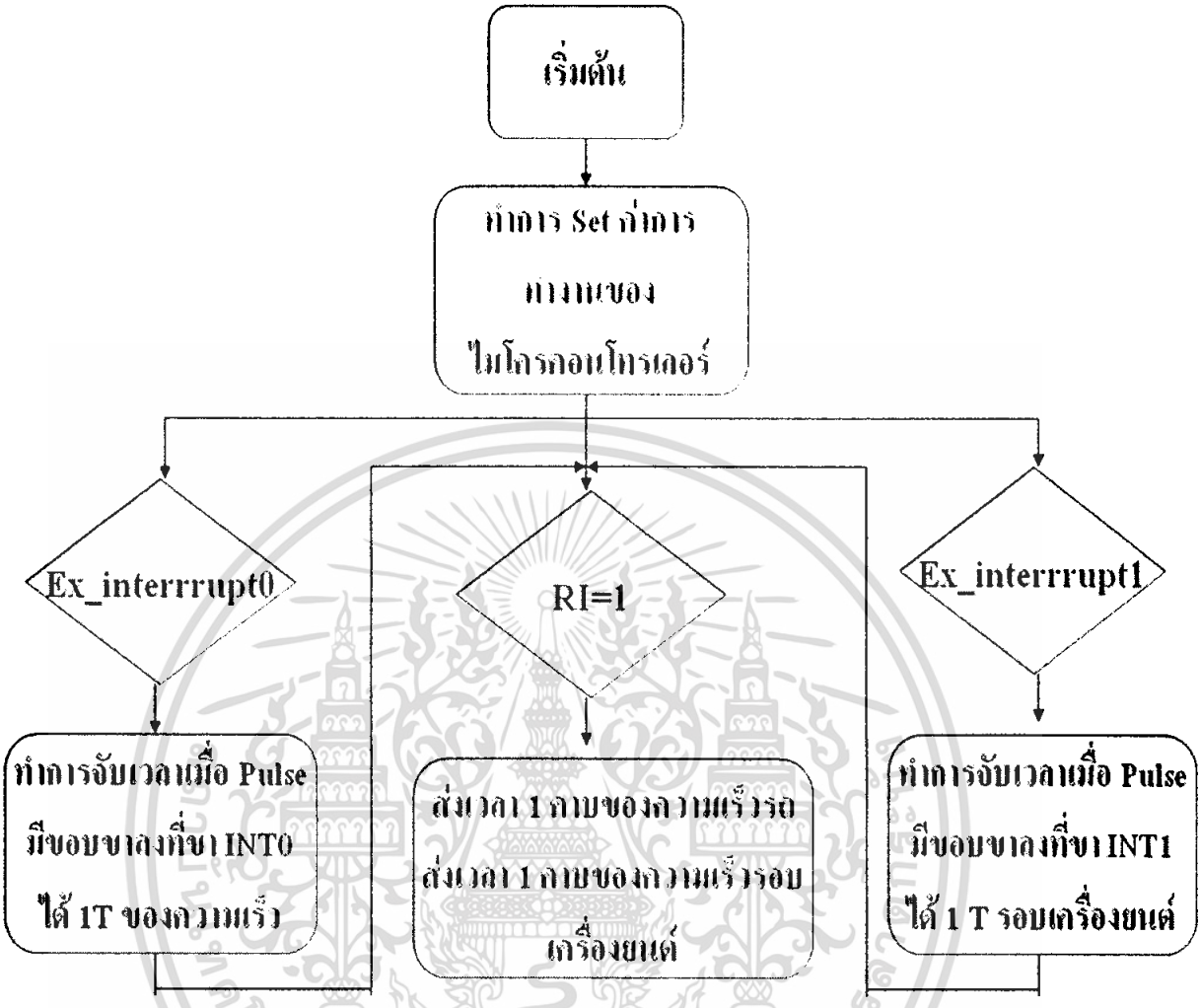


รูปที่ 3.2 Timing Diagram ของ ADC0804

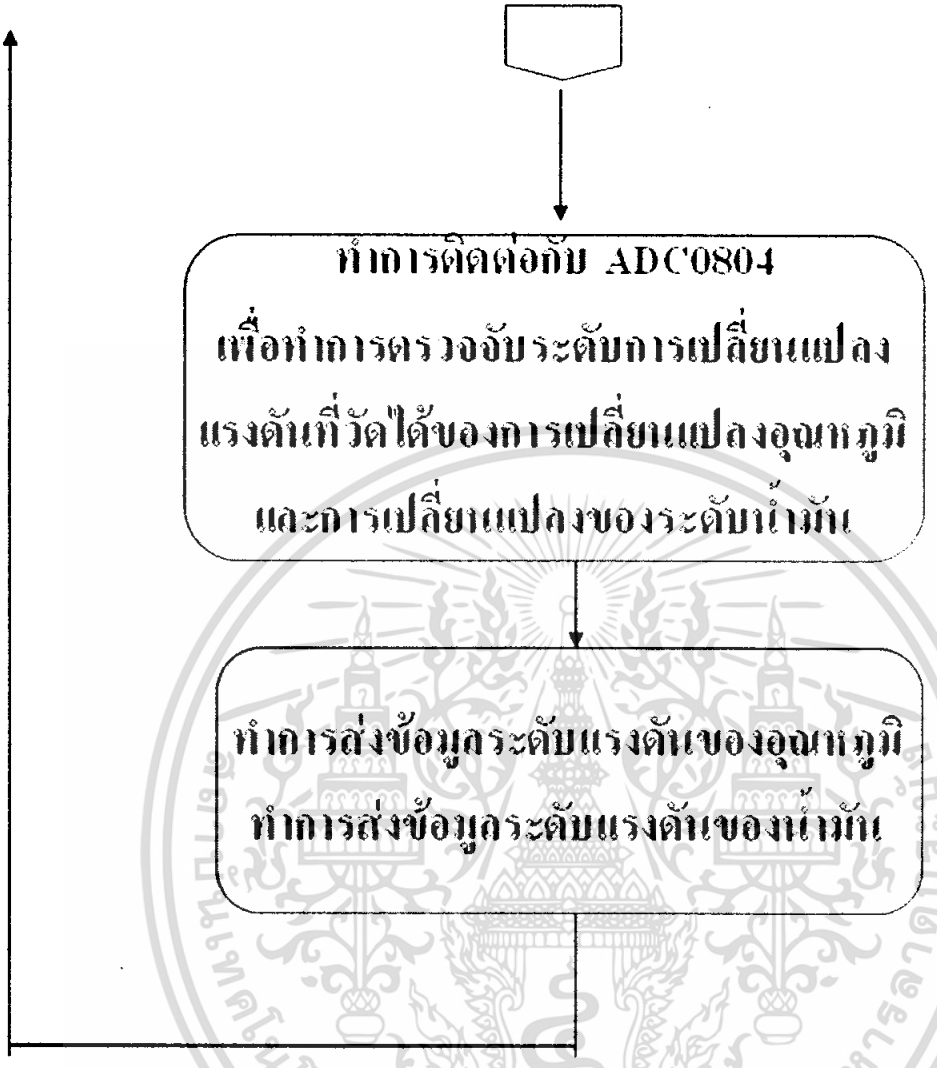
3.1.3 การต่อใช้งานของ ADC 0804

จากรูปที่ 3.1 ได้นำเอา ADC0804 ไปทำการต่อใช้งานโดยการใช้ไมโครคอนโทรเลอร์เป็นตัวควบคุมการทำงานของ A/D ซึ่งจะมีลักษณการทำงานเป็นไปตามนี้ คือ เมื่อมีการป้อนค่าให้ wr =0 rd =0 cs = 0

Intr =0 จะทำให้ adc 0804 มีการsampling ค่ามาเก็บไว้และจะได้ ข้อมูลที่เป็น digital ส่งให้กับไมโครคอนโทรเลอร์ต่อไป ซึ่งลักษณะการทำงานของการเก็บข้อมูลทั้งหมด เป็นไปตาม Flow chart ดังรูปที่ 3.3

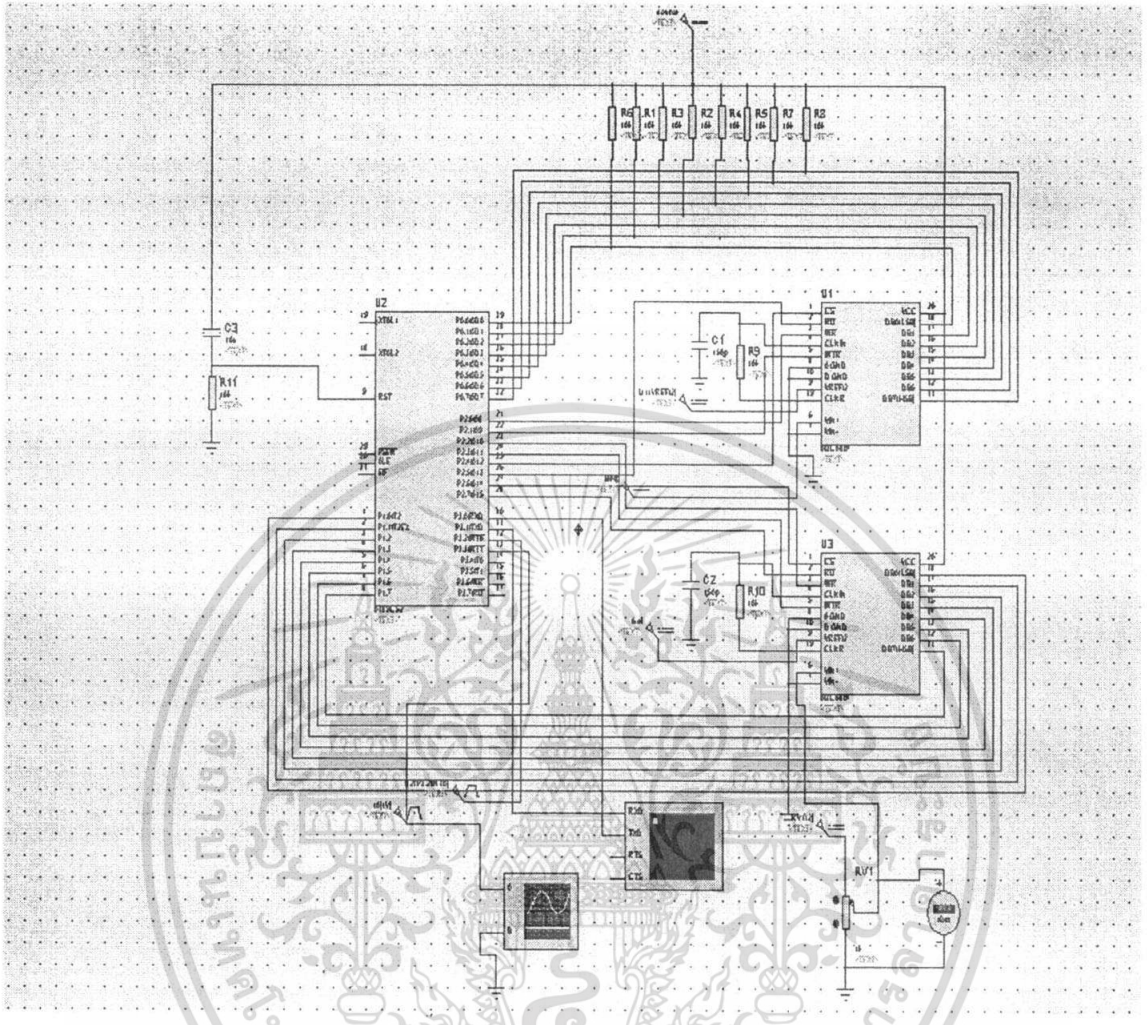


รูปที่ 3.3 Flowchart ของการทำงานของไมโครคอนโทรเลอร์



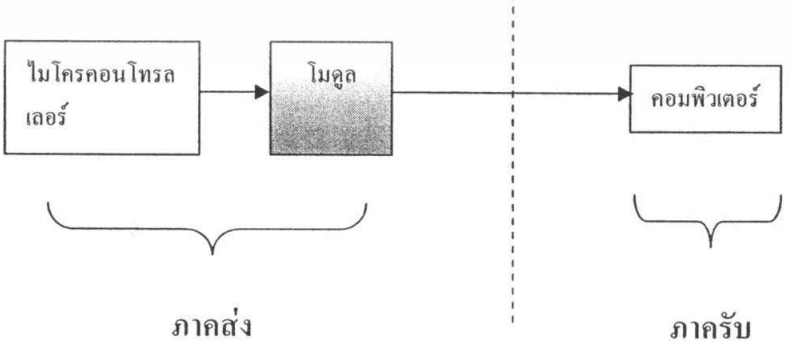
รูปที่ 3.3 Flowchart ของการทำงานของไมโครคอนโทรเลอร์

และหลังจากนั้นได้ทำการต่อวงจร ไมโครคอนโทรเลอร์เพื่อควบคุมการส่งข้อมูลดังรูปที่ 3.4



รูปที่ 3.4 วงจรส่วนควบคุมภาคการส่ง

ส่วนที่ 2 ของภาคส่ง



3.1.4 ลักษณะการทำงานของโมดูล

ในการส่งข้อมูลจากวงจรที่เชื่อมต่อกับโมดูล สามารถทำได้ในลักษณะเดียวกับการเชื่อมต่อแบบอนุกรม นั่นคือเราสามารถต่อขา RXD และ TXD ของโมดูลไปยังขา Tx และ Rx ของไมโครคอนโทรลเลอร์ได้โดยตรงโดยไม่จำเป็นต้องใช้วงจรแปลงระดับแรงดันแต่อย่างใด นอกจากนี้ยังสามารถใช้ขาแฮนด์เช็กต่างๆ ของโมดูลต่อเข้ากับวงจรภายนอกได้อย่างสมบูรณ์

เมื่อเราทำการตรวจจับ input ที่มาจาก ADC แล้วเราก็จะส่งให้ไมโครคอนโทรเลอร์เพื่อจัดลำดับข้อมูล จากนั้นก็จะเข้าสู่ภาคของการใช้โมดูลเพื่อที่จะทำการเชื่อมต่อซึ่งจะแสดงดังตารางดังรูปที่ 3.2

3.1.5 ข้อดีของ Ezy USB-MO2

1. สามารถทำอัตราการส่งถ่ายข้อมูลได้ถึง 1 เมกะบอดสำหรับการเชื่อมต่อแบบ RS232 และ 3 เมกะบอดสำหรับการเชื่อมต่อแบบ RS422,RS485
2. ภายในโมดูลสามารถจัดการโพโตคอล USB ได้อย่างสมบูรณ์ (ไม่ต้องการเฟิร์มแวร์สำหรับจัดการ USB เพิ่มเติม)
3. มีบัพเฟอร์ขนาด 384 ไบต์ สำหรับด้านส่ง (Tx) ลั 128 ไบต์ สำหรับด้านรับ (Rx)
4. มีฮาร์ดแวร์สำหรับช่วยในการทำแฮนด์เช็ก
5. สนับสนุนโหมดซัซเพนด์และรีซูมโดยผ่านทางขา SLEEP# และ RI#
6. สนับสนุนการจ่ายกำลังงานให้ตัวอุปกรณ์เมื่อต้องการกำลังงานสูงโดยใช้แหล่งจ่ายไฟจากบัส USB ผ่านทางขอ PWREN#
7. มีวงจรเร็คกูเรเตอร์ 3.3 V สำหรับจ่ายให้กับวงจรภายนอกที่ดึงกระแสไม่เกิน 5 mA
8. สามารถเชื่อมต่อวงจรลอจิกที่ใช้แรงดัน 5 โวลท์ และ 3.3 โวลท์ ได้อย่างสะดวก
9. มีวงจรรีเซ็ตขณะเริ่มทำการจ่ายไฟ (Power – On – Reset) อยู่ภายใน
10. สนับสนุนโหมดการส่งถ่ายข้อมูล USB แบบบัลก์และไอโซโครนัส
11. ทำงานที่ย่านไฟเลี้ยงตั้งแต่ 4.4 V ถึง 5.25 V
12. สนับสนุนโหมดการทำงานแบบบิตแบงค์ (Bit-Bang) ซึ่งอนุญาตให้ใช้บัสข้อมูลเป็นพอร์ต IO ขนาด8 บิตเพื่อใช้ในงานทั่วไปได้โดยไม่จำเป็นต้องใช้ไมโครคอนโทรเลอร์หรือวงจรลอจิกอื่น ๆ เพิ่มเติม
13. สนับสนุน USB 1.1 และ USB 2.0 (เฉพาะโหมดฟูลสปีด)
14. มี EEPROM ภายนอกสำหรับเก็บค่าพารามิเตอร์ต่าง ๆ ของ USB เช่นสตริงดีสคริปเตอร์ ซีเรียลนัมเบอร์,เวนเดอร์ไอดี(PID) และโพรดักซ์ไอดี (PID) ได้อย่างสะดวก
15. สามารถโปรแกรม EEPROM ที่อยู่บนโมดูลผ่านสาย USB ได้โดยตรง

16. มีจั้มเปอร์ช่วยอำนวยความสะดวกในการตั้งค่าการทำงานให้แก่โมดูล
17. เชื่อมต่อกับไมโครอคนโทรเลอร์หรือ FPGA ได้ง่าย

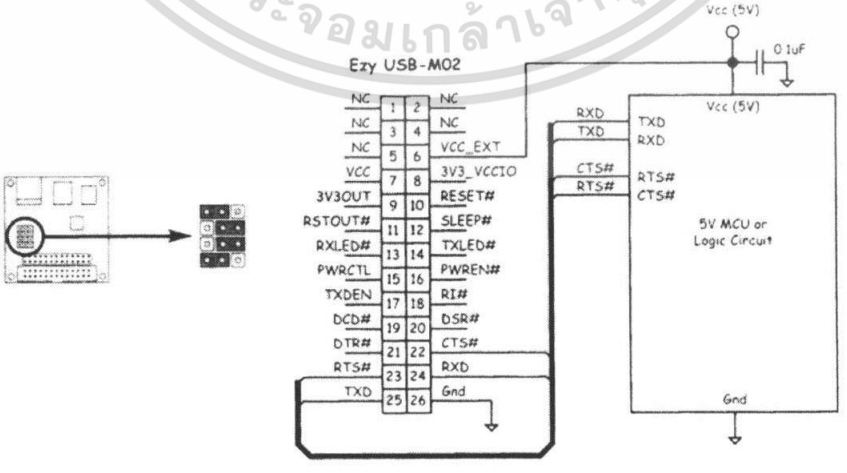
ตารางที่ 3.1 แสดงผลการทำงานของขาต่างๆของโมดูล

ขา	สัญญาณ	ชนิด	คำอธิบาย
1	NC	-	ไม่มีการต่อใช้งาน
2	NC	-	ไม่มีการต่อใช้งาน
3	NC	-	ไม่มีการต่อใช้งาน
4	NC	-	ไม่มีการต่อใช้งาน
5	NC	-	ไม่มีการต่อใช้งาน
6	VCC_EXT	แหล่งจ่ายไฟ	ใช้สำหรับต่อแหล่งจ่ายไฟจากภายนอกในกรณีที่ใช้งานแบบ self Power และ set จั้มเปอร์ J 1 ไปที่ VCCEXT
7	VCC	แหล่งจ่ายไฟ	ใช้สำหรับต่อแหล่งจ่ายไฟจากระบบบัสเพื่อจ่ายไฟให้แก่ วจรภายนอก ในกรณีที่ใช้งานแบบ Bus Powered และเซ็ตจั้มเปอร์ J1 ไปที่ VCCBUS
8	3VC_VCCIO	แหล่งจ่ายไฟ	ใช้สำหรับต่อกับแรงดัน +3.3 V ในกรณีที่เชื่อมต่อกับวงจรภายนอก ที่ใช้แรงดัน 3.3 V และเซ็ตจั้มเปอร์ J 4 ไปที่ VIOEXT
9	3V3OUT	เอาท์พุท	จ่ายเอาท์พุท 3.3 V ให้กับอุปกรณ์ลอจิกภายนอก ที่ใช้แรงดัน 3.3 V และดึงกระแสไม่มากนัก (น้อยกว่าหรือเท่ากับ 5 mA)
10	RESET#	อินพุท	ใช้สำหรับให้อุปกรณ์ภายนอกทำการรีเซ็ตโมดูลในขณะเริ่มทำงาน หากไม่ต้องการใช้ ให้เซ็ตจั้มเปอร์ J2 ไปที่ RENSNOR
11	RSTOUT#	เอาท์พุท	เป็นเอาท์พุทของวงจรกำเนิดสัญญาณรีเซ็ต ขานี้จะอยู่ในสภาพอิมพีแดนซ์สูงประมาณ 2มิลลิวินาที ภายหลังจากที่ VCC มีค่ามากกว่า 3.5 โวลท์ และวงจรสัญญาณนาฬิกาภายในเริ่มทำงาน จากนั้นมันจะยกระดับเอาท์พุทของมันขึ้นไปที่ 3.3 โวลท์
12	SLEEP#	เอาท์พุท	มีค่าเป็นลอจิกต่ำในระหว่างที่อยู่ในโหมดซีซเพนด์ ตามปกติจะใช้สำหรับตัดการจ่ายไฟให้แก่วงจร
13	RXLED#	เอาท์พุท	เอาท์พุทแบบคอลเล็คเตอร์เปิดใช้สำหรับขับ LED แสดงผลการรับข้อมูลของโมดูล
14	TXLED#	เอาท์พุท	เอาท์พุทแบบคอลเล็คเตอร์เปิดใช้สำหรับขับ LED แสดงผลการส่งข้อมูลของโมดูล

ขา	สัญญาณ	ชนิด	คำอธิบาย
15	PWRCTL	อินพุท	ใช้สำหรับกำหนดโหมดการใช้พลังงาน - ต่อกับระดับลอจิกต่ำในกรณีที่เป็นแบบ Bus Powered - ต่อกับระดับลอจิกสูงในกรณีที่เป็นแบบ Self Powered แต่ขานี้ไม่จำเป็นต้องต่อกับภายนอกเนื่องจากภายในโมดูลมีจั้มปอร์ J3 สำหรับกำหนดระดับแรงดันไว้แล้ว ซึ่งการเซ็ต J3 สามารถทำได้ดังนี้ - เซ็ต J3 ไปที่ PCSELF ในการณีที่เป็นการใช้งานแบบ Self Powered - เซ็ต J3 ไปที่ PCBUS ในกรณีที่เป็นการใช้งานแบบ Bus Powered
16	PWREN#	เอาท์พุท	- มีค่าเป็นลอจิกต่ำหลังจากที่ตัวอุปกรณ์ได้ทำการคอนฟิกผ่านสาย USB และจะมีค่าลอจิกสูงในขณะที่โฮสเข้าสู่โหมด Suspend - ใช้ควบคุมกำลังงานของอุปกรณ์ลอจิกภายนอกได้โดยใช้MOSFET แบบ P-Channel
17	TXDEN	เอาท์พุท	ใช้สำหรับเปิดการส่งข้อมูล RS485
18	RI#	อินพุท	UART - ขาสัญญาณ Ring Indication Control ในกรณีที่เปิดใช้ตัวเลือกรีโมทเวคอัพใน EEPROM ไว้ ขานี้สามารถใช้ในResume โฮสได้โดยการป้อนลอจิกต่ำให้แก่มัน
19	DCD#	อินพุท	UART - ขาสัญญาณ Data Carrieer Detect
20	DSR#	เอาท์พุท	UART - ขาสัญญาณ Data set Ready Control
21	DTR#	อินพุท	UART - ขาสัญญาณ Data Teminal Ready Control
22	CTS#	อินพุท	UART - ขาสัญญาณ Clear To Send Control
23	RTS#	เอาท์พุท	UART - ขาสัญญาณ Request To Send Control
24	RXD	อินพุท	UART – ขารับข้อมูล
25	TXD	เอาท์พุท	UART – ขาส่งข้อมูล
26	GND	แหล่งจ่ายไฟ	กราวด์

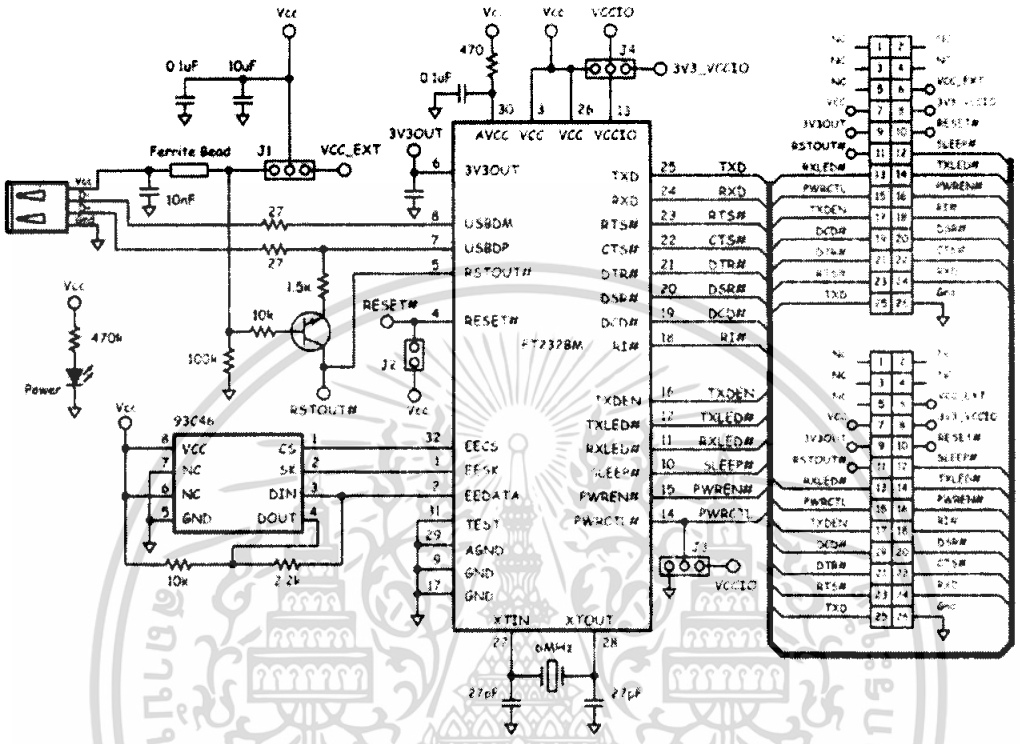
การเชื่อมต่อของโมดูล

การเชื่อมต่อแบบใช้กำลังงานจากตัวเอง



รูปที่ 3.5 การเชื่อมต่อแบบใช้กำลังงานจากตัวเองและการเซ็ตจั๊มเปอร์

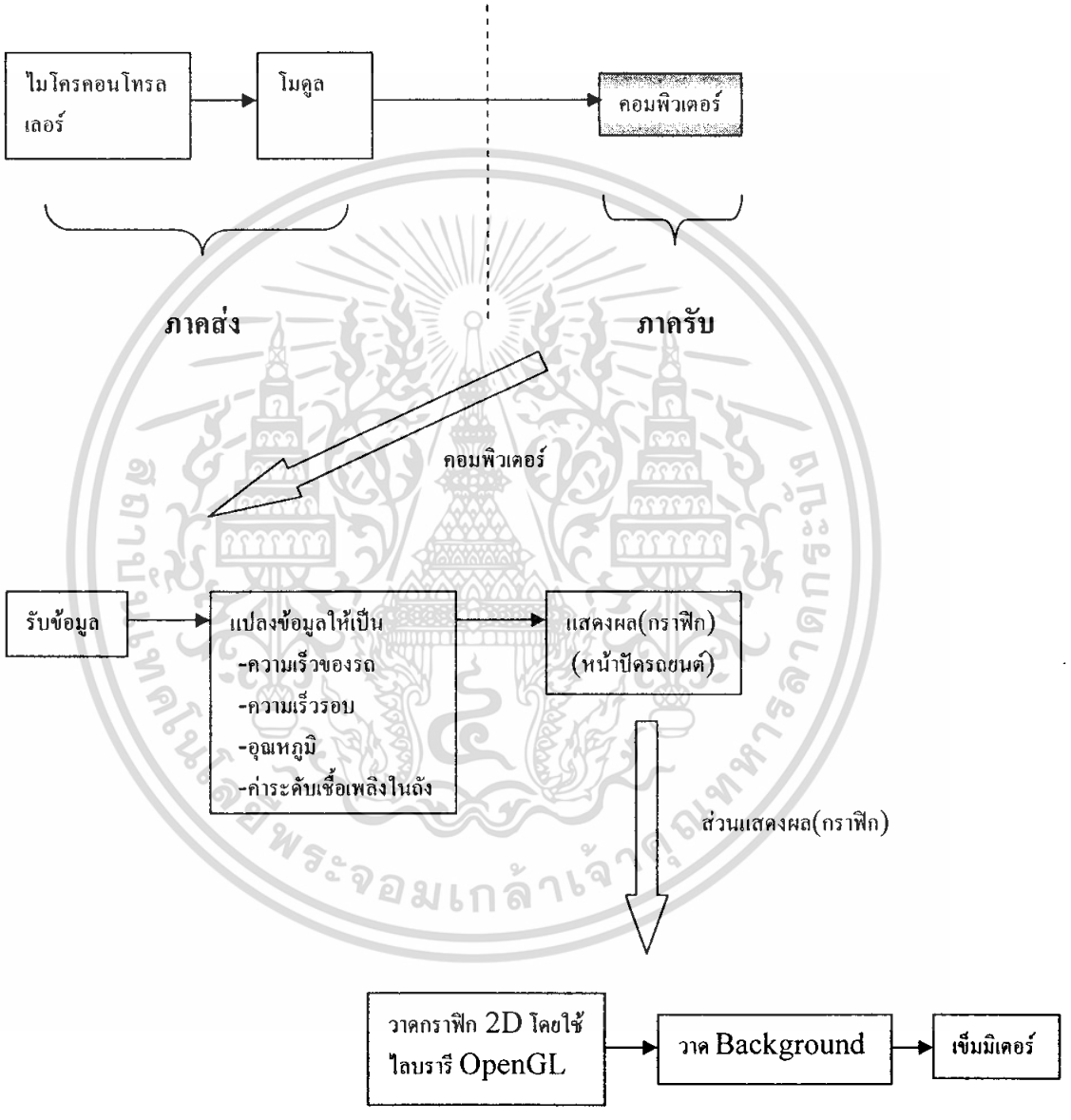
วงจรภายในของ Ezy USB-M02



รูปที่ 3.6 วงจรภายในของ Ezy USB-M02

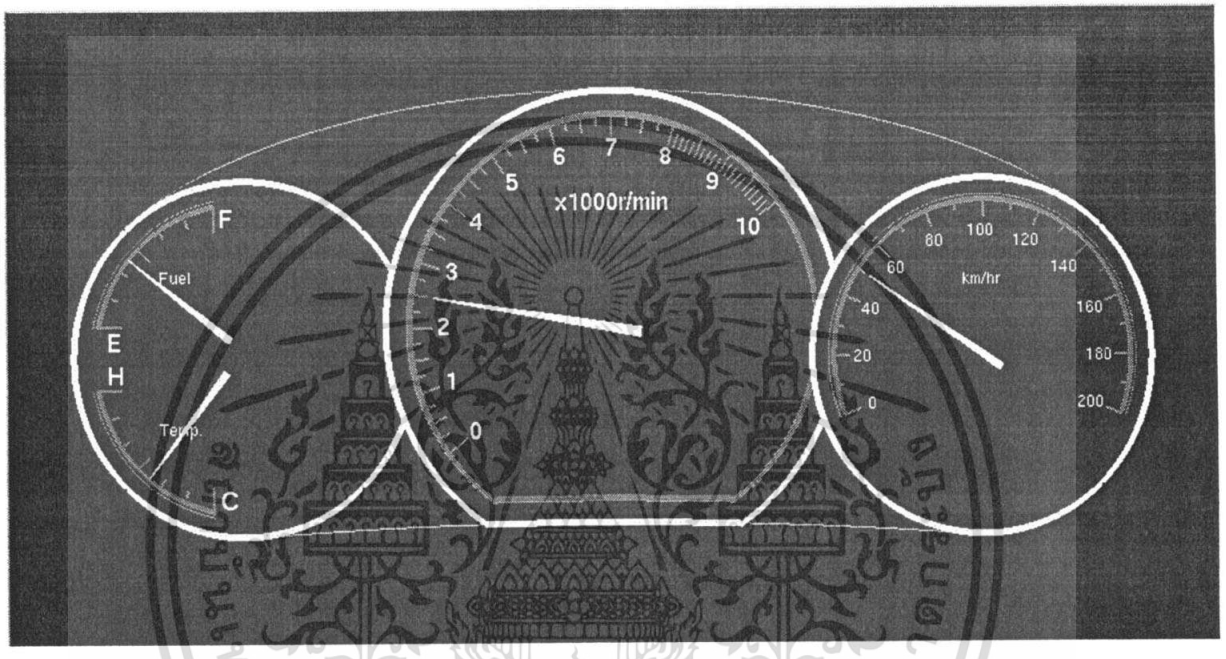
3.2 ภาครับข้อมูล

ทำหน้าที่รับข้อมูลที่เข้ามาจากภาคส่งเพื่อนำข้อมูลมาแสดงผลบนหน้าจอคอมพิวเตอร์ เพื่อให้แสดงผลความเร็วของรถยนต์ ความเร็วรอบ อุณหภูมิ และค่าระดับเชื้อเพลิงในถังเชื้อเพลิง ซึ่งมีบล็อกไดอะแกรมโดยรวมดังรูป



รูปที่3.7 บล็อกไดอะแกรมในการทำงานของภาครับ

ส่วนโปรแกรมที่ใช้เขียนได้ใช้โปรแกรม Microsoft Visual Studio เขียนโปรแกรมสำหรับรับข้อมูล แปลงข้อมูลที่ได้แล้วนำมาคำนวณเพื่อนำไปแสดงผลในส่วนของกราฟิก ซึ่งส่วนเขียนกราฟิก2Dได้ใช้ Library OpenGL ในการวาดรูปหน้าปัดรถยนต์และซึ่งได้ออกแบบหน้าปัดมีลักษณะดังรูปที่ 3.8



รูปที่ 3.8 รูปหน้าปัดรถยนต์

3.2.1 แนวคิดในการออกแบบในส่วนรับข้อมูลและประมวลผล

แบ่งข้อมูลที่จะให้ไมโครคอนโทรลส่งมาให้เป็น 4 ชุด คือ

- ข้อมูลของความเร็วรถยนต์ (km/hr) ข้อมูลจากความเร็วรถยนต์นี้จะวัดได้เป็น ลูกคลื่น (pulse) ซึ่งไมโครคอนโทรลเลอร์จะนับลูกคลื่นนี้แล้วส่งมายังภาครับ(คอมพิวเตอร์)
- ข้อมูลของความเร็วรอบเครื่อง (rpm) ข้อมูลของความเร็วรอบเครื่องจะวัดได้เป็นลูกคลื่น (pulse)

ซึ่งไมโครคอนโทรลเลอร์จะนับลูกคลื่นนี้แล้วส่งมายังภาครับ(คอมพิวเตอร์)

- ข้อมูลของระดับน้ำมัน (Fuel) ข้อมูลของระดับน้ำมันนี้จะเป็นระดับแรงดันไฟตรง 12V ซึ่งใช้ ADC แปลงแล้วให้ไมโครคอนโทรลเลอร์ส่งมายังภาครับ(คอมพิวเตอร์)
- ข้อมูลของอุณหภูมิ (Temperature) ข้อมูลของอุณหภูมินี้จะเป็นระดับแรงดันไฟตรง 12V ซึ่งใช้ ADC แปลงแล้วให้ไมโครคอนโทรลเลอร์ส่งมายังภาครับ(คอมพิวเตอร์)

การทำงานนั้นเริ่มโดยให้ภาครับ(คอมพิวเตอร์)ทำการส่ง 0xFF ให้ไมโครคอนโทรลเลอร์ เพื่อให้ไมโครคอนโทรลเลอร์รู้ว่าคอมพิวเตอร์พร้อมที่จะรับข้อมูลและให้ส่งข้อมูลกลับมา โดยแต่ละชุดข้อมูลที่ไมโครคอนโทรลเลอร์จะส่งจะมีเลขกำกับนำหน้าข้อมูลดังนี้

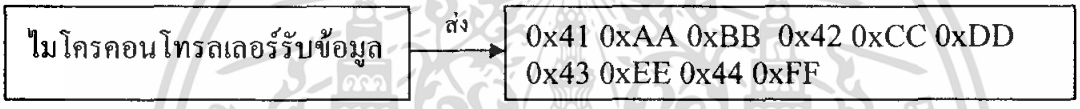
ให้ส่ง 0x41 (8 bit) ตามด้วย ข้อมูลของความเร็วรถยนต์ (16 bit) (ให้0xAA 0xBB แทนข้อมูล)และ

ส่ง 0x42 (8 bit) ตามด้วย ข้อมูลของความเร็วรอบเครื่อง (16 bit) (ให้0xCC 0xDDแทนข้อมูล) และ

ส่ง 0x43 (8 bit) ตามด้วย ข้อมูลของระดับน้ำมัน (8 bit) (ให้0xEEแทนข้อมูล) และ

ส่ง 0x44 (8 bit) ตามด้วย ข้อมูลของอุณหภูมิ(ให้0xFFแทนข้อมูล) (8 bit)

จะมีลักษณะดังรูป



3.2.1.1 การแปลงข้อมูลที่รับมาจากไมโครคอนโทรลเลอร์

เมื่อไมโครคอนโทรลเลอร์ส่งข้อมูลมาแล้ว เราจะต้องแยกเอาข้อมูลของแต่ละชุดออกมาและแปลงให้เป็นข้อมูลตามที่เราต้องการ ซึ่งเป็นดังนี้

- ข้อมูลของความเร็วรถยนต์และข้อมูลของความเร็วรอบเครื่อง ที่ได้รับมาจากไมโครคอนโทรลเลอร์นั้นจะเป็นข้อมูล 16 บิตของไทม์เมอร์ ดังนั้นเราจะต้องนำเอาข้อมูลที่ได้มาแปลงเป็นความถี่(Hz) โดยเข้าสมการ

$$\text{ความถี่(Hz)} = \frac{10^6}{1.084 \times \text{ข้อมูล16บิตที่แปลงเป็นฐาน10}}$$

แล้วจึงนำเอาข้อมูลที่แปลงได้ไปทำการคำนวณหาความเร็วรถยนต์และความเร็วรอบเครื่องต่อไป แล้วนำไปแสดงผล

- ส่วนข้อมูลของระดับน้ำมันและอุณหภูมิ ที่ได้รับมาจากไมโครคอนโทรลเลอร์จะเป็นระดับแรงดันไฟตรง 12 V ซึ่งจะแปลงให้เหลือ 5V ก่อนที่จะเข้า ADC แล้วให้ ADC

แปลงออกมาแล้วส่งให้ไมโครคอนโทรลเลอร์ แล้วไมโครคอนโทรลเลอร์จะส่งมายังภาครับ (คอมพิวเตอร์) อีกทีหนึ่ง ข้อมูลที่รับมาจากไมโครคอนโทรลเลอร์นั้นเราจะต้องนำมาแยก 4 บิตบนกับ 4 บิตล่าง เช่น 0x2Aแยกเป็น 2 กับ 10 แล้วนำไปคำนวณต่อโดยใช้สมการ

$$\text{แรงดันไฟตรง(V)} = \left(\frac{\text{4บิตบน}}{16} + \frac{\text{4บิตล่าง}}{256} \right) \times 5.12$$

ซึ่งจะได้ออกมาเป็นระดับแรงดันไฟตรง 0-5 V แล้วแปลงเป็นแรงดันไฟตรง 12 V ต่อไป แล้วนำไปแสดงผล

3.2.1.2 การคำนวณข้อมูลทั้ง 4 ที่ได้จากการแปลงข้อมูลแล้วเพื่อให้เป็นข้อมูลของ ความเร็วรถยนต์ ความเร็วรอบเครื่อง ระดับน้ำมัน และ อุณหภูมิ

ซึ่งจากข้อมูลของที่แปลงแล้วนั้น ความเร็วรถยนต์และความเร็วของรอบเครื่อง จะได้เป็นความถี่ และ ระดับน้ำมันและอุณหภูมิ จะได้เป็นระดับแรงดันไฟตรง 0-5 V เราจะต้องนำมาคำนวณต่อดังนี้

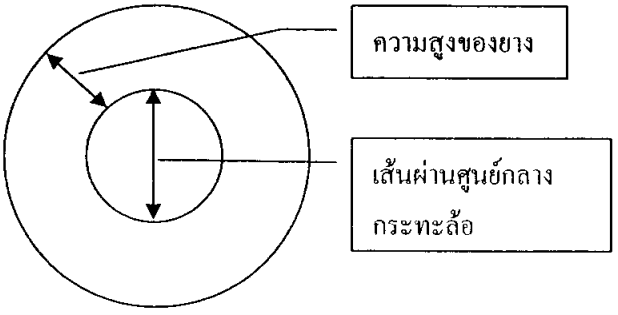
-ข้อมูลความเร็วรถยนต์ หลังจากที่เราได้ข้อมูลที่แปลงเป็นความถี่(Hz)แล้ว เราจะต้องนำค่านี้มาคำนวณต่อเพื่อหาความเร็วของรถยนต์และเนื่องจากสูตรคำนวณความเร็วคือ ผลคูณระหว่างเวลากับระยะทาง ซึ่งในการหาเวลานี้ทำได้โดยนำค่าความถี่นี้มาแปลงมาเป็นคาบเวลา(เศษส่วนกลับของความถี่นั้นเอง) ส่วนระยะทางนั้นก็คือการหาเส้นรอบวงของยางรถยนต์ที่ทำให้เกิดลูกคลื่น 1 ลูกคลื่น ฉะนั้นเราต้องรู้ถึงรายละเอียดเกี่ยวกับการคำนวณยางของรถยนต์บ้าง เช่น

185/65 R 14 หมายถึงอะไร

185 ตัวเลขตัวแรกนี้บอกถึง ขนาดความกว้างของยาง หน่วยเป็นมิลลิเมตร(mm.)

65 ตัวเลขตัวที่สองนี้บอกถึง %ของความสูงของยางของความกว้างยาง 1 ด้าน หน่วย มิลลิเมตร(mm.)

14 ตัวเลขตัวสุดท้ายบอกถึง เส้นผ่านศูนย์กระทะล้อ หน่วยเป็นนิ้ว (inch)



รูปที่ 3.9 ล้อรถยนต์

ดังนั้นจากตัวอย่างข้างต้น 185/65 R 14 เราสามารถคำนวณหาเส้นรอบวงของยาง ได้เพื่อนำไปใช้ในการคำนวณหาความเร็วของรถยนต์ จะได้ดังนี้

ขนาดความกว้างยาง = 185 mm.

ความสูงของแก้มยาง 65 % ของความกว้างยาง (185) = 120.25 mm.

เส้นผ่านศูนย์กลางกระทะล้อ = 14 inch = 355.6 mm.

หาเส้นผ่านศูนย์กลางของล้อทั้งหมด จะได้

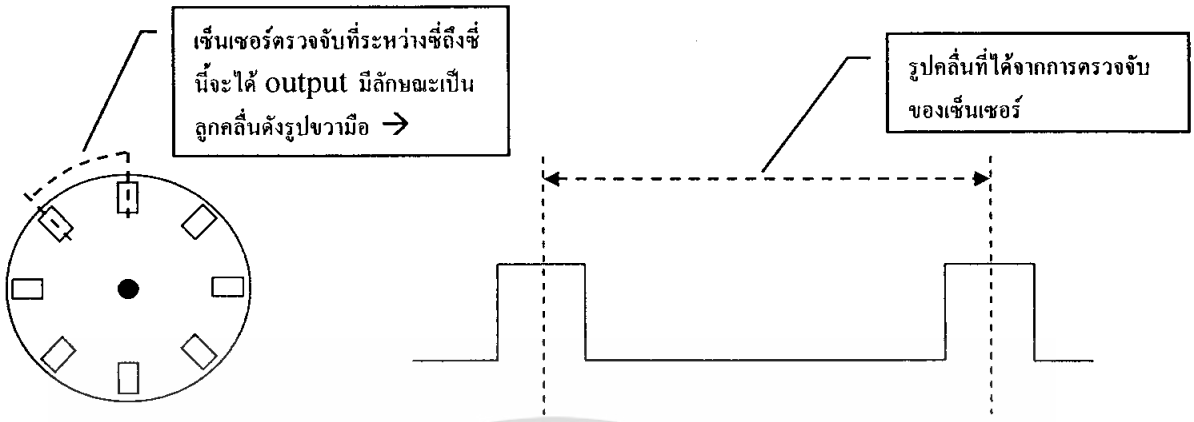
เส้นผ่านศูนย์กลางของล้อทั้งหมด = (ความสูงของยาง × 2) + เส้นผ่านศูนย์กลางกระทะล้อ

= (120.25 × 2) + (355.6) = 596.1 mm.

ดังนั้น จะได้เส้นรอบวงของล้อทั้งหมด $= 2\pi R$

$= 591.1 \times \pi = 1873.45$ mm.

เมื่อเรารู้เส้นรอบวงแล้ว สิ่งที่เราต้องรู้อีกอย่างหนึ่งก็คือ จำนวนซี่ที่เซ็นเซอร์ตรวจจับว่าใน 1 ล้อมีกี่ซี่



รูปที่ 3.10 ระยะระหว่างซี่ของวงล้อที่ใช้ในการตรวจจับของเซ็นเซอร์และสัญญาณเอาท์พุท ที่เซ็นเซอร์วัดได้ระหว่างซี่

สมมติให้ 1 ล้อ มี 40 ซี่ ฉะนั้นระยะห่างระหว่างซี่ถึงซี่จะ = เส้นรอบวง ÷ จำนวนซี่ และจากความถี่ที่ได้นำมาหาคาบเวลาจะได้ = 1÷ ความถี่ (จะได้ออกมาเป็นเวลาใน 1 ลูกคลื่น (pulse)) ซึ่งนั้นก็คือเวลาที่เซ็นเซอร์ตรวจจับซี่ต่อซี่นั้นเอง เราจะสามารถหาความเร็วได้โดย

สมการความเร็ว : $V = \frac{S}{T}$

โดยที่ V คือ ความเร็วของรถยนต์

S คือ ระยะทางของซี่หนึ่งถึงอีกซี่หนึ่ง(ระยะทางที่ทำให้เกิดลูกคลื่น 1 ลูกคลื่น)

T คือ เวลาใน 1 ลูกคลื่น (1 pulse)

เมื่อได้คำนวณได้ความเร็วของรถยนต์ออกมาแล้วก็นำค่าที่ได้ไปแสดงผลบนกราฟิกในส่วนของความเร็วรถยนต์

-<u>ข้อมูลความเร็วรอบเครื่องยนต์</u> หลังจากที่เราได้ข้อมูลที่แปลงเป็นความถี่(Hz)แล้ว เราจะต้องนำค่านี้คูณกับ 2 แล้วแปลงให้เป็น รอบ/นาที ก็จะได้เป็นข้อมูลความเร็วรอบเครื่องยนต์แล้วนำไปแสดงผลกราฟิกในส่วนของความเร็วรอบเครื่องยนต์

-<u>ข้อมูลระดับน้ำมัน</u> หลังจากที่เราได้ข้อมูลที่อยู่ในช่วงแรงดันไฟตรง 0-5 V แล้วเราจะต้องแปลงให้อยู่ในระดับแรงดันไฟตรง 0-12 V แทนแล้วนำไปแสดงผลบนกราฟิกในส่วนของ Fuel

-<u>ข้อมูลอุณหภูมิ</u> หลังจากที่เราได้ข้อมูลที่อยู่ในช่วง 0-5 V แล้วเราจะต้องแปลงให้อยู่ในระดับ 0-12 V แทน เช่นเดียวกับข้อมูลระดับน้ำมันแล้วนำไปแสดงผลบนกราฟิกในส่วนของ Temp.

3.2.2 แนวคิดในการออกแบบกราฟิกหน้าปัดแสดงผล นั้นจะแบ่งเป็นส่วนออกเป็น

1. ส่วนBackground ซึ่งส่วน Background นี้ จะแบ่งออกเป็น

   1.1 ส่วนของวงกลมและวงรี

   ในส่วนนี้จะใช้สมการวงกลม,วงรี กับการตัดกันของวงกลม,วงรี ในการวาดและใช้ฟังก์ชั่นใน Library OpenGL ลากเส้นเชื่อม

   1.2 ส่วนของเส้นสเกล

   ลักษณะในการออกแบบจะใช้สมการวงกลมเป็นหลัก

   1.3 ส่วนของอักษรบนมิเตอร์

   ซึ่งในการเขียนอักษรนั้นจะใช้ฟังก์ชั่นที่มีให้ใน Library OpenGL

2. ส่วนของเข็มมิเตอร์

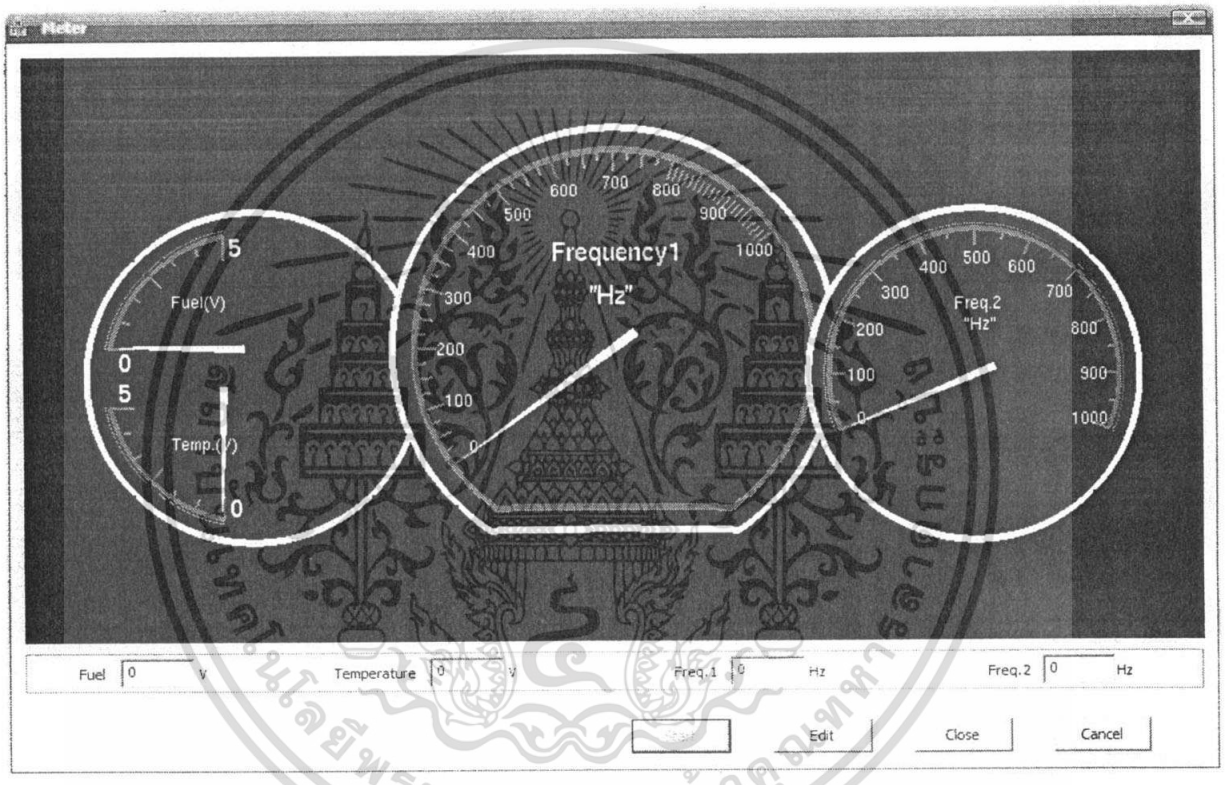
   จะใช้แนวคิดจากเส้นตรงที่มีจุดหนึ่งอยู่ที่จุดศูนย์กลางตลอด และ อีกจุดหนึ่งอยู่บนเส้น รอบวงของวงกลมซึ่งจะเป็นจุดที่เปลี่ยนค่า และเมื่อลากเส้นเชื่อมระหว่างจุด 2 จุดจะได้

   เป็นเข็มมิเตอร์ ซึ่งจุดที่อยู่บนเส้นรอบวงจะหาได้จาก $X = r \times \cos(A)$ และ $Y = r \times \sin(A)$ ซึ่ง r คือรัศมี(ซึ่งก็คือความยาวของเข็มนั่นเอง) ส่วน A คือมุม

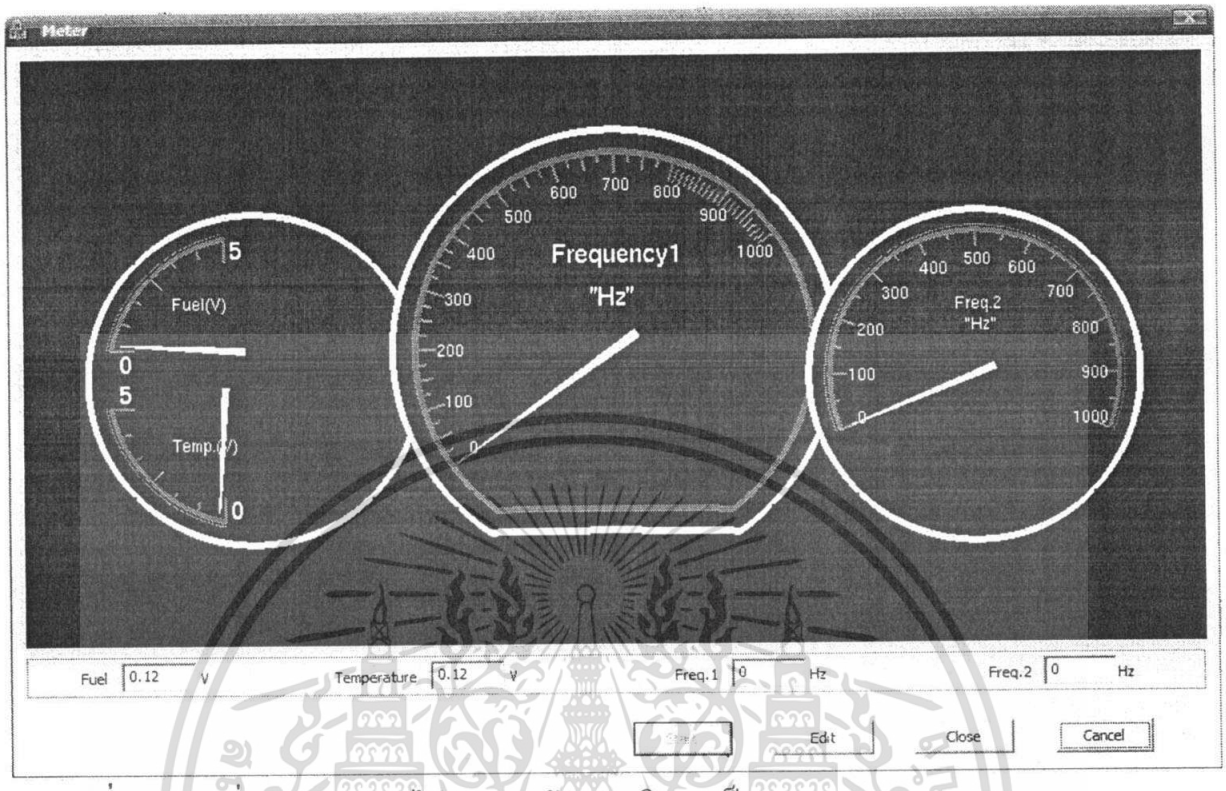
บทที่ 4

ผลการทดลอง

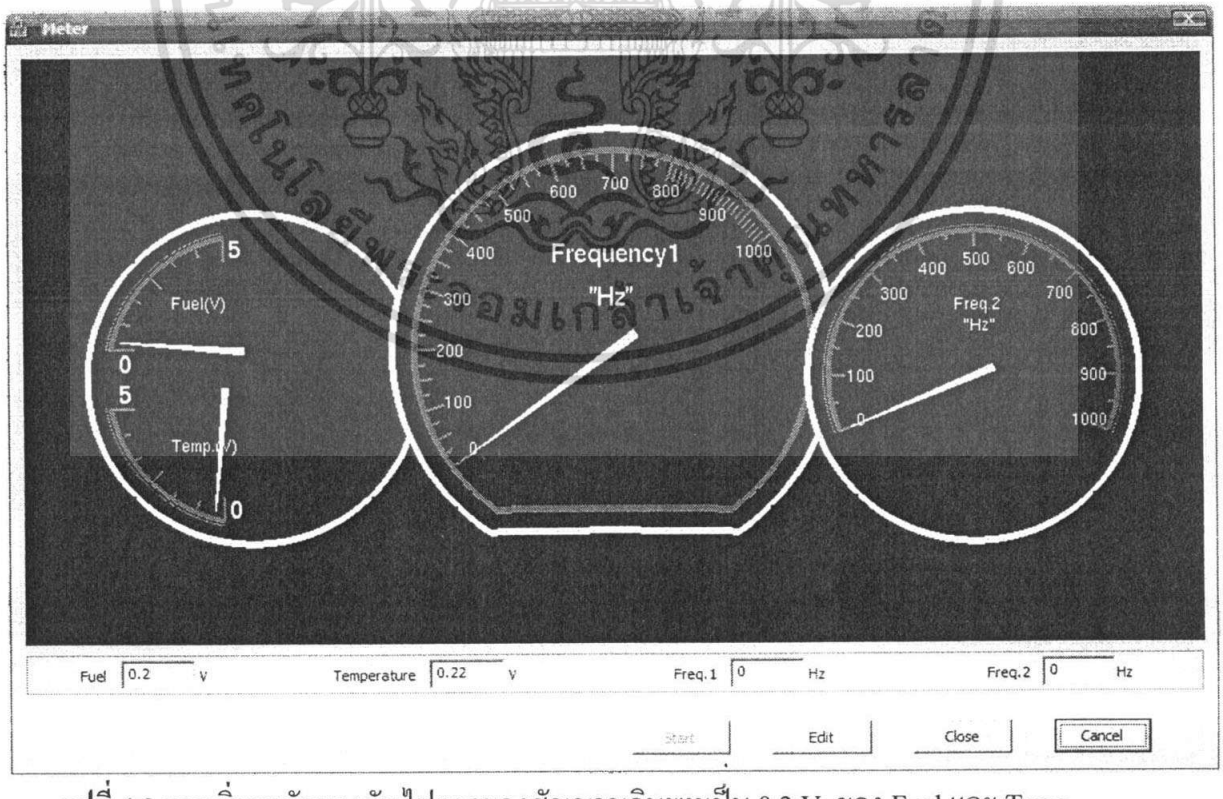
4.1การทดลอง จำลองวัดแรงดันไฟตรงจากแหล่งจ่ายไฟตรงแทนซึ่งแทนข้อมูลของระดับน้ำมันและอุณหภูมิ



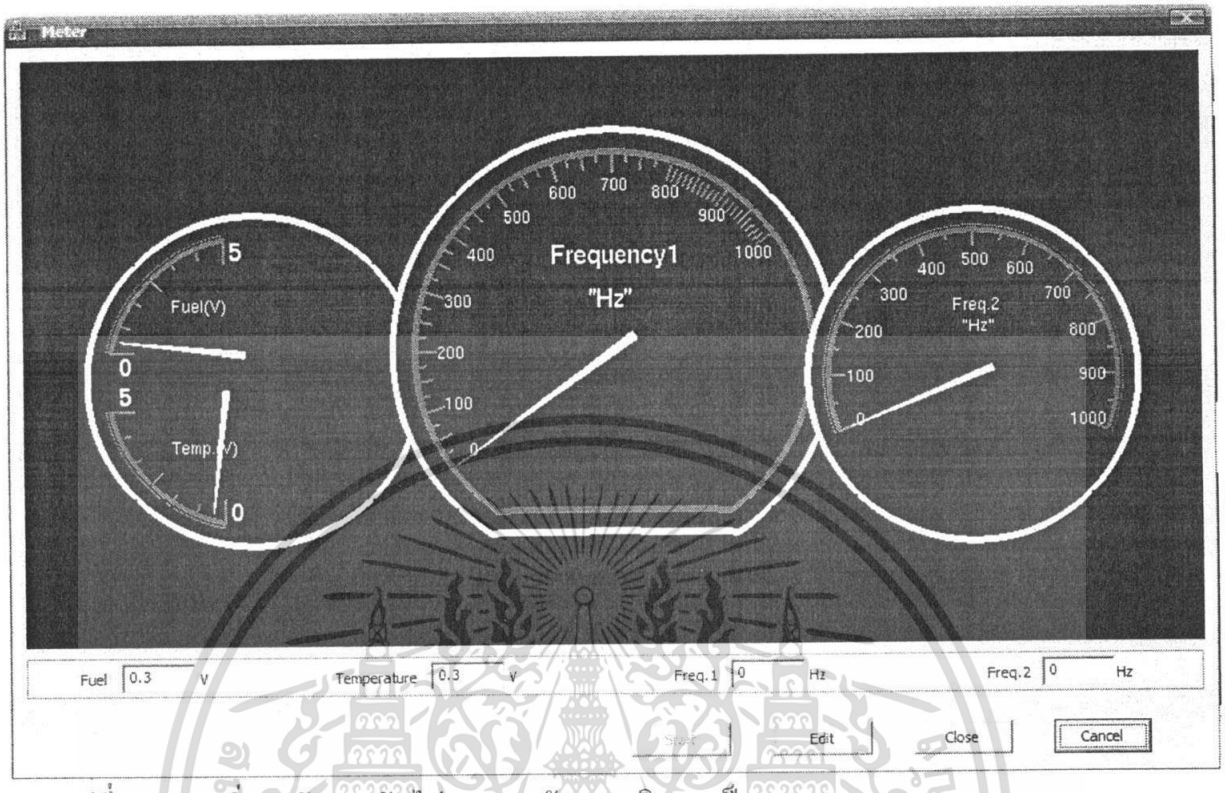
รูปที่ 4.1 ฟังก์ชั่นขณะเริ่มต้น เมื่อระดับแรงดันไฟตรงเท่ากับ 0 V



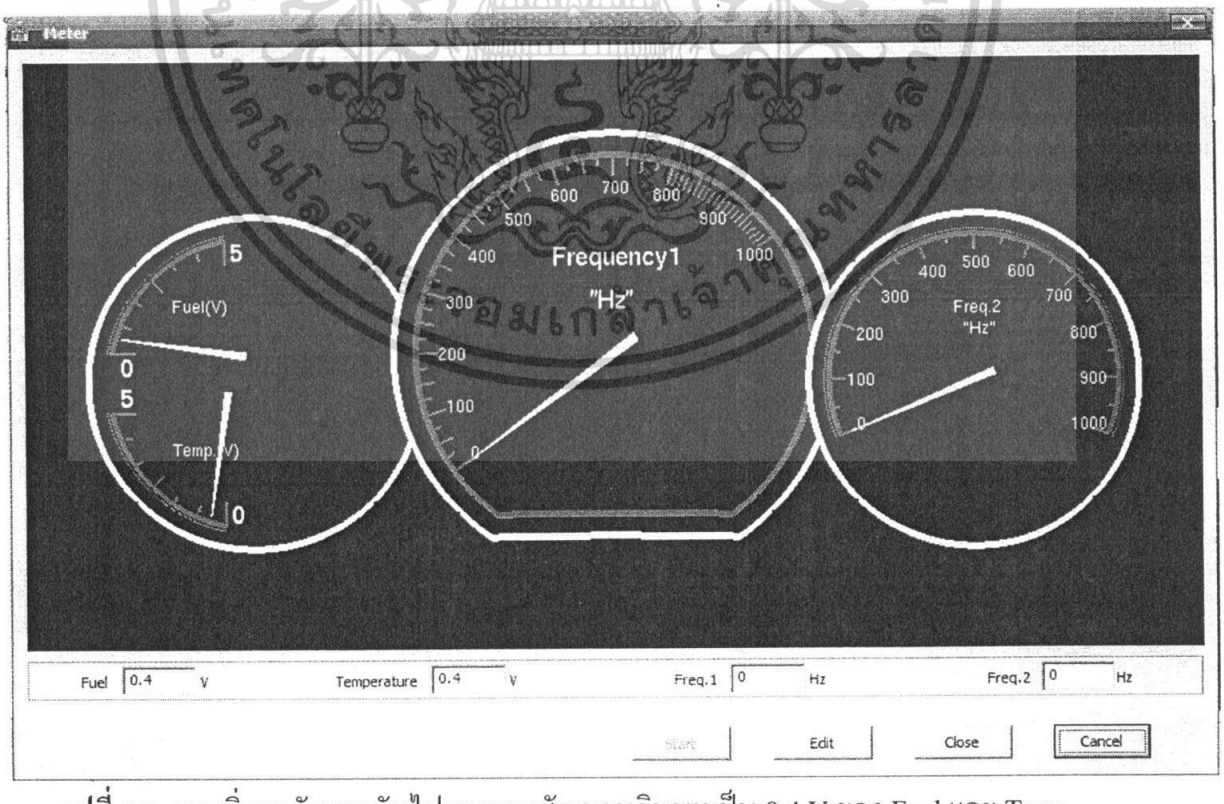
รูปที่ 4.2 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 0.1 V ของ Fuel และ Temp.



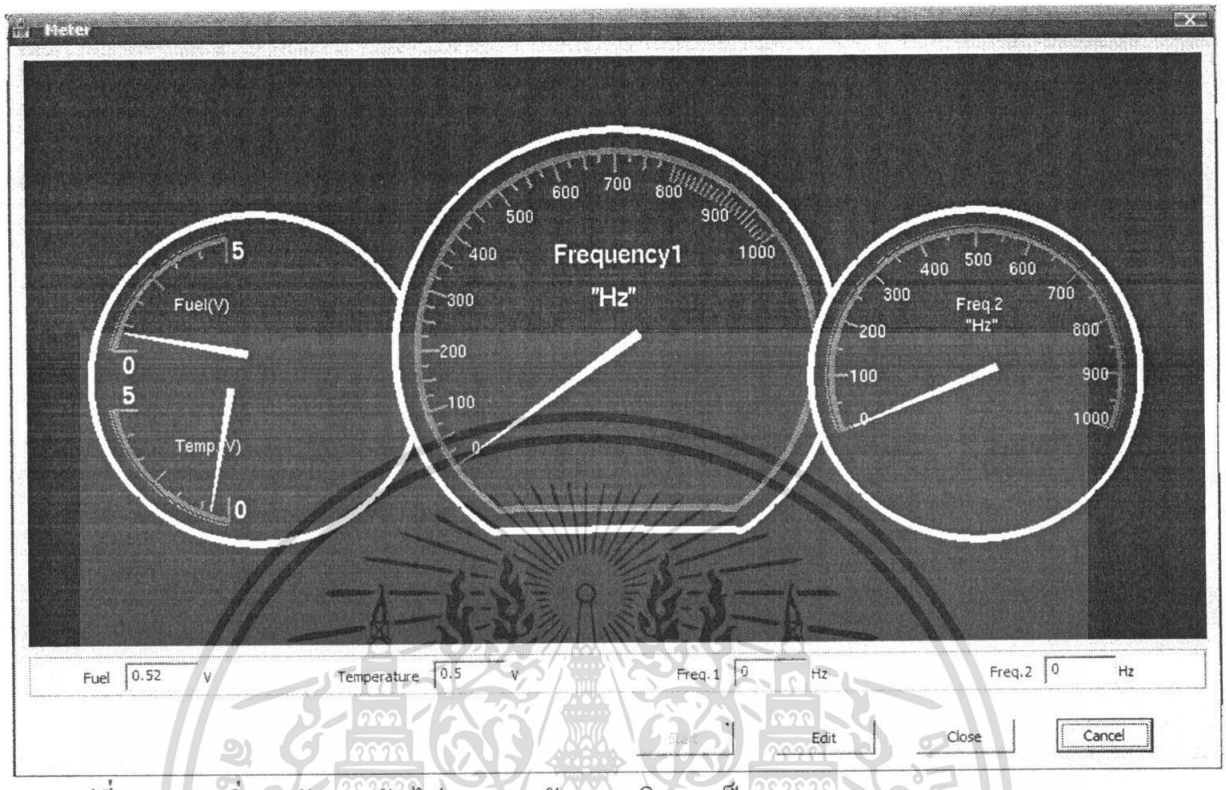
รูปที่ 4.3 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 0.2 V ของ Fuel และ Temp.



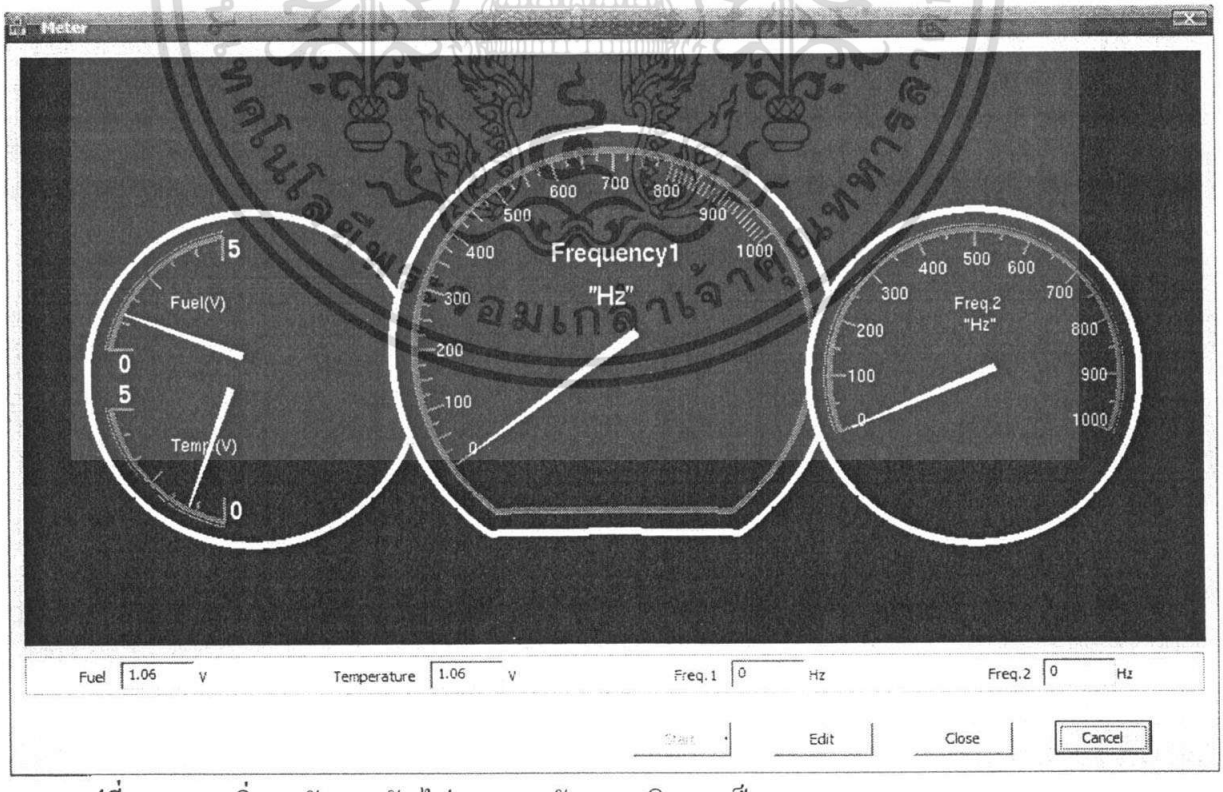
รูปที่ 4.4 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 0.3 V ของ Fuel และ Temp.



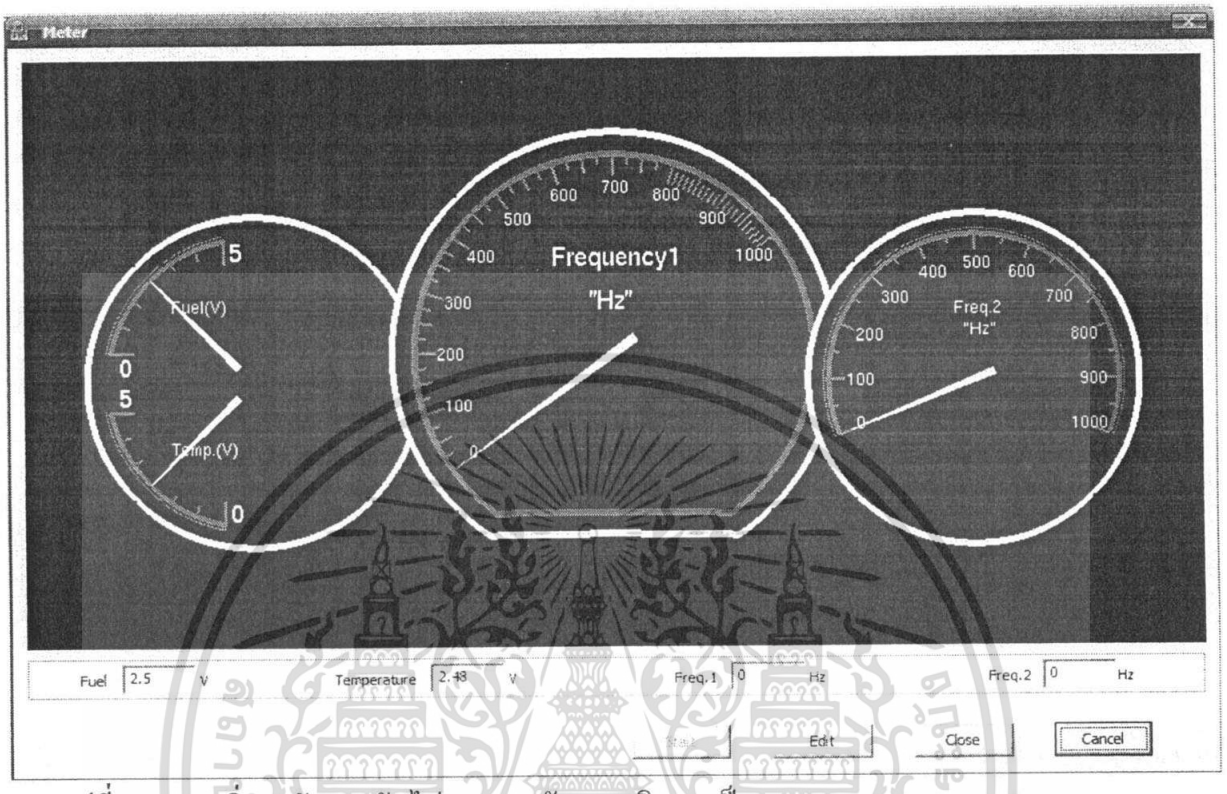
รูปที่ 4.5 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 0.4 V ของ Fuel และ Temp.



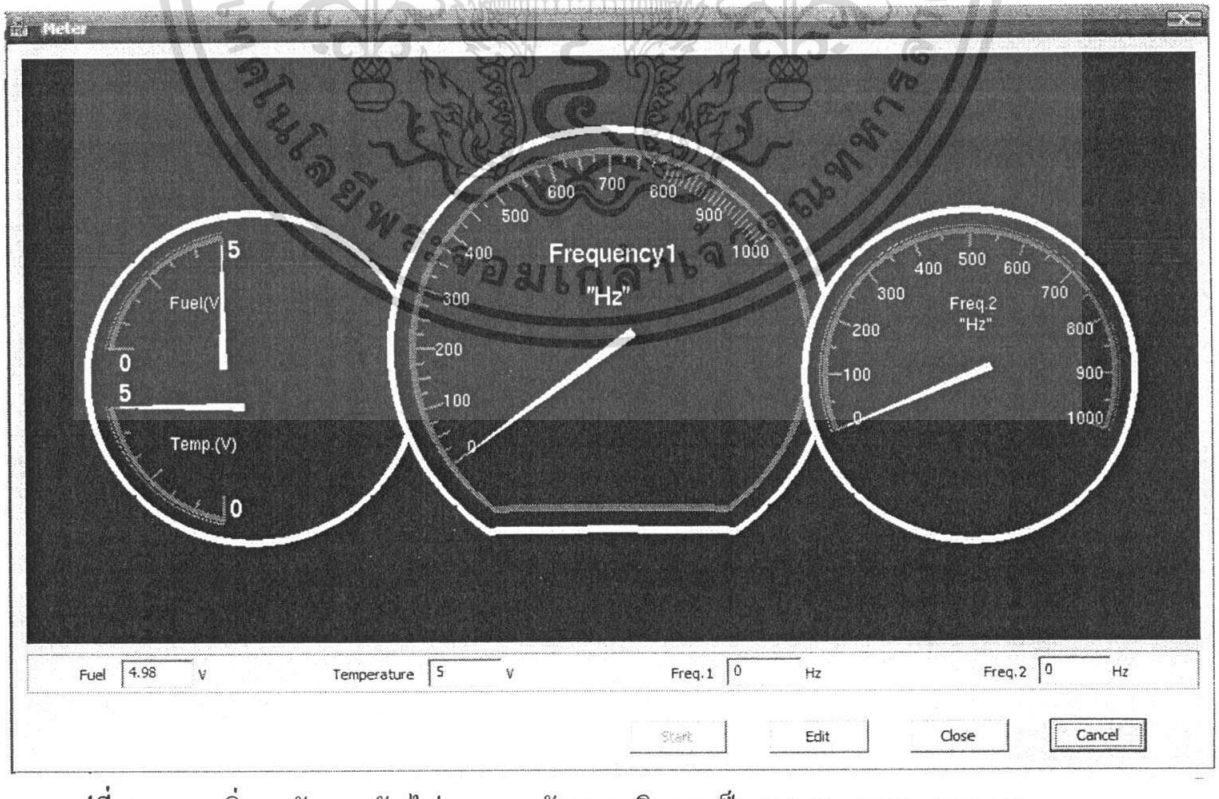
รูปที่ 4.6 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 0.5 V ของ Fuel และ Temp.



รูปที่ 4.7 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 1 V ของ Fuel และ Temp.



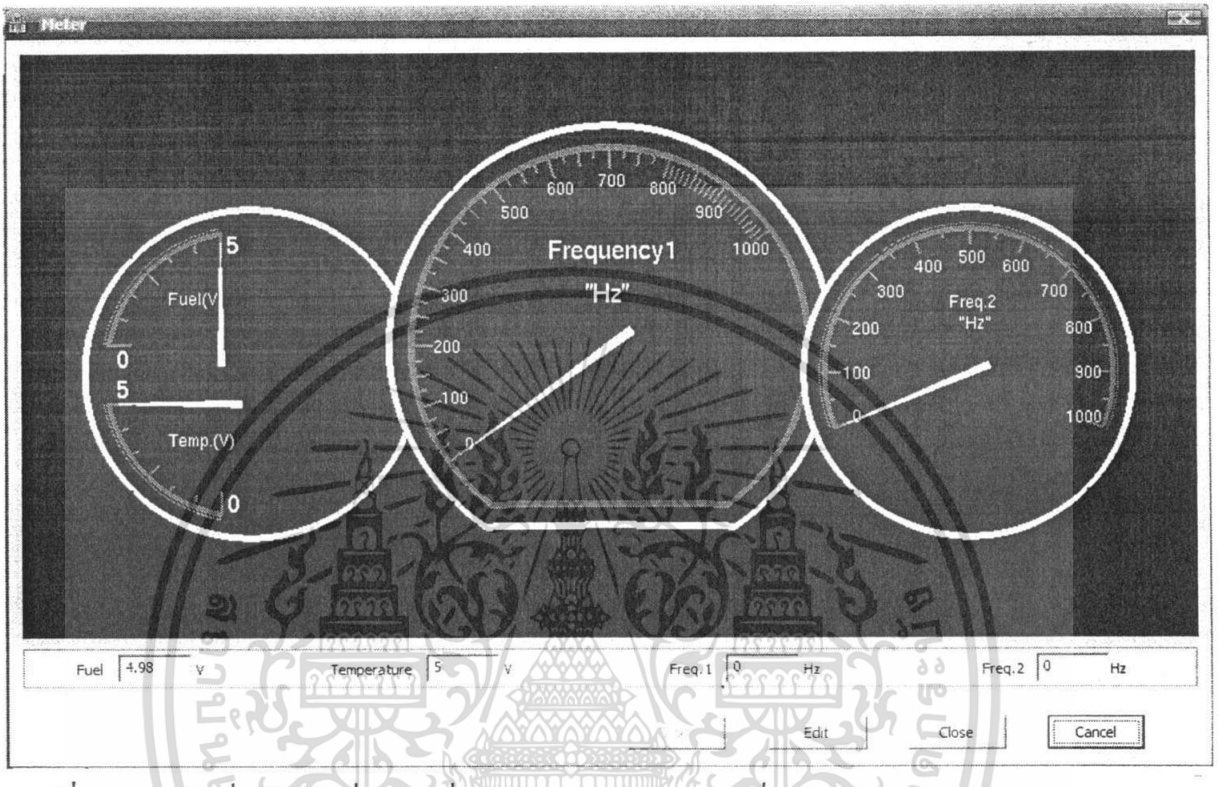
รูปที่ 4.8 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 2.5 V ของ Fuel และ Temp.



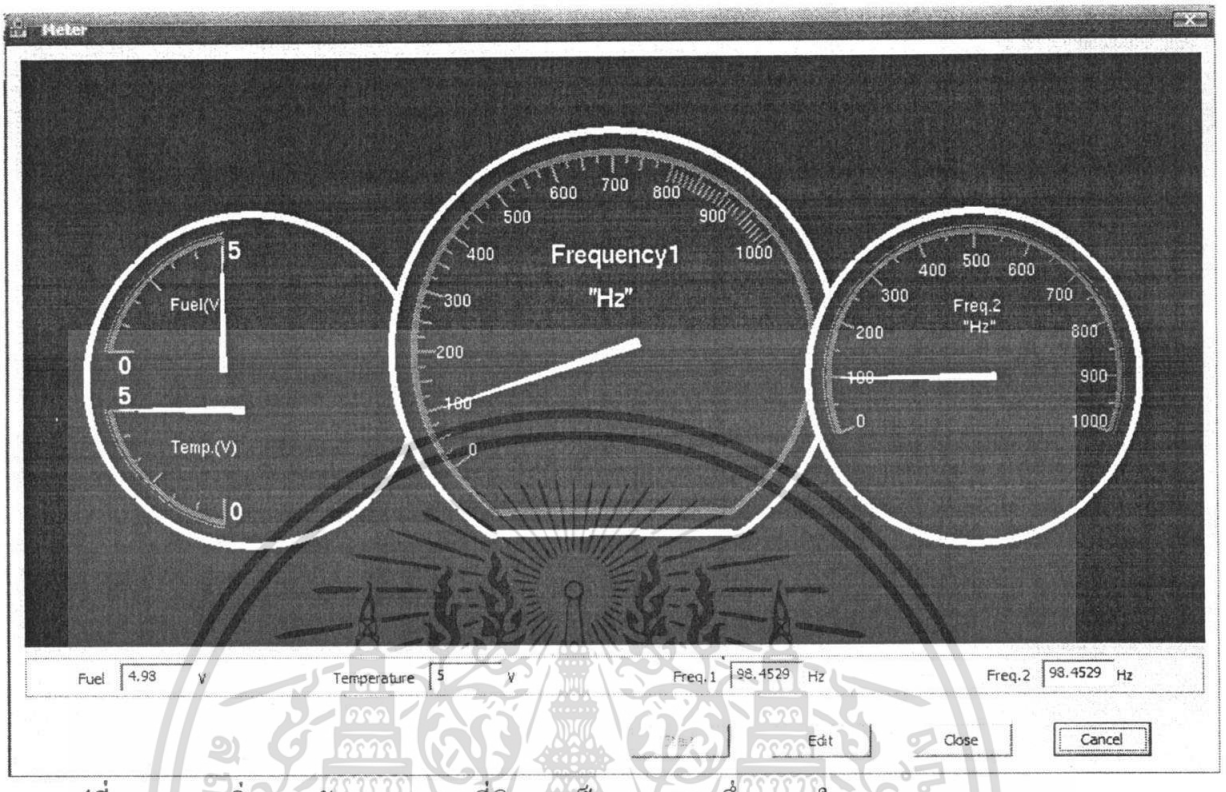
รูปที่ 4.9 การเพิ่มระดับแรงดันไฟตรงของสัญญาณอินพุทเป็น 2.5 V ของ Fuel และ Temp.

จากรูปที่ 4.1 – 4.9 สามารถอธิบายได้ดังนี้ ขณะที่เราทำการเพิ่มระดับของแรงดันไฟตรง เพื่อจำลองสัญญาณอินพุทที่เราสามารถตรวจจับได้จากรถยนต์นั้น จะถูกทำการเก็บข้อมูลด้วยการตรวจจับของไอซี ADC 0804 โดยทำการ sampling ระดับของแรงดัน มาเก็บไว้และส่งข้อมูลให้กับไมโครคอนโทรเลอร์แล้วส่งให้กับภาครับ (คอมพิวเตอร์) เมื่อภาครับส่งข้อมูล 0xFFไป ไมโครคอนโทรเลอร์ก็จะส่งข้อมูลไปให้โดยผ่านระบบการส่งข้อมูลแบบ usb ซึ่งจะมีตัวโมดูลแปลงระบบ serial ไปเป็นระบบ usb แล้วข้อมูลที่เก็บได้นั้นจะเข้าไปประมวลผลกราฟิกแสดงบนหน้าจอคอมพิวเตอร์ดังรูปข้างต้น

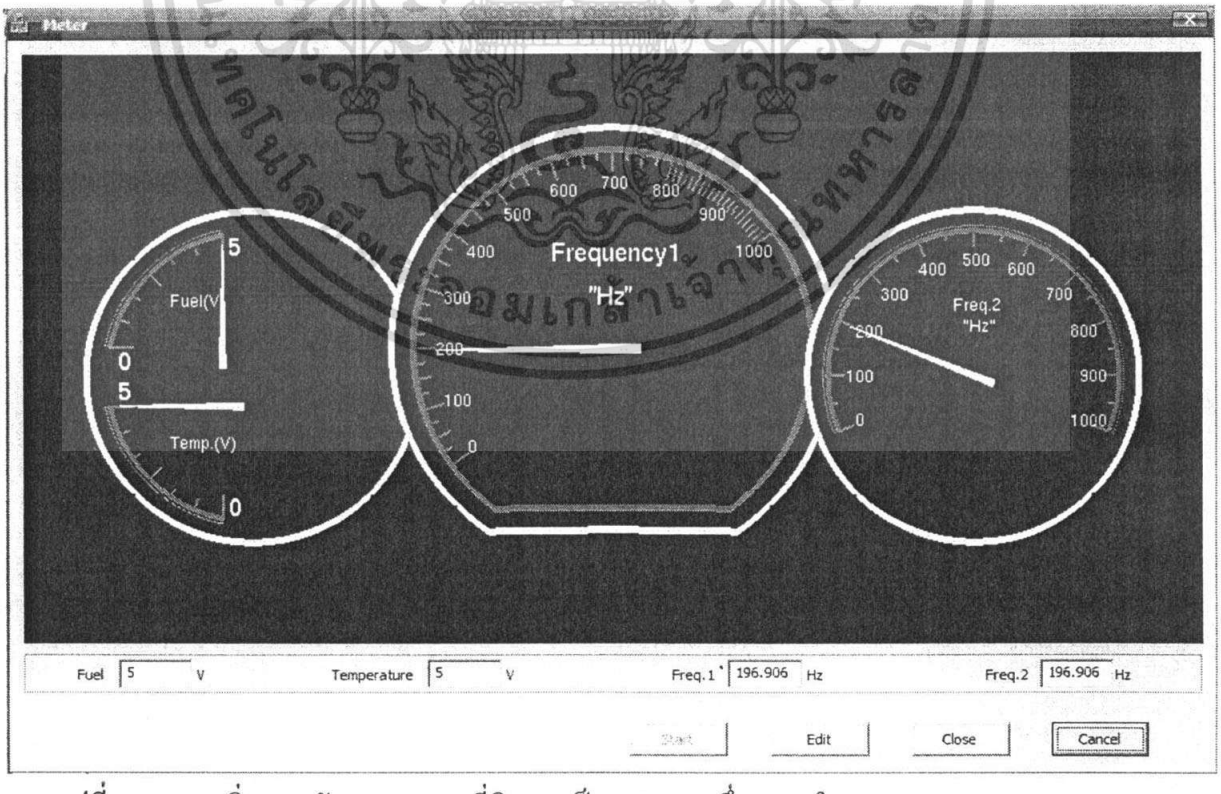
4.2 การทดลองจำลองวัดสัญญาณความถี่จากฟังก์ชั่นเจนเนอร์เรเตอร์แทนข้อมูลของความเร็วรถยนต์และความเร็วรอบเครื่องยนต์



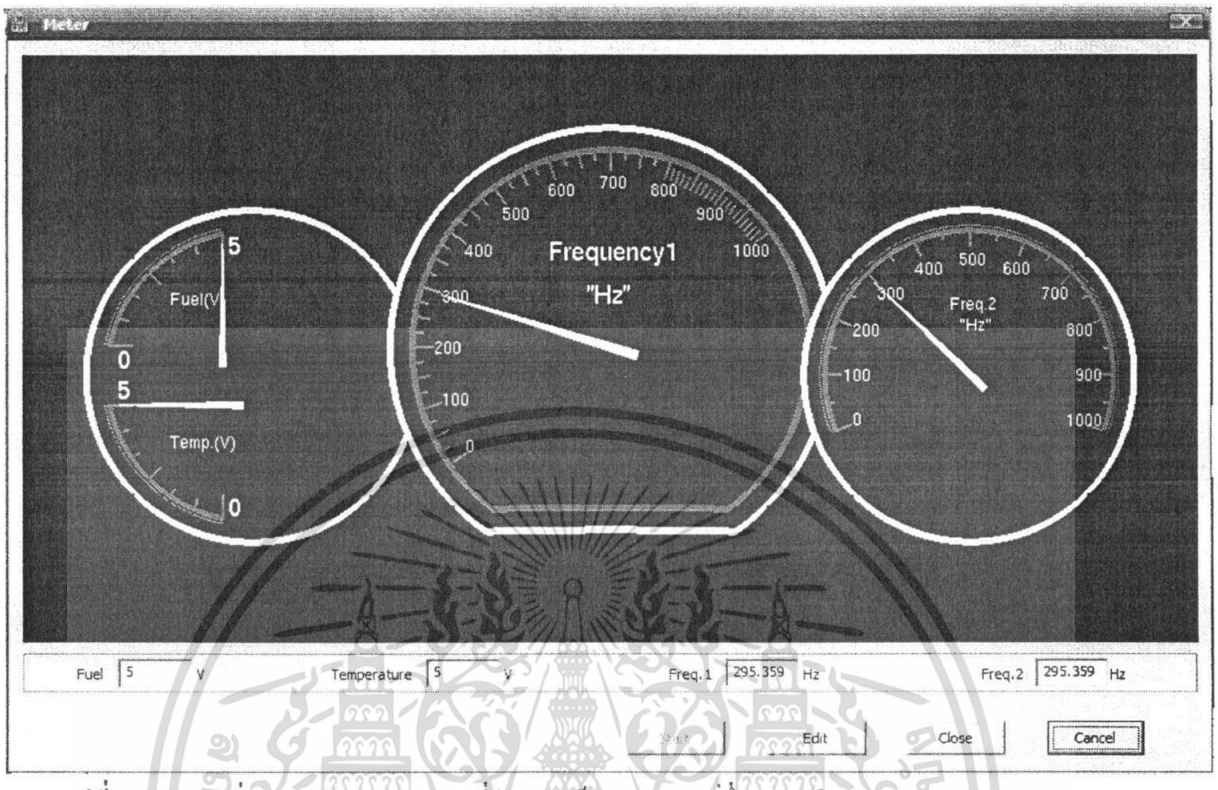
รูปที่ 4.10 ระดับเริ่มต้น คือเมื่อความถี่ของสัญญาณเท่ากับ 0 Hz ซึ่งแสดงใน Frequency 1 และ 2



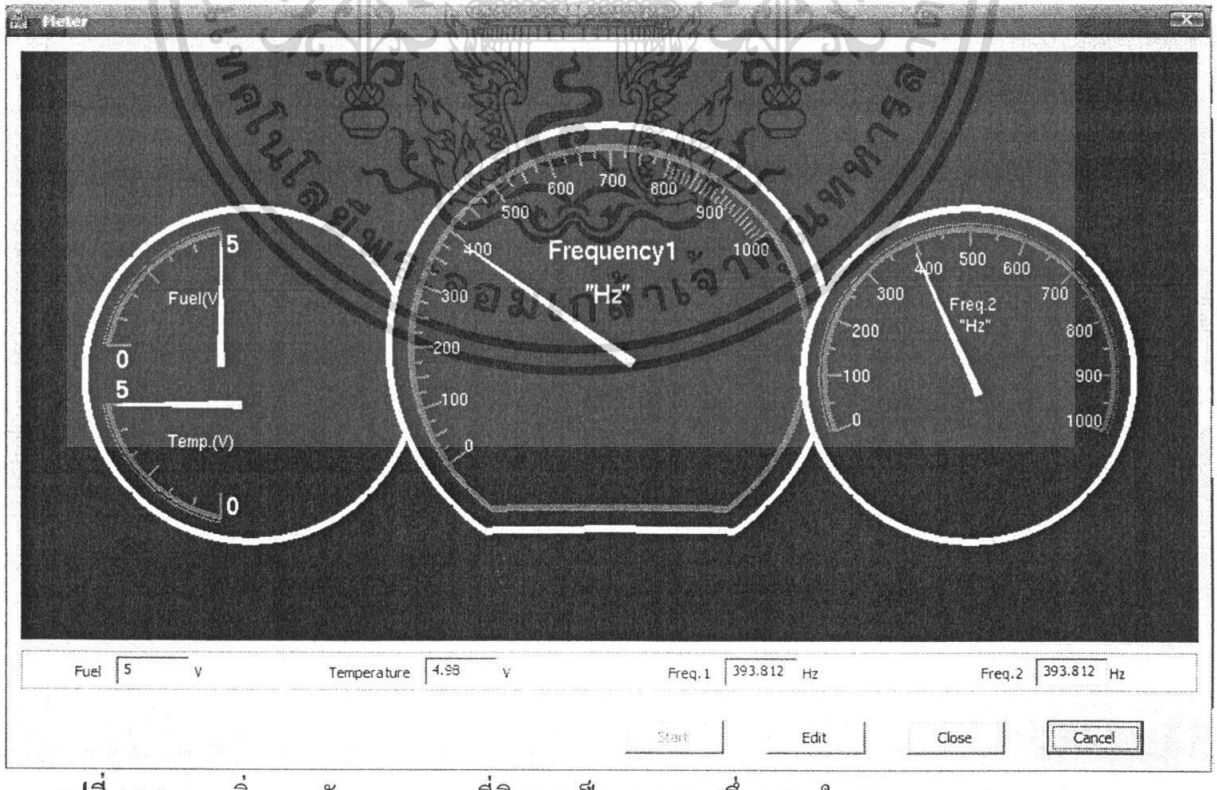
รูปที่ 4.11 การเพิ่มของสัญญาณความถี่อินพุทเป็น 100 Hz ซึ่งแสดงใน Frequency 1 และ 2



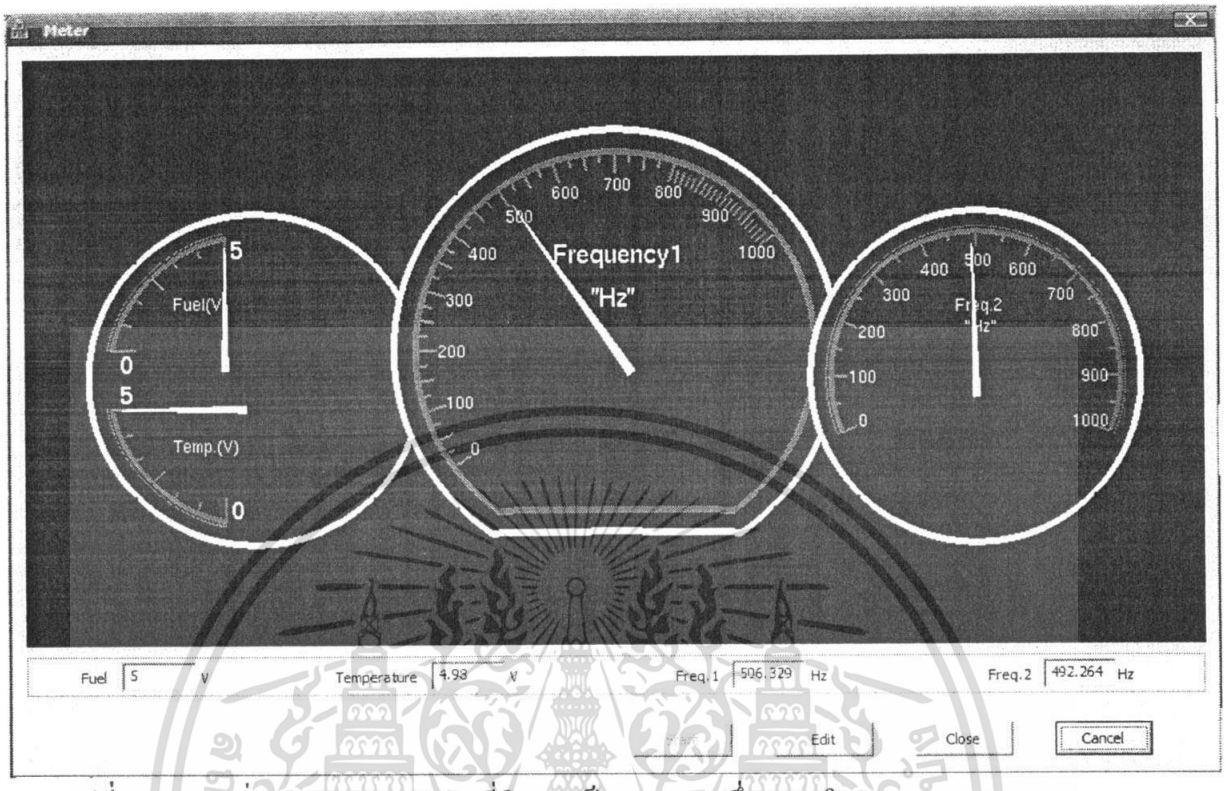
รูปที่ 4.12 การเพิ่มของสัญญาณความถี่อินพุทเป็น 200 Hz ซึ่งแสดงใน Frequency 1 และ 2



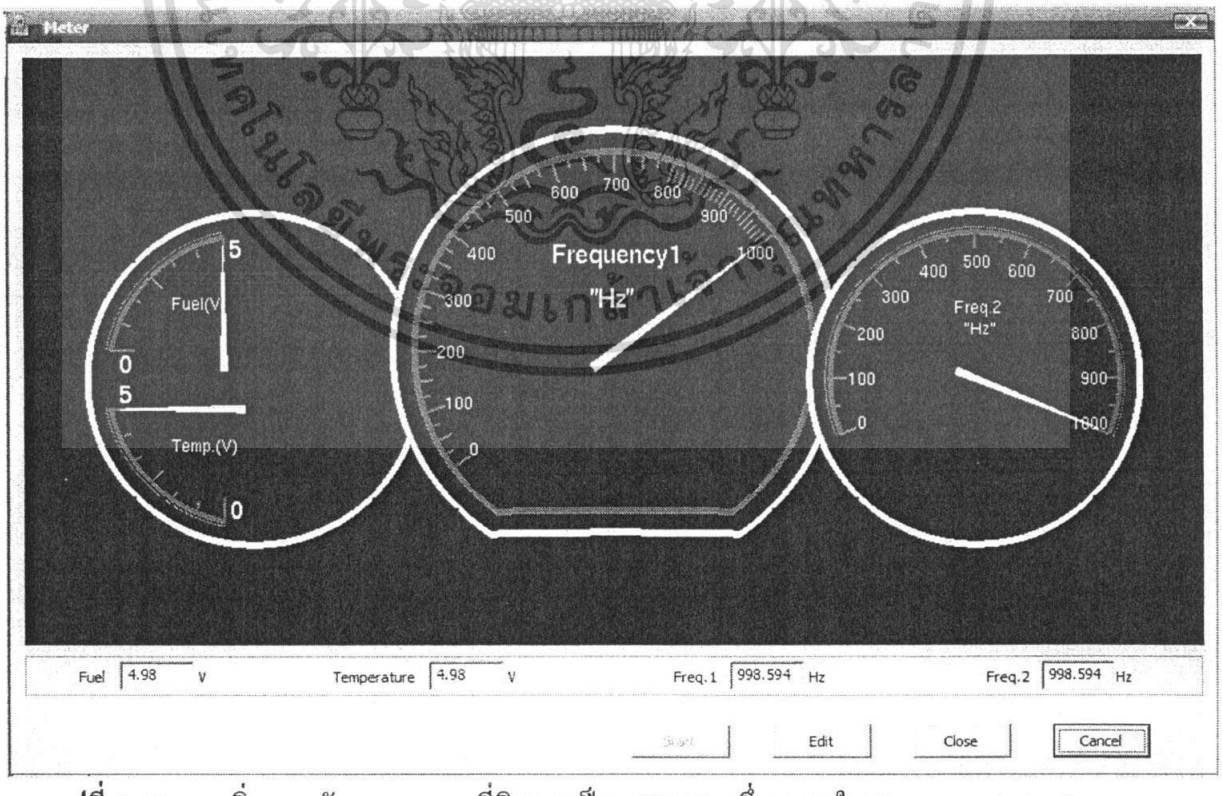
รูปที่ 4.13 การเพิ่มของสัญญาณความถี่อินพุทเป็น 300 Hz ซึ่งแสดงใน Frequency1 และ 2



รูปที่ 4.14 การเพิ่มของสัญญาณความถี่อินพุทเป็น 400 Hz ซึ่งแสดงใน Frequency1 และ 2



รูปที่ 4.15 การเพิ่มของสัญญาณความถี่อินพุทเป็น 500 Hz ซึ่งแสดงใน Frequency1 และ 2



รูปที่ 4.16 การเพิ่มของสัญญาณความถี่อินพุทเป็น 1000 Hz ซึ่งแสดงใน Frequency1 และ2

จากรูปที่ 4.10-4.16 สามารถอธิบายได้ว่า เมื่อเราทำการตรวจจับสัญญานอินพุทจากฟังก์ชั่นเจนเนอเรเตอร์ ซึ่งจะจำลองสัญญานอินพุทที่เราดึงจากรถยนต์ โดยใช้การตรวจจับจับจาก T1 และ T0 เพื่อทำการวัดสัญญาณความถี่ที่เข้ามา แล้วส่งข้อมูลที่ได้ให้กับคอมพิวเตอร์เพื่อให้คอมพิวเตอร์แสดงผลบนกราฟิก โดยทำการส่งผ่านข้อมูลผ่านโมดูลการแปลงจากระบบจาก Serial เป็นระบบ USB

บทที่ 5
บทสรุป

จากการทดลองเราได้ทำการตรวจจับความถี่ (ซึ่งแทนข้อมูลความเร็วรอบเครื่องยนต์และความเร็วรถยนต์) และ ระดับแรงดันไฟตรง (ซึ่งแทนข้อมูลระดับน้ำมันและอุณหภูมิ) ที่มาจากเครื่องฟังก์ชั่นเจนเนอเรเตอร์ และเครื่องพาวเวอร์ซัพพลายตามลำดับ ซึ่งจากการทดลองพบว่าผลที่ได้มีค่าใกล้เคียงจากกับอินพุทที่ป้อนเข้าไป การทำงานโดยรวมเราจะใช้ไมโครคอนโทรเลอร์เป็นตัวแปลงข้อมูลและจัดลำดับข้อมูลแล้วส่งไปยังคอมพิวเตอร์เพื่อประมวลผลแล้วแสดงผลบนกราฟิกที่วาดไว้ โดยจะแปลงสัญญาณความถี่ที่มาจากฟังก์ชั่นเจนเนอเรเตอร์โดยใช้ Timer ของไมโครคอนโทรลเลอร์เป็นนับความถี่จะได้เป็นข้อมูล 16 บิต และแปลงระดับแรงดันไฟตรงที่ได้จากเครื่องพาวเวอร์ซัพพลาย โดยใช้ ADC 0804 เป็นตัวทำการแปลงสัญญาณอะนาล็อกให้เป็นสัญญาณทางดิจิตอล โดย ADC 0804 ได้ถูกควบคุมโดย ไมโครคอนโทรเลอร์ โดยไมโครคอนโทรเลอร์ จะเป็นตัวตัวควบคุมการ sampling ค่ามาเก็บไว้ของ ADC 0804 ซึ่งเราจะสามารถตรวจวัดระดับแรงได้ได้ละเอียดถึง 8 บิต หลังจากที่ข้อมูลทั้ง 4 จัดเก็บเรียบร้อยแล้วก็จะถูกส่งข้อมูลให้กับเครื่องคอมพิวเตอร์โดยผ่านตัวโมดูลที่แปลงระบบการส่งให้เข้าสู่การจัดส่งข้อมูลแบบ USB เพื่อแปลงข้อมูลให้เป็นความถี่และเป็นระดับแรงดันไฟตรงเพื่อใช้คำนวณให้เป็นความเร็วรถยนต์ ความเร็วรอบเครื่องยนต์ ระดับน้ำมัน และ อุณหภูมิ แล้วแสดงผลบนกราฟิก

หนังสืออ้างอิง

1. ขจร อนุดิษย์ “การเขียนโปรแกรมควบคุมไมโครคอนโทรเลอร์ด้วยภาษา C” 344 หน้า, 2550

2. วรเทพ ไพบูรย์รัตนกร “สัมพัสโลก USB EZY ด้วย USB MODULE” 439 หน้า ,2537

3. ยุทธนา ลีลาศวัฒนกุล “คู่มือการเขียนโปรแกรมและใช้งาน Visual C++.Net” 728 หน้า, 2546

ภาคผนวก ก

Code program microcontroller

```
/check baroad rate 19200 recieve f and v check RI//
#include<reg51.h>
sbit Exinterr_v=P3^2;
sbit Exinterr_f=P3^3;
unsigned char th_1,th_0,t0,t1;
sfr T2CON = 0x0c8;
sfr TH2 = 0x0cd;
sfr TL2 = 0x0cc;
sfr RCAP2L=0x0ca;
sbit wr_2=P2^2;
sbit cs_2=P2^6;
sbit rd_2=P2^7;
sbit wr_1=P2^0;
sbit cs_1=P2^4;
sbit rd_1=P2^5;
sfr temp=0x90;
sfr fuel=0x80;
sbit sendtemp=P2^1;
sbit sendfuel=P2^3;
static bit set1=0;
static bit set2=0;
sfr RCAP2H=0x0cb;  //define byte registor//
unsigned char test;
void velocity(void) interrupt 0
{
     if(set1==0)
      {
        TR0=1;
        set1=~set1;
        }
        else
        {
        TR0=0;
        TF0=0;
        th_0=TH0;
        t0  =TL0;
        TH0=0x00;
        TL0=0x00;
        set1=~set1;
        }
}
void frequency(void) interrupt 2
{
     if(set2==0)
       {
        TR1=1;
        set2=~set2;
       }
       else
       {
```

```
        TR1=0;
        TF1=0;
        th_1=TH1;
        t1  =TL1;
        TH1=0x00;
        TL1=0x00;
        set2=~set2;
        }
}
void SerialInterrupt(void)interrupt 4
{
     if(RI == 1)
       {
       test = SBUF;
         RI=0;
       if(test==0x0FF)
          {
               SBUF=0x41;
               while(~TI);
           TI=0;
               SBUF=th_0;
               while(~TI);
               TI=0;
               SBUF=t0;
               while(~TI);
               TI=0;
           SBUF=0x42;
               while(~TI);
           TI=0;
               SBUF=th_1;
               while(~TI);
               TI=0;
               SBUF=t1;
               while(~TI);
               TI=0;
//open sending from adc1 for temp //
           cs_1=0;
           rd_1=0;
           sendtemp=0;
           wr_1=0;
           sendtemp=1;
           wr_1=1;
           SBUF=0x43;
           while(~TI);
           TI=0;
           SBUF=temp;
           while(~TI);
           TI=0;
           cs_1=1;
           rd_1=1;
//---------------------------------//
//open sending from adc2  for fuel//
           cs_2=0;
           rd_2=0;
           sendfuel=0;
           wr_2=0;
           sendfuel=1;
           wr_2=1;
```

```
            SBUF=0x44;
            while(~TI);
            TI=0;
            SBUF=fuel;
            while(~TI);
            TI=0;
            cs_2=1;
            rd_2=1;
            }
       else
             {
            RI=0;
             test=0;
             }
         }
}

void main(void)
  {
  Exinterr_v=1;
  Exinterr_f=1;
  T2CON=0x34; //timer 2 operate//T2CON=00110100
  TH2=0xff;
  TL2=0xdd;
  RCAP2H=0xff;
  RCAP2L=0xdd; //19.2k,RCAP2l=0xfa;//115.2k,RCAP2L=0xff;//691186
  SCON=0X52;                    //set scon for sending in sereil port//
  EA=1;
  ES=1;
  EX1=1;
  EX0=1;
  IP=0x15;
  RI=0;
  TI=0;
  TMOD=0x11;
  TH0=0x00;
  TL0=0x00;
  TH1=0x00;
  TL1=0x00;
  IT1=1;
  IT0=1;
  while(1);
  }
```

ภาคผนวก ข

Code program computer

Header

EditData.h

```
#pragma once
class CEditData : public CDialog
{
	DECLARE_DYNAMIC(CEditData)
public:
	CEditData(CWnd* pParent = NULL);
	virtual ~CEditData();
	enum { IDD = IDD_DIALOG_EDIT };
protected:
	virtual void DoDataExchange(CDataExchange* pDX);
	DECLARE_MESSAGE_MAP()
public:
	GLfloat m_width;
	GLfloat m_hight;
	GLfloat m_diameter;
	GLfloat m_sensor;
};
```

OpenGLControl.h

```
#pragma once
#include "afxwin.h"
#include "OpenGLDevice.h"
```

```
class COpenGLControl : public CWnd
{
	DECLARE_DYNAMIC(COpenGLControl)
public:
	COpenGLControl();
	virtual ~COpenGLControl();
	void Create(CRect rect, CWnd* parent);
	GLfloat sc;
	GLfloat sl1;
	GLfloat sl2;
	GLfloat sr;
protected:
	DECLARE_MESSAGE_MAP()
	afx_msg void OnPaint();
	afx_msg BOOL OnEraseBkgnd(CDC* pDC);
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
protected:
	void InitGL(void);
	void DrawGLScene(void);
	COpenGLDevice openGLDevice;
	CClientDC * dc;
	GLfloat x;
	GLfloat y;
	GLfloat rc;
	GLfloat rr;
	GLfloat rl;
	GLfloat pi;
};
```

OpenGLDevice.h

```
#pragma once
#include <windows.h>
#include "afxwin.h"
class COpenGLDevice
{
public:
        COpenGLDevice(void);
        COpenGLDevice(HDC hDC);
        void destroy(void);
        void makeCurrent(bool  disableOther= true);
        bool CreateRenderingContext(HDC hDC);
public:
        virtual ~COpenGLDevice(void);
protected:
        HGLRC m_hRC;
        HDC hDC;
        bool SetPixelFormatDes(HDC hDC);
};
```

Project4CDlg.h

```
#pragma once
#include "afxwin.h"
#include "openglcontrol.h"
#include "atltypes.h"
class CProject4CDlg : public CDialog
{
public:
        CProject4CDlg(CWnd* pParent = NULL);
        enum { IDD = IDD_PROJECT4C_DIALOG };
```

```
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
protected:
        HICON m_hIcon;
        virtual BOOL OnInitDialog();
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnBnClickedButtonEdit();
        COpenGLControl openGLControl;
        afx_msg void OnTimer(UINT_PTR nIDEvent);
        afx_msg void OnBnClickedButtonStart();
        afx_msg void OnBnClickedButtonClose();
        FT_HANDLE ftHandle;
        GLfloat m_wheel_width;
        GLfloat m_wheel_hight;
        GLfloat m_wheel_diameter;
        GLfloat m_wheel_sensor;
        GLfloat Time_v;
        GLfloat Time_r;
        GLfloat m_Fuel;
        GLfloat m_Temp;
        GLfloat m_Engine;
        GLfloat m_Velocity;
        CRect rect_update;
};
```

Source code

EditData.cpp

```
#include "stdafx.h"
#include "Project4C.h"
#include "EditData.h"
IMPLEMENT_DYNAMIC(CEditData, CDialog)
CEditData::CEditData(CWnd* pParent /*=NULL*/)
	: CDialog(CEditData::IDD, pParent)
	, m_width(0)
	, m_hight(0)
	, m_diameter(0)
	, m_sensor(0)
{
}
CEditData::~CEditData()
{
}
void CEditData::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDIT_WIDTH, m_width);
	DDX_Text(pDX, IDC_EDIT_HIGHT, m_hight);
	DDX_Text(pDX, IDC_EDIT_DIAMETER, m_diameter);
	DDX_Text(pDX, IDC_EDIT_SENSOR, m_sensor);
}
BEGIN_MESSAGE_MAP(CEditData, CDialog)
END_MESSAGE_MAP()
```

OpenGLControl.cpp

```
#include "stdafx.h"
```

```
#include "Project4C.h"
#include "OpenGLControl.h"
#include "math.h"
IMPLEMENT_DYNAMIC(COpenGLControl, CWnd)
COpenGLControl::COpenGLControl()
{
	dc = NULL;
	rc = 3.0;
	dc = NULL;
	rr = (rc*3)/4;
	rl = (rc*3)/4;
	y  = 0;
	x  = 0;
	pi = 3.14 ;
	sc = 24;
	sr = 18;
	sl1 = 48;
	sl2 = 0 ;
}
COpenGLControl::~COpenGLControl()
{
	if (dc)
	{
		delete dc;
	}
}
BEGIN_MESSAGE_MAP(COpenGLControl, CWnd)
	ON_WM_PAINT()
	ON_WM_ERASEBKGND()
	ON_WM_SIZE()
```

```
        ON_WM_CREATE()
END_MESSAGE_MAP()
void COpenGLControl::InitGL(void)
{
        glShadeModel(GL_SMOOTH);
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glClearDepth(1.0f);
        glDepthFunc(GL_LEQUAL);
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
}
void COpenGLControl::DrawGLScene(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();



                        // DRAWING CODE
              /*-----------------------วงกลม-----------------------------*/
        GLfloat a;
        a = (19*rc)/8;
        glLineWidth(5.0);
        glColor3f(1.0f,1.0f,1.0f);
        glBegin(GL_LINE_STRIP);
        for(GLfloat x=0.969*rc;x>=-rc;x-=0.01)
        {
                y = sqrt(rc*rc-x*x);
                glVertex2f(x,y);
        }
        for(GLfloat x=-rc;x<=-0.56*rc;x+=0.01)
```

```
{
        y = sqrt(rc*rc-x*x);
        glVertex2f(x,-y);
}
a = (2*19*rc)/8;
for(GLfloat x=-0.56*rc;x<=0.56*rc;x+=0.01)
{
        y = rc*sqrt((1+x/a)*(1-x/a));
        glVertex2f(x,y-1.82*rc);
}
for(GLfloat x=0.56*rc;x<=0.920*rc;x+=0.01)
{
        y = sqrt(rc*rc-x*x);
        glVertex2f(x,-y);
}
glEnd();
GLfloat rcs;
glColor3f(1.0f,0.0f,0.0f);
glBegin(GL_LINE_STRIP);
rcs = (rc*9)/10 ;
for(GLfloat x=0.996*rcs;x>=-rcs;x-=0.01)
{
        y = sqrt(rcs*rcs-x*x);
        glVertex2f(x,y);
}
for(GLfloat x=-rcs;x<=-0.535*rc;x+=0.01)
{
        y = sqrt(rcs*rcs-x*x);
        glVertex2f(x,-y);
}
```

```
a = (2*19*rc)/8;
for(GLfloat x=-0.535*rc;x<=0.535*rc;x+=0.01)
{
        y = rc*sqrt((1+x/a)*(1-x/a));
        glVertex2f(x,y-1.72*rc);
}
for(GLfloat x=0.535*rc;x<=0.975*rcs;x+=0.01)
{
        y = sqrt(rcs*rcs-x*x);
        glVertex2f(x,-y);
}
glEnd();
        /*-------------------------------------------------------------------*/
glColor3f(1.0f,1.0f,1.0f);
glBegin(GL_LINE_LOOP);


for(GLfloat x=(7*rc)/8;x<=(19*rc)/8;x+=0.01)
{
        y = sqrt(rr*rr-(x-(13*rc)/8)*(x-(13*rc)/8));
        glVertex2f(x,y-rc/8);
}
for(GLfloat x=(19*rc)/8;x>=(7*rc)/8;x-=0.01)
{
        y = sqrt(rr*rr-(x-(13*rc)/8)*(x-(13*rc)/8));
        glVertex2f(x,-y-rc/8);
}
glEnd();
        /*-------------------------------------------------------------------*/
glBegin(GL_LINE_STRIP);
```

```
        for(GLfloat x=-(0.969*rc);x>=-(19*rc)/8;x-=0.01)
        {
                y = sqrt(rl*rl-(x+(13*rc)/8)*(x+(13*rc)/8));
                glVertex2f(x,y-rc/8);
        }
        for(GLfloat x=-(19*rc)/8;x<=-(0.920*rc);x+=0.01)
        {
                y = sqrt(rl*rl-(x+(13*rc)/8)*(x+(13*rc)/8));
                glVertex2f(x,-y-rc/8);
        }
        glEnd();


                /*-----------------------------สเกลมิเตอร์-----------------------------*/
glColor3f(1.0f,0.0f,0.0f);
glLineWidth(2.0);
for(GLfloat i=4;i<=12;i+=1)
        {
                GLfloat a = cos(pi*i/10);
                GLfloat b = sin(pi*i/10);
                glBegin(GL_LINES);
                glVertex2f(0.9*rc*a,0.9*rc*b);
                glVertex2f(0.8*rc*a,0.8*rc*b);
                glEnd();
        }
for(GLfloat i=4.5;i<=12;i+=1)
```

```
        {
                GLfloat a = cos(pi*i/10);
                GLfloat b = sin(pi*i/10);
                glBegin(GL_LINES);
                glVertex2f(0.9*rc*a,0.9*rc*b);
                glVertex2f(0.825*rc*a,0.825*rc*b);
                glEnd();
        }
for(GLfloat i=4.25;i<=12;i+=0.5)
        {
                GLfloat a = cos(pi*i/10);
                GLfloat b = sin(pi*i/10);
                glBegin(GL_LINES);
                glVertex2f(0.9*rc*a,0.9*rc*b);
                glVertex2f(0.85*rc*a,0.85*rc*b);
                glEnd();
        }
for(GLfloat i=2;i<=4;i+=0.5)
        {
                GLfloat a = cos(pi*i/10);
                GLfloat b = sin(pi*i/10);
                glBegin(GL_LINES);
                glVertex2f(0.9*rc*a,0.9*rc*b);
                glVertex2f(0.8*rc*a,0.8*rc*b);
                glEnd();
        }
for(GLfloat i=2;i<=4;i+=0.1)
        {
                GLfloat a = cos(pi*i/10);
                GLfloat b = sin(pi*i/10);
```

```
                glBegin(GL_LINES);
                glVertex2f(0.86*rc*a,0.86*rc*b);
                glVertex2f(0.8*rc*a,0.8*rc*b);
                glEnd();
        }
        /*-------------------------------------------------------------------------------------------*/
GLfloat rrs;
glLineWidth(1.0);
for(rrs=0.905*rr;rrs>=0.875*rr;rrs-=0.03*rr)
        {
                glBegin(GL_LINE_STRIP);
                for(GLfloat x=(13*rc)/8+rrs*cos((9*pi)/8);x>=(13*rc)/8-rrs;x-=0.01)
                {
                        y = sqrt(rrs*rrs-(x-(13*rc)/8)*(x-(13*rc)/8));
                        glVertex2f(x,-y-rc/8);
                }
                for(GLfloat x=(13*rc)/8-rrs;x<=(13*rc)/8+rrs;x+=0.01)
                {
                        y = sqrt(rrs*rrs-(x-(13*rc)/8)*(x-(13*rc)/8));
                        glVertex2f(x,y-rc/8);
                }
                for(GLfloat x=(13*rc)/8+rrs;x>=(13*rc)/8+rrs*cos((-1.03*pi)/8);x-=0.01)
                {
                        y = sqrt(rrs*rrs-(x-(13*rc)/8)*(x-(13*rc)/8));
                        glVertex2f(x,-y-rc/8);
                }
                glEnd();
                glLineWidth(4.0);
        }
```

```
glLineWidth(2.0);
for(GLfloat i=-1;i<=9;i+=1)
                {
                        GLfloat a = cos(pi*i/8);
                        GLfloat b = sin(pi*i/8);
                        glBegin(GL_LINES);
                        glVertex2f((13*rc)/8+0.875*rr*a,-rc/8+0.875*rr*b);
                        glVertex2f((13*rc)/8+0.775*rr*a,-rc/8+0.775*rr*b);
                        glEnd();
                }
for(GLfloat i=-0.5;i<=9;i+=1)
                {
                        GLfloat a = cos(pi*i/8);
                        GLfloat b = sin(pi*i/8);
                        glBegin(GL_LINES);
                        glVertex2f((13*rc)/8+0.875*rr*a,-rc/8+0.875*rr*b);
                        glVertex2f((13*rc)/8+0.825*rr*a,-rc/8+0.825*rr*b);
                        glEnd();
                }
                /*-------------------------------------------------------------------*/
GLfloat rls1,rls2 ;
glLineWidth(1.0);
for(rls1 = 0.905*rl ; rls1>=0.875*rl; rls1-=0.03*rl)
        {
                glBegin(GL_LINE_STRIP);
                for(GLfloat x=-1.758*rc;x>=-(2.227*rc);x-=0.01)
        {
                y = sqrt(rls1*rls1-(x+(13*rc)/8)*(x+(13*rc)/8));
                glVertex2f(x,y-rc/8);
        }
```

```
glEnd();
glBegin(GL_LINE_STRIP);
for(GLfloat y=0.0076*rc;y<=(0.477*rc);y+=0.01)
        {
                x = sqrt(rls1*rls1-(y+rc/8)*(y+rc/8));
                glVertex2f(-x-(13*rc)/8,y);
        }
                glEnd();
                glLineWidth(4.0);
        }

glLineWidth(2.0);
for(GLfloat i=2;i<=4;i+=2)
        {
                GLfloat a = cos(pi*i/4);
                GLfloat b = sin(pi*i/4);
                glBegin(GL_LINES);
                glVertex2f(-1.758*rc+0.685*rl*a,0.0076*rc+0.685*rl*b);
                glVertex2f(-1.758*rc+0.535*rl*a,0.0076*rc+0.535*rl*b);
                glEnd();
        }
glBegin(GL_LINES);
glVertex2f(-1.758*rc+0.625*rl*cos(pi*3/4),0.0076*rc+0.625*rl*sin(pi*3/4));
glVertex2f(-1.758*rc+0.525*rl*cos(pi*3/4),0.0076*rc+0.525*rl*sin(pi*3/4));
glEnd();

for(GLfloat i=6;i<=12;i+=1)
        {
                GLfloat a = cos(pi*i/12);
                GLfloat b = sin(pi*i/12);
```

```
                glBegin(GL_LINES);
                glVertex2f(-1.758*rc+0.625*rr*a,0.0076*rc+0.625*rr*b);
                glVertex2f(-1.758*rc+0.575*rr*a,0.0076*rc+0.575*rr*b);
                glEnd();
        }
                /*-----------------------------------------------------------------------*/
glLineWidth(1.0);
for(rls2 = 0.905*rl ; rls2>=0.875*rl; rls2-=0.03*rl)
        {
                glBegin(GL_LINE_STRIP);
                for(GLfloat x=-1.758*rc;x>=-(2.227*rc);x-=0.01)
                {
                        y = sqrt(rls2*rls2-(x+(13*rc)/8)*(x+(13*rc)/8));
                        glVertex2f(x,-y-rc/8);
                }
                glEnd();
                glBegin(GL_LINE_STRIP);
                for(GLfloat y=-0.258*rc;y>=-0.727*rc;y-=0.01)
                {
                        x = sqrt(rls2*rls2-(y+rc/8)*(y+rc/8));
                        glVertex2f(-x-(13*rc)/8,y);
                }
                glEnd();
                glLineWidth(4.0);
        }

glLineWidth(2.0);//เส้นขีด
for(GLfloat i=4;i<=6;i+=2) //ขีดยาว
        {
                GLfloat a = cos(pi*i/4);
```

```
                GLfloat b = sin(pi*i/4);
                glBegin(GL_LINES);
                glVertex2f(-1.758*rc+0.685*rl*a,-0.258*rc+0.685*rl*b);
                glVertex2f(-1.758*rc+0.535*rl*a,-0.258*rc+0.535*rl*b);
                glEnd();
        }
glBegin(GL_LINES);
glVertex2f(-1.758*rc+0.625*rl*cos(pi*5/4),-0.258*rc+0.625*rl*sin(pi*5/4));
glVertex2f(-1.758*rc+0.525*rl*cos(pi*5/4),-0.258*rc+0.525*rl*sin(pi*5/4));
glEnd();

for(GLfloat i=12;i<=18;i+=1)//ขีดสั้น
        {
                GLfloat a = cos(pi*i/12);
                GLfloat b = sin(pi*i/12);
                glBegin(GL_LINES);
                glVertex2f(-1.758*rc+0.625*rr*a,-0.258*rc+0.625*rr*b);
                glVertex2f(-1.758*rc+0.575*rr*a,-0.258*rc+0.575*rr*b);
                glEnd();
        }
            /*-----------------------------ตัวเลขบนมิเตอร์-----------------------------*/
glColor3f(1.0f,1.0f,1.0f);
char i[11] ={'0','1','2','3','4','5','6','7','8','9','1'};
for(int j=12;j>=2;j--)
        {
                glRasterPos2f (0.75*rc*cos((pi*j)/10)-0.2*cos(pi/4),0.75*rc*sin((pi*j)/10)-
0.1*sin(pi/4));
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, i[12-j]);
                if(j<12)
                {
```

```
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
                    if(j==2)
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
                }
        }
char cha[] = {'F','r','e','q','u','e','n','c','y','1'};
char cha_Hz[] = {"","H","z",""};
glRasterPos2f(-0.275*rc,0.4*rc);
for(int j=0;j<=9;j++)
        {
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,cha[j]);
        }
glRasterPos2f(-0.1*rc,0.2*rc);
for(int j=0;j<4;j++)
        {
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,cha_Hz[j]);
        }

                /*-------------------------------------------------------------------*/

int buff ;
for(int j=9;j>=-1;j--)
{
        glRasterPos2f ((13*rc)/8+0.7*rr*cos((pi*j)/8)-0.2*cos(pi/4),-rc/8+0+0.7*rr*sin((pi*j)/8)-
        0.1*sin(pi/4));
        buff = j+3;
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, i[12-buff]);
        if(buff<12)
        {
```

```
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
                if(buff==2)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,i[0]);
        }
}

char cha_v[] = {'F','r','e','q','.','2'};
glRasterPos2f((13*rc)/8-0.125*rr,-rc/8+0.4*rr);
for(int j=0;j<=5;j++)
{
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,cha_v[j]);
}
glRasterPos2f((13*rc)/8-0.05*rc,-rc/8+0.2*rc);
for(int j=0;j<4;j++)
{
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,cha_Hz[j]);
}
                /*-----------------------------------------------------------------*/

glRasterPos2f (-1.758*rc+0.625*rl*cos((4*pi)/4),-0.0876*rc);
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'0');
glRasterPos2f (-1.728*rc,-0.0376*rc+0.625*rl*sin((2*pi)/4));
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'1');
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'2');

char cha2[] = {'F','u','e','l','(','V',')'};  // Fuel
glRasterPos2f(-1.758*rc-0.313*rl,0.0076*rc+0.25*rl);
for(int j=0;j<=6;j++)
{
```

```
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,cha2[j]);
}
        /*-------------------------------------------------------------------*/
glRasterPos2f (-1.758*rc+0.625*rl*cos((4*pi)/4),-0.228*rc);
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'1');
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'2');
glRasterPos2f (-1.728*rc,-0.285*rc+0.625*rl*sin((6*pi)/4));
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'0');

char cha3[] = {'T','e','m','p','.','(','V',')'};
glRasterPos2f(-1.758*rc-0.313*rl,-0.258*rc-0.25*rl);
for(int j=0;j<=7;j++)
{
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,cha3[j]);
}

        /*-----------------------------เข็มมิเตอร์-----------------------------*/

glColor3f(1.0,1.0,1.0);
GLfloat rpc,y1,y2,x1,x2,npc,n;
n  = 20;
npc = (3.14)/n;
if(sc<(2*n)/10)
        sc=(2*n)/10;
if(sc>(12*n)/10)
        sc=(12*n)/10;
rpc = (rc*1)/8;
```

```
x = 0.8*rc*cos(sc*npc);
y = 0.8*rc*sin(sc*npc);

x1 = -rpc*cos((sc-0.05*n)*npc);
x2 = -rpc*cos((sc+0.05*n)*npc);

y1 = -rpc*sin((sc-0.05*n)*npc);
y2 = -rpc*sin((sc+0.05*n)*npc);

glLineWidth(1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();

                /*------------------------------------------------------------------------*/

GLfloat rpr,npr;
n  =  16;
npr = (3.14)/n;
if(sr<(-1*n)/8)
        sr=(-1*n)/8;
if(sr>(9*n)/8)
        sr=(9*n)/8;
```

```
rpr = (rr*1)/8;

x =  rc*1.625+0.8*rr*cos(sr*npr);
y = -rc*0.125+0.8*rr*sin(sr*npr);

x1 = rc*1.625-rpr*cos((sr-0.05*n)*npr);
x2 = rc*1.625-rpr*cos((sr+0.05*n)*npr);

y1 =  -rc*0.125-rpr*sin((sr-0.05*n)*npr);
y2 =  -rc*0.125-rpr*sin((sr+0.05*n)*npr);

glLineWidth(1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
                /*------------------------------------------------------------------------*/
GLfloat rpl1,npl1,rl1;
n   = 48;
npl1 = (3.14)/n;
if(sl1>(24*n)/24)
        sl1=(24*n)/24;
if(sl1<(12*n)/24)
        sl1=(12*n)/24;
```

```
rl1  = (3*rl)/4;
rpl1 = (rl*1)/8;

x = -rc*1.758+(rl*0.625)*cos(sl1*npl1);
y =  rc*0.0076+(rl*0.625)*sin(sl1*npl1);

x1 =-rc*1.758-rpl1*cos((sl1-0.05*n)*npl1);
x2 =-rc*1.758-rpl1*cos((sl1+0.05*n)*npl1);

y1 = rc*0.0076-rpl1*sin((sl1-0.05*n)*npl1);
y2 = rc*0.0076-rpl1*sin((sl1+0.05*n)*npl1);

glLineWidth(1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();

                /*-------------------------------------------------------------------------*/

GLfloat rpl2,npl2,rl2;
n   = 48;
npl2 = (3.14)/n;
if(sl2>(36*n)/24)
```

```
        sl2=(36*n)/24;
if(sl2<(24*n)/24)   //max = 48, min = 72
        sl2=(24*n)/24;
rl2  = (3*rl)/4;
rpl2 = (rl*1)/8;

x = -rc*1.758+(rl*0.625)*cos(sl2*npl2);    // จุดศ.ก.อยู่ที่ (-1.758*rc,-0.258*rc)
y = -rc*0.258+(rl*0.625)*sin(sl2*npl2);

x1 = -rc*1.758-rpl1*cos((sl2-0.05*n)*npl2);
x2 = -rc*1.758-rpl1*cos((sl2+0.05*n)*npl2);

y1 = -rc*0.258-rpl1*sin((sl2-0.05*n)*npl2);
y2 = -rc*0.258-rpl1*sin((sl2+0.05*n)*npl2);

glLineWidth(1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(x,y);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
SwapBuffers(dc->m_hDC);
}
```

```
void COpenGLControl::OnPaint()
{
	CPaintDC dc(this);
	openGLDevice.makeCurrent();
	DrawGLScene();
}
BOOL COpenGLControl::OnEraseBkgnd(CDC* pDC)
{
	return true ;
}
void COpenGLControl::Create(CRect rect, CWnd* parent)
{
	CString className = AfxRegisterWndClass(
		CS_HREDRAW | CS_VREDRAW | CS_OWNDC,
		NULL,
		(HBRUSH)GetStockObject(BLACK_BRUSH),
		NULL);
	CreateEx(
		0,
		className,
		"OpenGL",
		WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN,
		rect,
		parent,
		0);
}
void COpenGLControl::OnSize(UINT nType, int cx, int cy)
{
	CWnd::OnSize(nType, cx, cy);
	if (cy == 0)
```

```
        {
                cy = 1;
        }
        glViewport(0,0,cx,cy);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-8.0f,8.0f,-4.0f,4.0f);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
int COpenGLControl::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CWnd::OnCreate(lpCreateStruct) == -1)
                return -1;
        dc = new CClientDC(this);
        openGLDevice.CreateRenderingContext(dc->m_hDC);
        InitGL();
        return 0;

}
```

OpenGLDevice.cpp

```
#include "StdAfx.h"
#include "OpenGLDevice.h"
COpenGLDevice::COpenGLDevice(void)
{
        m_hRC = NULL;
        hDC   = NULL;
}
COpenGLDevice::COpenGLDevice(HDC hDC)
```

```
{
        CreateRenderingContext(hDC);
}
COpenGLDevice::~COpenGLDevice(void)
{
        destroy();
}
void COpenGLDevice::destroy(void)
{
        if(wglGetCurrentContext())
        {
                wglMakeCurrent(NULL,NULL);
        }
        if(m_hRC)
        {
                wglDeleteContext(m_hRC);
                m_hRC = NULL;
        }
}
bool COpenGLDevice::CreateRenderingContext(HDC hDC)
{
        if (!hDC)
        {
                return false;
        }
        if (!SetPixelFormatDes(hDC))
        {
                return false;
        }
        m_hRC = wglCreateContext(hDC);
```

```
        if(!m_hRC)
        {
                AfxMessageBox(_T("wglCreateContext failed"),MB_ICONWARNING);
                return false ;
        }
        bool  bResult = wglMakeCurrent(hDC,m_hRC);
        if(!bResult)
        {
                AfxMessageBox(_T("wglMakeCurrent failed"),MB_ICONWARNING);
                return false ;
        }
        COpenGLDevice::hDC = hDC;
        return true ;
}


bool COpenGLDevice::SetPixelFormatDes(HDC hDC)
{
        DEVMODE resolution;
        int m_intPixelFormat;
        EnumDisplaySettings(NULL, ENUM_CURRENT_SETTINGS, &resolution);
        PIXELFORMATDESCRIPTOR pixelFD;
        pixelFD.nSize                   = sizeof(PIXELFORMATDESCRIPTOR);
        pixelFD.nVersion                = 1;
        pixelFD.dwFlags                 = PFD_DRAW_TO_WINDOW|
        PFD_DOUBLEBUFFER|PFD_SUPPORT_OPENGL;
        pixelFD.iPixelType              = PFD_TYPE_RGBA;
        pixelFD.cColorBits              = 32;
        pixelFD.cRedBits                = 0;
```

```
pixelFD.cRedShift                = 0;
pixelFD.cGreenBits               = 0;
pixelFD.cGreenShift              = 0;
pixelFD.cBlueBits                = 0;
pixelFD.cBlueShift               = 0;
pixelFD.cAlphaBits               = 0;
pixelFD.cAlphaShift              = 0;
pixelFD.cAccumBits               = 0;
pixelFD.cAccumRedBits            = 0;
pixelFD.cAccumGreenBits          = 0;
pixelFD.cAccumBlueBits           = 0;
pixelFD.cAccumAlphaBits          = 0;
pixelFD.cDepthBits               = 16;
pixelFD.cStencilBits             = 0;
pixelFD.cAuxBuffers              = 0;
pixelFD.iLayerType               = PFD_MAIN_PLANE;
pixelFD.bReserved                = 0;
pixelFD.dwLayerMask              = 0;
pixelFD.dwVisibleMask            = 0;
pixelFD.dwDamageMask             = 0;

m_intPixelFormat = ChoosePixelFormat(hDC,&pixelFD);
if(m_intPixelFormat==0)
{
        m_intPixelFormat =1;
        if(DescribePixelFormat(hDC,m_intPixelFormat,sizeof(PIXELFORMATDESCR
        IPTOR),&pixelFD)==0)
        {
                AfxMessageBox(_T("Choose pixel format
                failed"),MB_ICONWARNING);
```

```
                return false;
            }
        }
        if(SetPixelFormat(hDC,m_intPixelFormat,&pixelFD)==FALSE)
        {
            AfxMessageBox(_T("Set pixel format failed"),MB_ICONWARNING);
            return false;
        }
        return true;
}
void COpenGLDevice::makeCurrent(bool  disableOther)
{
        if (m_hRC != NULL)
        {
            if (disableOther)
            wglMakeCurrent(NULL,NULL);

            wglMakeCurrent(hDC, m_hRC);
        }
}
```

Project4CDlg.cpp

```
#include "stdafx.h"
#include "Project4C.h"
#include "Project4CDlg.h"
#include "EditData.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
CProject4CDlg::CProject4CDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CProject4CDlg::IDD, pParent)
	, m_Fuel(0)
	, m_Temp(0)
	, m_Engine(0)
	, m_Velocity(0)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	m_wheel_width = 185;
	m_wheel_hight = 65;
	m_wheel_diameter = 14;
	m_wheel_sensor = 41;
	Time_v = 0;
	Time_r = 0;
	m_Fuel = 0;
	m_Temp = 0;
	m_Engine = 0;
	m_Velocity = 0;
}
void CProject4CDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDIT_FUEL, m_Fuel);
	DDX_Text(pDX, IDC_EDIT_TEMP, m_Temp);
	DDX_Text(pDX, IDC_EDIT_ENGINE, m_Engine);
	DDX_Text(pDX, IDC_EDIT_VELOCITY, m_Velocity);
}
BEGIN_MESSAGE_MAP(CProject4CDlg, CDialog)
	ON_WM_PAINT()
```

```
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_BN_CLICKED(IDC_BUTTON_EDIT, &CProject4CDlg::OnBnClickedButtonEdit)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_BUTTON_START,
&CProject4CDlg::OnBnClickedButtonStart)
	ON_BN_CLICKED(IDC_BUTTON_CLOSE,
&CProject4CDlg::OnBnClickedButtonClose)
END_MESSAGE_MAP()
BOOL CProject4CDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);
	CRect rect;
	GetDlgItem(IDC_STATIC_SIZE)->GetWindowRect(rect);
	GetDlgItem(IDC_STATIC_SIZE)->GetWindowRect(rect_update);
	ScreenToClient(rect);
	openGLControl.Create(rect,this);
	return TRUE;
}
void CProject4CDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);
		SendMessage(WM_ICONERASEBKGND,
		reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
```

```
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}
HCURSOR CProject4CDlg::OnQueryDragIcon()
{
        return static_cast<HCURSOR>(m_hIcon);
}


void CProject4CDlg::OnBnClickedButtonEdit()
{
        CEditData Edit ;
        Edit.DoModal();
        if(IDOK)
        {
                UpdateData(TRUE);
                m_wheel_width  = Edit.m_width;
                m_wheel_hight   = Edit.m_hight;
                m_wheel_diameter = Edit.m_diameter;
                m_wheel_sensor   = Edit.m_sensor;
        }
}
```

```
void CProject4CDlg::OnBnClickedButtonStart()
{
	FT_STATUS ftStatus ;
	ftStatus = FT_OpenEx("Astron Logic Ezy USB-
	M02",FT_OPEN_BY_DESCRIPTION,&ftHandle);
	if(ftStatus == FT_OK)
	{
		GetDlgItem(IDC_BUTTON_START)->EnableWindow(FALSE);
		FT_ResetDevice(ftHandle);
		FT_Purge(ftHandle,FT_PURGE_RX||FT_PURGE_TX);
		FT_ResetDevice(ftHandle);
		FT_SetTimeouts(ftHandle,3000,3000);
		Sleep(150);
		SetTimer(1,50,NULL);
		if (!SetTimer (1, 50, NULL))
		{
			MessageBox (_T ("Timer failed"), _T
			("Error"),MB_ICONSTOP|MB_OK);
		}
		FT_STATUS ftBaudRate = FT_SetBaudRate(ftHandle,9600);
			if (ftBaudRate != FT_OK)
			{
				AfxMessageBox(_T("Baud Rate Error"));
			}
		unsigned char txbuf;
		DWORD ret_bytes;
		txbuf = 0xff;
		ftStatus = FT_Write(ftHandle,&txbuf,1,&ret_bytes);
	}
	else
```

```
        {
                MessageBox(_T("failed"));
        }
}
void CProject4CDlg::OnTimer(UINT_PTR nIDEvent)
{
if(nIDEvent==1)
{
DWORD EventDword;
DWORD RxBytes;
DWORD TxBytes;
CString str;
if(FT_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDword)==FT_OK)
{
if(RxBytes>0)
{
FT_STATUS ftStatus;
DWORD BytesReceived;
unsigned char *RxBuffer;
RxBuffer = new unsigned char [RxBytes];
FT_SetTimeouts(ftHandle,500,0);
ftStatus = FT_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);
if(ftStatus == FT_OK)
        {
                if(BytesReceived == RxBytes)
                {
                        for(DWORD i=0;i<RxBytes;i++)
                        {
                                int iTmp;
                                int Buff,Buff_high,Buff_low,high,low;
```

```
GLfloat dat,dat1,dat2,value,freq1,freq2;
iTmp = (int)RxBuffer[i];
if((i == 0)&&(iTmp==0x41))
{
        Buff_high = RxBuffer[1];
        Buff_low  = RxBuffer[2];
        Buff_high <<= 8;
        Buff = Buff_high|Buff_low ;
        dat = Buff;
        freq1 = 1000000/(1.084*dat);
        openGLControl.sr = ((1000-freq1)/50)-2;
        m_Velocity = freq1;
        UpdateData(FALSE);
}
if((i==3)&&(iTmp==0x42))
{
        Buff_high = RxBuffer[4];
        Buff_low  = RxBuffer[5];
        Buff_high <<= 8;
        Buff = Buff_high|Buff_low ;
        dat = Buff;
        freq2 = 1000000/(1.084*dat);
        openGLControl.sc = ((1000-freq2)/50)+4 ;
        m_Engine  = freq2;
        UpdateData(FALSE);

}

if((i == 6)&&(iTmp==0x43))
```

```
        {
                Buff = (int)RxBuffer[7];
                high = Buff&0xF0;
                high >>=4;
                low = Buff&0x0F;
                dat1 = high;
                dat2 = low;
                value = ((dat1/16)+(dat2/256))*5.12;
                value = (value-0.1)*2.4;
                openGLControl.sl1 = ((12-value)*2)+24 ;
                m_Fuel = value;
                UpdateData(FALSE);
        }

        if((i == 8)&&(iTmp==0x44))
        {
                Buff = (int)RxBuffer[9];
                high = Buff&0xF0;
                high >>=4;
                low = Buff&0x0F;
                dat1 = high;
                dat2 = low;
                value = ((dat1/16)+(dat2/256))*5.12;
                value = (value-0.1)*2.4;
                openGLControl.sl2 = ((12-value)*2)+48 ;
                m_Temp = value;
                UpdateData(FALSE);
        }
}
        DWORD ret_bytes;
```

```
                    unsigned char txbuf;
                    txbuf = 0xff;
                    FT_STATUS ftStatus;
                    ftStatus = FT_Write(ftHandle,&txbuf,1,&ret_bytes);

                }delete [] RxBuffer;
            }
        }
        }
        InvalidateRect(&rect_update,false);
        }
            CDialog::OnTimer(nIDEvent);
        }
        void CProject4CDlg::OnBnClickedButtonClose()
        {
            FT_Close(ftHandle);
            KillTimer(1);
            GetDlgItem(IDC_BUTTON_START)->EnableWindow(TRUE);
            openGLControl.sr = 18;
            openGLControl.sc = 24;
            openGLControl.sl1 = 48;
            openGLControl.sl2 = 72;
            InvalidateRect(&rect_update,false);
            m_Velocity = 0 ;
            m_Engine   = 0 ;
            m_Fuel     = 0 ;
            m_Temp     = 0 ;
            UpdateData(FALSE);
```