การประยุกต์ใช้ไมโครคอนโทรลเลอร์ ARM7 เป็นตัวควบคุมพีไอดีสำหรับเตอบ

IMPLEMENTATION OF ARM 7 MICROCONTROLLER FOR PID CONTROLLER FOR OVEN

นายกฤษ พิพัฒนกุล
นางสาวดวงตะวัน ไตรภูวนาถ
นายมารุต หลายลักษณะดี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2551

IMPLEMENTATION OF ARM 7 MICROCONTROLLER FOR PID CONTROLLER FOR OVEN

KRIT PIPATTANAKHUN

DOUNGTAWAN TRIPOOWANAT

MAROOT LAILAKSANADEE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUE OF TECHNOLOGY LARDKRABANG

2008

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์	การประยุกต์ใช้ไมโครคอนโทรลเลอร์ ARM7 เป็นตัวควบคุมพีไอดีสำหรับเตาอบ IMPLEMENTATION OF ARM 7 MICROCONTROLLER FOR PID CONTROLLER FOR OVEN		
นักศึกษาผู้จัดทำ	นายกฤษ	พิพัฒนกุล	**รหัสนักศึกษา** 49015361
	นางสาวดวงตะวัน	ไตรภูวนาถ	**รหัสนักศึกษา** 49015368
	นายมารุต	หลายลักษณะดี	**รหัสนักศึกษา** 49015386
ปริญญา	วิศวกรรมศาสตรบัณฑิต		
สาขาวิชา	วิศวกรรมการวัดคุม		
ปีการศึกษา	2551		

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
ผู้ช่วยศาสตราจารย์ ประภาส เริงรื่น รองศาสตราจารย์ อาจินต์ น่วมสำราญ	

หัวข้อปริญญานิพนธ์ การประยุกต์ใช้ไมโครคอนโทรลเลอร์ ARM7 เป็นตัวควบคุมพีไอดี สำหรับเตาอบ

IMPLEMENTATION OF ARM 7 MICROCONTROLLER FOR PID CONTROLLER FOR OVEN

นักศึกษาผู้จัดทำ นายกฤษ พิพัฒนกุล **รหัสนักศึกษา** 49015361

นางสาวดวงตะวัน ไตรภูวนาถ **รหัสนักศึกษา** 49015368

นายมารุต หลายลักษณะดี **รหัสนักศึกษา** 49015386

อาจารย์ที่ปรึกษา ผู้ช่วยศาสตราจารย์ ประภาส เริงรื่น

รองศาสตราจารย์ อาจินต์ น่วมสำราญ

ปีการศึกษา 2551

บทคัดย่อ

ปริญญานิพนธ์เล่มนี้ได้นำเสนอวิธีการนำไมโครคอนโทรลเลอร์ ARM 7 มาประยุกต์ใช้เป็นตัวควบคุมพีไอดีเพื่อควบคุมกระบวนการเตาอบขนาดเล็ก โดยใช้ทฤษฎีของ Ziegler Nichols และ Trial and Error มาใช้ในการออกแบบเพื่อหาค่าพารามิเตอร์ของตัวควบคุม และใช้โปรแกรมภาษาซีในการพัฒนาไมโครคอนโทรลเลอร์ ARM 7 เพื่อการติดต่อสื่อสารระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ใช้ RS-232 ซึ่งเป็นการสื่อสารแบบอนุกรม (Serial Port) ส่วนในการแสดงผลตอบสนองของระบบใช้โปรแกรม Visual basic เข้ามาช่วยในการแสดงผล และ จะมีการจำลองกระบวนการด้วยโปรแกรม Simulink เพื่อเปรียบเทียบการทดลองจริง

Thesis Title Implementation of ARM 7 Microcontroller for PID Controller for Oven

Authors Mr.Krit Phiphatthanakul

Miss.Doungtawan Tripoowanat

Mr.Maroot Lailaksanadee

Thesis Advisor Asst. Prof. Prapas Roengruen

Assoc. Prof. Arjin Numsomran

Year 2008

ABSTRACT

This dissertation represents. The implementation of ARM7 microcontroller for PID controller to control oven procedure. According to Ziegler Nichols and Trial and Error theory, we apply it to compute the controller parameter. And use C language program to write the program for ARM7 microcontroller. Communication between computer and microcontroller uses RS-232 by using serial port communication. In term of display of usage Visual Basic respondent to assist and imitate the process with Simulink program for actual experiment comparison.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับคำปรึกษาและความอนุเคราะห์จาก ผศ. ประภาส เริงรื่น ซึ่งเป็นอาจารย์ผู้ควบคุมปริญญานิพนธ์ และ คณาจารย์ภาควิชาวิศวกรรมการวัดคุม

ขอขอบพระคุณ รศ. อาจินต์ น่วมสำราญ ซึ่งได้รับคำปรึกษาและ ให้ความช่วยเหลือในการทำโครงงานนี้อีกทั้งช่วยตรวจสอบแก้ไขข้อผิดพลาดต่าง ๆ พร้อมทั้งแนะนำวิธีแก้ปัญหาอย่างเต็มที่

ขอขอบพระคุณอาจารย์ภาควิชาวิศวกรรมการวัดคุมที่เอื้อเฟื้ออุปกรณ์และ เครื่องมือในการทดลอง และให้การช่วยเหลือ อีกทั้งให้คำปรึกษาเป็นอย่างดีจนสามารถทำปริญญานิพนธ์ฉบับนี้เสร็จสิ้นลงตามวัตถุประสงค์

คณะผู้จัดทำ

สารบัญ

หน้า

บทคัดย่อภาษาไทย.....I

บทคัดย่อภาษาอังกฤษ.....II

กิตติกรรมประกาศ.....III

สารบัญ.....IV

สารบัญตาราง.....VII

สารบัญภาพ.....IX

บทที่ 1 บทนำ.....1

1.1 ความสำคัญของปริญญานิพนธ์.....1

1.2 วัตถุประสงค์ของปริญญานิพนธ์.....1

1.3 ของเขตของปริญญานิพนธ์.....1

1.4 ขั้นตอนการศึกษา.....2

1.5 ประโยชน์ที่คาดว่าจะได้รับ.....2

บทที่ 2 ไมโครคอนโทรลเลอร์ ARM7.....3

2.1 สถาปัตยกรรม ARM7.....3

2.1.1 การติดต่อพอร์ตและการควบคุมการทำงานของพอร์ตอเนกประสงค์ (General Purpose I/O: GPIO).....4

2.1.2 การติดต่อและการเขียนโปรแกรมสั่งงานวงจรแปลงอนาลอกเป็นดิจิตอล(A/D).....9

2.1.3 การติดต่อและการเขียนโปรแกรมสั่งงานของวงจรแปลงดิจิตอลเป็นอนาลอก (D/A).....17

2.1.4 เขียนโปรแกรมติดต่อกับ REAL TIME CLOCK.....21

2.1.5 การอินเตอร์รัปต์ (Interrupt).....24

2.1.6 การติดต่อและสั่งงาน Display 7-Segment 8 Digit กับไมโครคอนโทรลเลอร์ ARM731.....27

สารบัญ(ต่อ)

หน้า

2.1.7 คำสั่งควบคุมโมดูลตัวแสดงผลแบบผลึกเหลว (LCD interface) 34
2.2 การใช้งานโปรแกรม KEIL uVision 3 37
2.3 การ Download Hex file ให้กับ MCU ของบอร์ด.....39

บทที่ 3 ทฤษฎีตัวควบคุมPID.....43
3.1 การควบคุมแบบป้อนกลับด้วยตัวควบคุมพีไอดี.....45
3.2 กริยาการควบคุมแบบป้อนกลับ.....45
3.2.1 กริยาการควบคุม ON-OFF.....45
3.2.2 กริยาการควบคุมแบบ Proportional(P).....46
3.2.3 กริยาการควบคุมแบบ Integral (I).....47
3.2.4 กริยาการควบคุมแบบ Derivative (D).....48
3.2.5 กริยาการควบคุมแบบ Proportional-Integral(PI).....49
3.2.6 กริยาการควบคุมแบบ Proportional-Derivative(PD).....49
3.2.7 กริยาการควบคุมแบบ Proportional-Integral-Derivative(PID).....50
3.3 การคำนวณหาค่าพารามิเตอร์ของตัวควบคุม PID จากผลตอบสนองขอกระบวนการ.....51
3.3.1 การปรับค่าพารามิเตอร์ของตัวควบคุม PID โดยวิธีของ Ziegler-Nichols.....51
3.3.1.1 วิธี Process Reaction Curve.....52
3.3.1.2 วิธี Ultimate Method.....53
3.3.1.3 วิธี Trial and Error (วิธีการลองผิดลองถูก).....54
3.3.2 การปรับค่าพารามิเตอร์ของตัวควบคุม PID โดยวิธี Damped Oscillation.....55
3.4 ข้อกำหนด (Specifications) ของผลตอบสนองชั่วครู่ของระบบ.....56

บทที่ 4 การออกแบบตัวควบคุม.....58
4.1 การออกแบบ HardWare.....59
4.1.1 เพาเวอร์ซัพพลายชนิดสอง.....59
4.1.2 วงจรปรับระดับแรงดัน.....60

สารบัญ(ต่อ)

หน้า

4.1.3 วงจรแปลงสัญญาณแรงดันไฟฟ้าเป็นสัญญาณกระแสไฟฟ้า.....62

4.1.4 การออกแบบคีย์บอร์ดควบคุ(SW) 63

4.1.5 การออกแบบตัวแสดงผล.....64

4.1.5.1 ตัวแสดงผล 7-Segment.....64

4.1.5.2 ตัวแสดงผลแบบผลึกเหลว(LCD).....65

4.1.5.3 ตัวแสดงผลLED.....67

4.2 การออกแบบ SoftWare.....67

4.2.1 การออกแบบ อัลกอริทึม สำหรับตัวควบคุม PID ซึ่งไม่ต่อเนื่องทางเวลา.....67

4.2.2 การเลือกช่วงเวลาในการสุ่มค่า (Choice of Sampling Interval).....70

4.2.3 การแปลงสัญญาณอนาลอกเป็นดิจิตอล.....72

4.2.4 การแปลงสัญญาณดิจิตอลเป็นอนาลอก.....73

4.2.5 การ Scaling.....74

บทที่ 5 การทดลองและผลการทดลอง.....75

5.1 กระบวนการที่ใช้ในการทดสอบตัวควบคุม PID.....75

5.2 การทดลองตัวควบคุม PID โดยวิธี Process Reation Curve ของ (Ziegler-Nichols).....76

5.3 การทดลองตัวควบคุม PID โดยวิธี Trial&Error.....82

บทที่ 6 สรุปผลการวิจัยและข้อเสนอแนะ.....85

6.1 สรุปผลการการทดลอง.....85

6.2 ปัญหาและอุปสรรค์.....85

6.3 ข้อเสนอแนะ.....86

บรรณานุกรม.....87

ภาคผนวก ก.....88

ภาคผนวก ข.....126

สารบัญ(ต่อ)

หน้า

ภาคผนวก ค.....130

สารบัญตาราง

ตารางที่ **หน้า**

2.1 GPIO Pin Function Descriptions....4

2.2 GPxCON Registers....6

2.3 GPxCON MMR Bit Descriptions....6

2.4 GPxDAT Registers....7

2.5 GPxDAT MMR Bit Descriptions....8

2.6 GPxSET Registers....8

2.7 GPxSET MMR Bit Descriptions....8

2.8 GPxCLR Registers....9

2.9 GPxCLR MMR Bit Descriptions....9

2.10 กำหนดค่าคุณสมบัติให้กับ ADC....11

2.11 ADCCON MMR Bit Designations....12

2.12 ADCCP Register....13

2.13 ADCCP MMR Bit Designation....14

2.14 ADCCN Register....15

2.15 ADCCN MMR Bit Designation....15

2.16 ADCDAT Register....15

2.17 ADCRST Register....16

2.18 ADCGN Register....16

2.19 DACxCON Registers....20

2.20 DAC0CON MMR Bit Designations....20

2.21 DACxDAT Registers....20

2.22 DAC0DAT MMR Bit Designations....21

2.23 แสดงแอดเดรสของ Register ต่างๆ ใน DS1307....22

2.24 IRQ/FIQ MMRs Bit Description....24

2.25 IRQSTA Register....24

2.26 IRQSIG Register....24

2.27 IRQEN Register....24

2.28 รูปแบบการส่งข้อมูลแบบอนุกรม (16 บิต)....29

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

2.29 Register Address Map 29

2.30 Decode-Mode (Address(HEX)=x0X9) 29

2.31 โหมด Decode แบบรหัสเลขฐานสอง 32

2.32 โหมดการปรับความเข้มของ 7 Segment (Address (HEX) =0xXA) 33

2.33 โหมด Scan-Limit (Address (HEX) =0xXB).....34

2.34 การใช้งานขาต่างๆ.....35

2.35 การรับเขียนอ่านข้อมูล. 35

2.36 คำสั่งการควบคุม LCD 36

3.1 แสดงค่าพารามิเตอร์ของตัวควบคุมแบบต่างๆตามวิธี Process Reaction Curve.....53

3.2 แสดงค่าพารามิเตอร์ของตัวควบคุมแบบต่างๆตามวิธี Ultimate Method.....54

4.1 การหาขนาดของหม้อแปลง 60

สารบัญภาพ

ภาพที่ **หน้า**

2.1 แสดงการจัดขาของ ADUC7024 4

2.2 แสดงรูปสัญญาณเมื่อทำงานอยู่ในโหมด Fully Difference 10

2.3 Pseudo Differential and Single-Ended Modes 10

2.4 การแปลงสัญญาณ ADC เมื่อทำงานอยู่ในโหมด Differential 11

2.5 การแสดงผลลัพท์ข้อมูล 16

2.6 วงจรแปลงดิจิตอลเป็นอนาลอก (D/A) 17

2.7 ระบบการแปลงสัญญาณดิจิตอลเป็นอะนาล็อก.....17

2.8 Transfer Curve ในอุดมคติของ DAC 3 บิต 18

2.9 คลื่นซายน์ที่สร้างจาก DAC 19

2.10 ลักษณะวงจรการต่อสายและรายละเอียดขาสัญญาณต่างๆ.....21

2.11 Timing Diagram การส่งข้อมูล.....22

2.12 โหมดการเขียนข้อมูล.....23

2.13 การอ่านข้อมูล.....23

2.14 ภาพโครงสร้างของ Timer.....26

2.15 วงจร Display 7-Segment 8 Digit กับไมโครคอนโทรลเลอARM7.....27

2.16 Timing Diagram ของ MAX 7219.....28

2.17 โหมดการทำงานปกติ.....31

2.18 Schematic ของการต่อจอแสดงผล LCD กับไมโครคอนโทรลเลอร์ ADuC 7024.....34

2.19 Timing Diagram แสดงการเขียนข้อมูล.....36

2.20 Timing Diagram แสดงการอ่านข้อมูล.....37

2.21 การเปิดโปรแกรม u Vision3.....37

2.22 การสร้างไฟล์โปรเจคและเลือก ARM 7 เบอร์ ADuV 7024.....38

2.23 การเขียน source code และเพิ่มเข้าไปใน source file 38

2.24 การสร้างโปรเจคและ HEX ไฟล์สำหรับ PROM 39

2.25 ผลการสั่ง Run โปรแกรม “ARMWSD” 39

2.26 การเลือกกำหนดหมายเลข Comport 40

2.27 การเลือกกำหนดหมายเลข Comport 40

2.28 การจะ Download ให้กับ MCU มารอไว้ใน Buffer 41

สารบัญภาพ(ต่อ)

ภาพที่ **หน้า**

2.29 การเลือกกำหนดชื่อ HEX File 41

2.30 ข้อความ บอกให้ผู้ใช้รีเซ็ต MCUให้ทำงานใน Monitor Mode 41

2.31 การเริ่มต้นทำการ Download HEX File 42

2.32 การคลิกเมาส์ที่ปุ่ม "Run"เพื่อให้ MCU เริ่มต้นทำงาน.....42

3.1 แสดงโครงสร้างของระบบควบคุมแบบป้อนกลับโดยทั่วไป 44

3.2 แสดงกริยาการควบคุมแบบ ON-OFF 45

3.3 แสดงคุณสมบัติของกริยาการควบคุมแบบ Proportional.....47

3.4 แสดงผลตอบสนองของกริยาการควบคุมแบบ Integral 48

3.5 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ Derivative 48

3.6 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบPI (Direct action) 49

3.7 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ PD.....50

3.8 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ PI (Direct action) 51

3.9 แสดงค่าพุ่งเกินที่ต้องการเมื่อใช้การปรับด้วยวิธี Ziegler-Nichols 52

3.10 แสดงผลตอบสนองรูปตัว S เมื่อปรับโดยใช้วิธี Process Reaction Curve 52

3.11 แสดงผลตอบสนองเวลาเกิดการแกว่งอย่างต่อเนื่อง เมื่อปรับโดยใช้วิธี Ultimate method.....53

3.12 แสดงค่าอัตราการเสื่อม 1/4เมื่อปรับโดยวิธี Damped Oscillation.....55

3.13 แสดงข้อกำหนดของผลตอบสนองชั่วครู่ของระบบ.....57

4.1 แสดงตัวควบคุมPID 58

4.2 แสดงฮาร์ดแวร์ตัวควบคุมโดยรวม.....59

4.3 แสดงวงจรเพาเวอร์ซับพลาย.....59

4.4 แสดงวงจรปรับระดับแรงดัน.....60

4.5 วงจรแปลงสัญญาณแรงดันไฟฟ้าเป็นสัญญาณกระแสไฟฟ้า.....62

4.6 Schematic และการทำงานของSW.....64

4.7 Schematic ของ 7-Segment.....64

4.8 การแสดงภาพผลของ 7-Segment บนตัวควบคุม PID 65

4.9 Schematicของการต่อจอแสดงผ LCDกับไมโครคอนโทรลเลอร์ ADuc7024.....66

4.10 การแสดงผลของ LED บนตัวควบคุม PID 66

สารบัญภาพ(ต่อ)

ภาพที่ **หน้า**

4.11 Schematic และการแสดงผลของLED.. 67

4.12 แสดงให้เห็นเสถียรภาพระหว่างตัวควบคุมแบบ PID และ PYID.....71

5.1 แสดงภาพเตาอบไฟฟ้า.....75

5.2 แสดงลักษณะการออกแบบกระบวนการ.....76

5.3 แสดงผลตอบสนองรูปตัวS เมื่อใช้วิธี Process Reaction Curve 77

5.4 แสดงผลตอบสนองรูปตัว S จากการทดลองลูปเปิดที่ MV=20% ไปเป็น 60%.....78

5.5 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) Setpoint = 60%.....79

5.6 แสดงผลตอบสนองของตัวควบคุมPI (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60 % เป็น 80 %.....79

5.7 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance 80

5.8 แสดงผลตอบสนองของตัวควบคุม PID (Ziegler-Nichols) Setpoint = 60%.....80

5.9 แสดงผลตอบสนองของตัวควบคุม PID (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%.....81

5.10 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance.....81

5.11 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error) Setpoint = 60%.....82

5.12 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%.....82

5.13 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error) เมื่อ เปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance 83

5.14 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error) Setpoint = 60%.....83

5.15 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%.....84

5.16 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance.....84

บทที่ 1
บทนำ

1.1 ความสำคัญของปริญญานิพนธ์

ระบบควบคุมอัตโนมัติแบบ PID นั้น ได้ถูกนำมาใช้ควบคุมกระบวนการในการผลิตเป็นเวลานานกว่า 50 ปีมาแล้ว ซึ่งในอุตสาหกรรมการผลิตแบบอัตโนมัติทั่วไปในปัจจุบันก็ยังนิยมใช้ตัวควบคุมแบบ PID อย่างแพร่หลาย ทั้งนี้เพราะรูปแบบของตัวควบคุม PID เป็นรูปแบบที่สามารถควบคุมกระบวนการต่างๆได้อย่างกว้างขวาง

ในอดีตตัวควบคุมเป็นแบบอนาลอก สร้างขึ้นด้วยอุปกรณ์ที่เป็นอนาลอกเช่น ออปแอมป์ แต่ในปัจจุบันได้มีการพัฒนาทางด้านไมโครโปรเซสเซอร์เพิ่มขึ้นอย่างมากมาย ทำให้เป็นไปได้ที่จะสร้างตัวควบคุมโดยใช้ไมโครคอนโทรลเลอร์ที่มีขนาดเล็กที่ซึ่งหาง่ายและ ราคาไม่แพงมากนัก อีกทั้ง ยังให้ประสิทธิภาพในการทำงานที่ดีขึ้น

ส่วนการปรับและการออกแบบค่าพารามิเตอร์ของตัวควบคุม PID ซึ่งได้พัฒนาขึ้นมาพร้อมกับการมีตัวควบคุม PID ออกมากันมากมาย การปรับค่าพารามิเตอร์ของตัวควบคุม PID จึงเป็นสิ่งที่จำเป็นเพื่อให้ได้การควบคุมที่ดีที่สุดซึ่งทำได้หลายวิธี บางวิธีจะอาจพิจารณาจากผลตอบสนองของระบบ และอาศัยประสบการณ์ บางวิธีอาศัยหลักการทางคณิตศาสตร์และส่วนใหญ่จะใช้การพิจารณาจากเงื่อนไขในโดเมนเวลามากกว่าในโดเมนความถี่ จนถึงปัจจุบันวิธีที่ยังได้รับความนิยมอยู่วิธีหนึ่งคือ วิธีของ Ziegler-Nichols

ซึ่งสำหรับปริญญานิพนธ์นี้ จะนำเอาไมโครคอนโทรลเลอร์ ARM7 มาประยุกต์สร้างเป็นตัวควบคุม PID สำหรับกระบวนการทางอุตสาหกรรม เพื่อให้เป็นการเพิ่มความน่าเชื่อถือและเพิ่มศักยภาพในการปฏิบัติงานของตัวควบคุม

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาโครงสร้างของระบบควบคุมแบบ PID
2. เพื่อออกแบบตัวควบคุมแบบ PID ที่สามารถควบคุมระบบ ให้มีความเสถียรภาพและมีสมรรถนะที่ดี
3. สร้างตัวควบคุม PID โดยใช้ไมโครคอนโทรลเลอร์ ARM7

1.3 ขอบเขตของปริญญานิพนธ์

1. ศึกษาโครงสร้างของระบบควบคุมแบบ PID รวมทั้งวิธีการหาค่าพารามิเตอร์ โดยได้ใช้วิธีZiegler-Nichols, Root locus และ Trial & Error

2. วิเคราะห์และออกแบบระบบควบคุมในโดเมนเวลา โดยใช้โปรแกรม MATLAB รวมทั้งจำลองระบบควบคุมใน Simulink

3. ศึกษาถึงสถาปัตยกรรมภายในและ การใช้งานของฟังก์ชั่นต่าง ๆ โดยใช้วิธีการควบคุมไมโครคอนโทรลเลอร์ ARM 7

4. สร้างตัวควบคุม PID บนไมโครคอนโทรลเลอร์ ARM 7

5. ทดลอง ตัวควบคุม PID กับกระบวนการจริง(กระบวนการเตาอบ)

1.4 ขั้นตอนการศึกษา

1. ศึกษาวิธีการออกแบบตัวควบคุม PID
2. ศึกษาวิธีการหาค่าพารามิเตอร์ของตัวควบคุมด้วยวิธีต่าง ๆ
3. ศึกษาสถาปัตยกรรมภายใน รวมทั้งฟังก์ชั่นการทำงานต่างๆของไมโครคอนโทรลเลอร์ ARM7
4. ศึกษาการเขียนโปรแกรมภาษาซี ซึ่งใช้สำหรับควบคุมการทำงานไมโครคอนโทรลเลอร์ ARM7
5. ศึกษาการเขียนโปรแกรม Visual Basic เพื่อแสดงผลกราฟบนจอคอมพิวเตอร์
6. ศึกษาฟังก์ชั่นการทำงานของตัวควบคุม PID
7. ศึกษาการทำงานของกระบวนการเตาอบ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ปริญญานิพนธ์นี้ จะช่วยให้เข้าใจการควบคุมการทำงานของกระบวนการทางอุตสาหกรรมโดยใช้ตัวควบคุมแบบ PID ได้ดียิ่งขึ้น ซึ่งสามารถนำไปพัฒนาเพื่อใช้ในการควบคุมการทำงานของกระบวนการทางอุตสาหกรรมในรูปแบบต่าง ๆ ได้ เช่น ควบคุมระดับของเหลว ควบคุมความดัน หรือควบคุมอัตราการไหลเป็นต้น ซึ่งเป็นการง่ายในการพัฒนา เนื่องจากตัวควบคุมถูกสร้างมาจากไมโครคอนโทรลเลอร์ อีกทั้งหลักการควบคุมกระบวนการแบบ PID ยังเป็นหลักการที่มีประสิทธิภาพในการนำไปใช้ควบคุมอีกด้วยโครงงานนี้จะเป็นจุดเริ่มต้นในการใช้งานไมโครคอนโทรลเลอร์ ARM7 มาใช้ทำการประมวลผล และยังเป็นแนวทางแก่ผู้สนใจที่จะนำความรู้ของโครงงานนี้ไปใช้ในการพัฒนาในอนาคตต่อไป

บทที่ 2
ไมโครคอนโทรลเลอร์ ARM7

2.1 สถาปัตยกรรม ARM7

สถาปัตยกรรมของ ARM7 เป็นซีพียูแบบ RISC มีขนาด 32 บิต ภายในมีบัสขนาด 32 บิต ซึ่งตัวเดียวที่เราใช้สำหรับรับข้อมูลและ โดยชุดคำสั่งจะมีขนาด 32 บิตคงที่ในขณะที่ข้อมูลสามารถเลือกได้ว่าจะมีขนาด 8, 16 หรือ 32 บิต

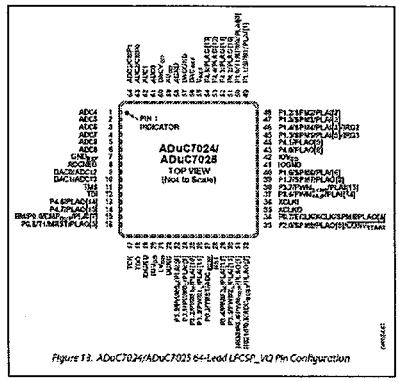
โครงสร้างของ ARM7 จะเป็นแบบที่เรียบง่าย มีชุดคำสั่งไม่มากนักและ จะประหยัดพื้นที่สารกึ่งตัวนำที่ใช้สร้างประหยัดพลังงาน

สถาปัตยกรรมของ ARM7 จะเป็นแบบ load-and-store ในการประมวลผลข้อมูลใดๆ ต้องกระทำผ่านทางรีจิสเตอร์เริ่มต้นด้วยการโหลดค่าจากหน่วยความจำมาเก็บในรีจิสเตอร์ แล้วนำค่ามาประมวลผลเสร็จแล้วจะเขียนค่าเก็บในหน่วยความจำดังเดิม

รีจิสเตอร์ของ ARM7 ที่ใช้งานได้นั้นสำหรับผู้ใช้มีทั้งหมด 16 ตัวคือ R0-R15 โดยทุกตัวมีขนาด 32 บิตโดย R0-R12 ซึ่งรีจิสเตอร์ทั่วไปไม่ได้กำหนดหน้าที่การทำงานพิเศษ สำหรับ R12ทำหน้าที่เป็น stack pointer (SP) , R14 ทำหน้าที่เป็น link register (LR) และ R15 ทำหน้าที่เป็น Program Counter (PC)

ความสามารถของไมโครคอนโทรลเลอร์ ANALOG DEVICES ADUC7024 พอสรุปได้ดังนี้

1. ไมโครคอนโทรลเลอร์ขนาด 16/32 บิต ARM7TDMI ในตัวถัง LQFP 64 ขา
2. หน่วยความจำ Static RAM ขนาด 8 kB
3. หน่วยความจำ Flash Program Memory ขนาด 62 kB
4. วงจรแปลงแอนะลอกเป็นดิจิตอล(ADC)ความละเอียด 12 bit จำนวน 10 ชุด
5. วงจรแปลงดิจิตอลเป็นแอนะลอก(DAC)ความละเอียด 12 bit จำนวน 2 ชุด
6. มีขาที่ใช้เป็นขาอินพุต เอาต์พุต จำนวน 40 ขา
7. PWM (Pulse width modulation) 6 เอาต์พุต
8. วงจรสื่อสารอนุกรม I^2C ความเร็วสูง
9. มีวงจรสื่อสารอนุกรม UART จำนวน 1 พอร์ต และวงจรสื่อสาร SPI จำนวน 1 พอร์ต



ภาพที่ 2.1 แสดงการจัดขาของ ADUC7024

2.1.1 การติดต่อพอร์ตและการควบคุมการทำงานของพอร์ตอเนกประสงค์ (General Purpose I/O : GPIO)

ADUC7024 มีพอร์ตอเนกประสงค์ (General Purpose I/O) ขนาด 32 บิตให้ใช้งาน 5 พอร์ต คือ P0-P4 โดยแต่ละพอร์ตจะมี8ขา PX.0-PX.7 รวม 40ขา ทั้งหมดทำหน้าที่เป็นได้ทั้งอินพุตพอร์ตและเอาต์พุตพอร์ต

แต่เนื่องจากแต่ละขาของพอร์ต P0-P4 มีหน้าที่การทำงานได้หลายแบบจึงต้องมีการกำหนดหน้าที่การทำงานของแต่ละพอร์ตโดยรีจิสเตอร์ดังนี้

ตารางที่ 2.1 GPIO Pin Function Descriptions

Port	Pin	Configuration			
		00	01	10	11
0	P0.0	GPIO	CMP	MS2	PLA[7]
	P0.1	GPIO	PWM2H	BLE	
	P0.2	GPIO	PWM2L	BHE	
	P0.3	GPIO	TRUST	A16	ADCbusy
	P0.4	GPIO/IRQ0	PWMtmp	MS1	PLAO[1]
	P0.5	GPIO/IRQ1	ADCbusy	MS0	PLAO[2]
	P0.6	GPIO/T1	MRST	AE	PLAO[3]
	P0.7	GPIO	ECLK/XCLK'	SIN	PLAO[4]

ตารางที่ 2.1 (ต่อ) GPIO Pin Function Descriptions

Port	Pin	Configuration			
		00	01	10	11
1	P1.0	GPIO/T1	SIN	SCLO	PLA[0]
	P1.1	GPIO	SOUT	SDAO	PLA[1]
	P1.2	GPIO	RTS	SCL1	PLA[2]
	P1.3	GPIO	CTS	SDA1	PLA[3]
	P1.4	GPIO/IRQ2	RI	CLK	PLA[4]
	P1.5	GPIO/IRQ3	DCD	MISO	PLA[5]
	P1.6	GPIO	DSR	MOSI	PLA[6]
	P1.7	GPIO	DTR	CSL	PLAO[0]
2	P2.0	GPIO	CONVstart	SOUT	PLAO[5]
	P2.1	GPIO	PWMOH	WS	PLAO[6]
	P2.2	GPIO	PWMOL	AS	PLAO[7]
	P2.3	GPIO		AE	
	P2.4	GPIO	PWMOH	MS0	
	P2.5	GPIO	PWMOL	MS1	
	P2.6	GPIO	PWM1H	MS2	
	P2.7	GPIO	PWM1L	MS3	
3	P3.0	GPIO	PWMOH	AD0	PLA[8]
	P3.1	GPIO	PWMOL	AD1	PLA[9]
	P3.2	GPIO	PWM1H	AD2	PLA[10]
	P3.3	GPIO	PWM1L	AD3	PLA[11]
	P3.4	GPIO	PWM2H	AD4	PLA[12]
	P3.5	GPIO	PWM2L	AD5	PLA[13]
	P3.6	GPIO	PWMtmp	AD6	PLA[14]
	P3.7	GPIO	PWMsytic	AD7	PLA[15]

ตารางที่ 2.1 (ต่อ) GPIO Pin Function Descriptions

Port	Pin	Configuration			
		00	01	10	11
4	P4.0	GPIO		AD8	PLA[8]
	P4.1	GPIO		AD9	PLA[9]
	P4.2	GPIO		AD10	PLA[10]
	P4.3	GPIO		AD11	PLA[11]
	P4.4	GPIO		AD12	PLA[12]
	P4.5	GPIO		AD13	PLA[13]
	P4.6	GPIO		AD14	PLA[14]
	P4.7	GPIO		AD15	PLA[15]

ตารางที่ 2.2 GPxCON Registers

Name	Address	Default Value	Access
GP0CON	0xFFFFF400	0x00000000	R/W
GP1CON	0xFFFFF404	0x00000000	R/W
GP2CON	0xFFFFF408	0x00000000	R/W
GP3CON	0xFFFFF40C	0x00000000	R/W
GP4CON	0xFFFFF410	0x00000000	R/W

ตารางที่ 2.3 GPxCON MMR Bit Descriptions

Bit	Description
31:30:00	Reservde
29:28:00	Select Function of Px.7Pin
27:26:00	Reserved
25:24:00	Select Function of Px.6Pin
23:22	Reserved
21:20	Select Function of Px.5Pin
19:18	Reserved

ตารางที่ 2.3 (ต่อ) GPxCON MMR Bit Descriptions

Bit	Description
17:16	Select Function of Px.4Pin
15:14	Reserved
13:12	Select Function of Px.3Pin
11:10	Reserved
9:08	Select Function of Px.2Pin
7:06	Reserved
5:04	Select Function of Px.1Pin
3:02	Reserved
1:00	Select Function of Px.0Pin

ค่าแต่ละบิตของรีจิสเตอร์ GPxCON ซึ่งแสดงได้ในตารางที่ 2.3 จากตารางจะพบว่าแต่ละขาจะมีหน้าที่การทำงานได้ถึง 3 หรือ 4 หน้าที่ ตัวอย่างเช่น ขา P0.0 จะมีหน้าที่ในการทำงานได้ถึง 4 หน้าที่ คือ GPIO0.0 , CMP , MS0 และ PLAI [7] ถ้าต้องการกำหนดค่าให้ขาของ P0.0 - P0.7 มีการทำงานเป็น GPIO ทำได้โดยการเขียนค่า 0 ทุกบิตให้กับ GP0CON ซึ่งเขียนเป็นภาษาซี ได้ดังนี้

```
GP0CON=0x00000000;          //Set P0.0 – P0.7 to GPIO Function
```

ซึ่งจากด้านบนนั้นเป็นการกำหนดค่าควบคุม ซึ่งเป็นการทำงานของพอร์ตอเนกประสงค์ (GPIO)

หลังจากที่กำหนดค่าให้พอร์ตของแต่ละตัว มีการทำงานเป็น GPIO แล้วในการควบคุมการทำงานของ GPIO จะสั่งงานผ่านทางรีจิสเตอร์ 4 ตัวได้แก่ GPxPAR, GPxDAT, GPxSET และ GPxCLR โดยรีจิสเตอร์แต่ละตัวจะมีแอดเดรสแสดงในตาราง

ตารางที่ 2.4 GPxDAT Registers

Name	Address	Default Value	Access
GP0DAT	0xFFFFF420	0x000000XX	R/W
GP1DAT	0xFFFFF430	0x000000XX	R/W
GP2DAT	0xFFFFF440	0x000000XX	R/W
GP3DAT	0xFFFFF450	0x000000XX	R/W
GP4DAT	0xFFFFF460	0x000000XX	R/W

ตารางที่ 2.5 GPxDAT MMR Bit Descriptions

Bit	DESCRIPTION
31:24:00	Direction of the Data. Set to 1 by user to configure the GPIO pin as an output. Cleared to 0 by user to configure the GPIO pin as an input.
23:16	Port x Data Output.
15:08	Reflect the State of Port x Pins at Reset (read only).
7:00	Port x Data input (read only).

การสั่งงานของ GPIO นั้นเราเริ่มต้นด้วยการกำหนดทิศทางของพอร์ตก่อนว่าจะให้เป็นอินพุตพอร์ตหรือเอาต์พุต โดยการกำหนดค่าที่รีจิสเตอร์ GPxDAT โดยค่าบิตใดที่เป็น 0 คือให้ขาของบิตนั้นเป็นอินพุต ถ้าให้ค่าเป็น 1 ขาของบิตนั้นจะเป็นเอาต์พุตตัวอย่าง เช่น ต้องการให้ขาของ P4.0-P4.3 เป็น input และ P4.4-P4.7 เป็น Output GP4DAT = 0xF0F00000 ;

หลังจากกำหนดให้พอร์ตเป็นเอาต์พุตแล้ว ถ้าต้องการสั่งให้พอร์ตที่เป็นเอาต์พุตนี้มีค่าเป็น 1 ต้องสั่งที่รีจิสเตอร์ GPxSET ตัวอย่างเช่น ต้องการให้ขา P4.4 และ P4.5 เป็น 1 จะต้องกำหนดค่าให้กับรีจิสเตอร์ GPxSET ดังนี้ GP4SET=0x00300000;

ตารางที่ 2.6 GPxSET Registers

Name	Address	Default Value	Access
GP0SET	0xFFFFF424	0x000000XX	W
GP1SET	0xFFFFF434	0x000000XX	W
GP2SET	0xFFFFF444	0x000000XX	W
GP3SET	0xFFFFF454	0x000000XX	W
GP4SET	0xFFFFF464	0x000000XX	W

ตารางที่ 2.7 GPxSET MMR Bit Descriptions

Bit	DESCRIPTION
31:24:00	Reserved.
23:16	Data PortxSet Bit. Set to 1 by user to set bit on portx;also sets the corresponding bit in the GPxDAT MMR. Cleared to 0by user; does not affect the data out.
15:00	Reserved

การเขียนค่าเป็น 0 ให้รีจิสเตอร์ GP4SET จะไม่มีผลต่อขาของพอร์ตที่ตรงกับบิตนั้นขาของพอร์ตจะมีค่าคงเดิม ถ้าต้องการสั่งให้พอร์ตที่เป็นเอาต์พุตนี้มีค่าเป็น 0 ต้องสั่งที่รีจิสเตอร์ GPxCLR ตัวอย่าง ต้องการสั่งให้ขา P4.5 และ P4.7 เป็น 0 จะต้องกำหนดค่าให้กับรีจิสเตอร์ GP4CLR ดังนี้

GP4CLR=0x00A00000;

ตารางที่ 2.8 GPxCLR Registers

Name	Address	Default Value	Access
GP0CLR	0xFFFFF428	0x000000XX	W
GP01LR	0xFFFFF438	0x000000XX	W
GP2CLR	0xFFFFF448	0x000000XX	W
GP3CLR	0xFFFFF458	0x000000XX	W
GP4CLR	0xFFFFF468	0x000000XX	W

ตารางที่ 2.9 GPxCLR MMR Bit Descriptions

Bit	DESCRIPTION
31:24:00	Reserved.
23:16	Data PortxSet Bit. Set to 1 by user to clear bit on portx;also clear the corresponding bit in the GPxDAT MMR. Cleared to 0by user; does not affect the data out.
15:00	Reserved

ข้อควรระวัง ในการสั่งให้ขาพอร์ตมีค่าเป็น 0 นี้ไม่ได้สั่งให้เขียนค่าของ 0 ให้กับรีจิสเตอร์ GPxCLR ต้องเขียนเป็น 1 เท่านั้น

2.1.2 การติดต่อและการเขียนโปรแกรมสั่งงานของวงจรแปลงแอนะลอกเป็นดิจิตอล(A/D)

ในปัจจุบันมีการนำไมโครโปรเซสเซอร์ไมโครคอนโทรลเลอร์และ คอมพิวเตอร์เข้ามาช่วยในการควบคุมอุปกรณ์ต่าง ๆ มากมาย ซึ่งทำให้การควบคุมนั้นทำได้ง่าย สะดวก และรวดเร็วยิ่งขึ้น แต่ในการควบคุมนั้นเราจำเป็นจะต้องใช้สัญญาณดิจิตอล ซึ่งในการติดต่อกับไมโครโปรเซสเซอร์ ไมโครคอนโทรลเลอร์ หรือคอมพิวเตอร์ แต่สัญญาณที่ใช้ในการควบคุมอุปกรณ์หรือกระบวนการต่าง ๆ นั้นจะเป็นสัญญาณแบบอนาล็อก ดังนั้นจึงจำเป็นอย่างมากที่จะต้องให้มีการเปลี่ยนแปลงค่าสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลเสียก่อน เพื่อจะให้ได้เป็นสัญญาณของไมโครโปรเซสเซอร์

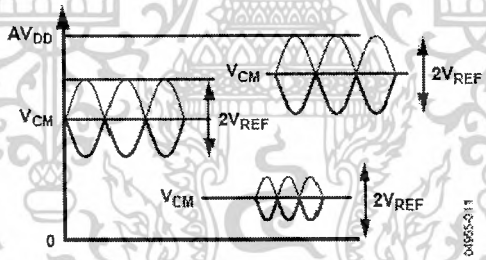
ไมโครคอนโทรลเลอร์หรือคอมพิวเตอร์ที่รู้จักกันแล้วนำจึงสัญญาณที่ได้ไปให้ไมโครโปรเซสเซอร์ไมโคคอนโทรลเลอร์ หรือคอมพิวเตอร์เพื่อใช้ในการประมวลผลเพื่อควบคุมระบบต่อไป

แม้ว่าสัญญาณอนาล็อกนั้นมีความแน่นอนและ แม่นยำสูงแต่สัญญาณอนาล็อกนั้นก็ควบคุมได้ยาก เนื่องจากในสภาพแวดล้อมมีสัญญาณรบกวนอยู่มากและ การที่จะทำให้ระบบการควบคุมแบบอนาล็อกมีความสามารถในควบคุมได้เท่ากับการควบคุมแบบดิจิตอลนั้นจะทำได้ยาก เนื่องจากวงจรควบคุมแบบอนาล็อกมีความซับซ้อนสูง

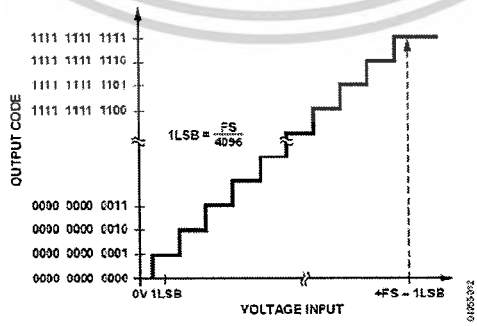
อย่างไรก็ตาม สัญญาณดิจิตอลก็ไม่ได้สามารถทดแทนความละเอียดของสัญญาณอนาล็อกได้อย่างสมบูรณ์แต่จะทำให้การควบคุมนั้นทำได้ง่าย และสะดวกยิ่งขึ้น

การแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล มีประโยชน์มากในการควบคุมอุปกรณ์สวิตชิ่งซึ่งมีลักษณะการแปลงสัญญาณได้หลายวิธี ซึ่งแต่ละวิธีจะมีอัลกอริทึมที่มีความรวดเร็วในการทำงาน และการใช้อุปกรณ์ฮาร์ดแวร์ต่างกันด้วยทำให้ขนาดและราคาต่างกันจึงขึ้นกับอยู่ความต้องการของผู้ใช้ที่จะต้องเลือกให้เหมาะสมกับงานและงบประมาณที่มีอยู่

สำหรับฟังก์ชั่น ADC ของไมโครคอนโทรลเลอร์ ADuC7024 นั้นจะมีขนาด 12-bit 10 channel สามารถรับสัญญาณอนาล็อกได้ตั้งแต่ 0 – VREF เมื่อทำงานอยู่ในโหมดของ single-ended และถ้าทำงานอยู่ในโหมด Fully Difference จะสามารถรับสัญญาณอินพุทแบบอนาล็อกได้ตั้งแต่ค่า 0 –AVDD ด้วยค่าแอมปลิจูดสูงสุดเท่ากับ 2VREF



ภาพที่ 2.2 แสดงรูปสัญญาณเมื่อทำงานอยู่ในโหมด Fully Difference

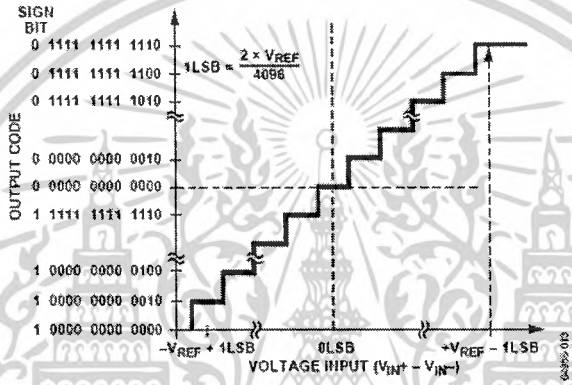


ภาพที่ 2.3 Pseudo Differential and Single-Ended Modes

เมื่อไมโครคอนโทรลเลอร์ของเรานั้นได้ทำงานอยู่ในโหมดของ Pseudo Differential and Single-Ended ย่านในการทำงานที่สามารถรับอินพุทจากภายนอกจะอยู่ในช่วง 0 V ถึง VREF ถ้าจะแปลงให้เป็นรหัสเลขฐานสองทำได้โดย

1. 1 LSB = FS/4096 หรือ
2. 2.5 V/4096 = 0.61 mV หรือ
3. 610 μV when VREF = 2.5 V

ซึ่งคุณสมบัติการแปลงสัญญาณในทางอุดมคติแสดงได้ดังภาพที่ 2.3 ส่วนในภาพที่ 2.4 เป็นคุณสมบัติ ของการแปลงสัญญาณ ADC เมื่อทำงานอยู่ในโหมด Differential



ภาพที่ 2.4 การแปลงสัญญาณ ADC เมื่อทำงานอยู่ในโหมด Differential

ตารางที่ 2.10 กำหนดค่าคุณสมบัติให้กับ ADC

Name	Address	Default Value	Access
ADCCON	0xFFFF0500	0x0600	R/W

สำหรับการเขียนโปรแกรมเพื่อให้สามารถรับสัญญาณอะนาล็อกได้นั้นจำเป็นที่จะต้องอ่านคู่มือของ MCU เบอร์นั้นๆ เพราะเนื่องจากว่าMCUแต่ละตัวก็จะมีลักษณะการเข้าถึงที่ต่างกันเพราะฉะนั้นการอ่านคู่มือของ MCU จึงมีความจำเป็นมาก

ตารางที่ 2.10 จะเป็นตารางสำหรับกำหนดค่าเริ่มต้นการทำงานให้กับ ADC เช่น ชื่อตัวแปร แอดเดรส รวมไปถึงการกำหนดโหมดการต่อใช้งานของ ADC ด้วยซึ่งสามารถที่จะเลือกรูปแบบการแปลงสัญญาณได้ตามตารางที่ 2.11

ตารางที่ 2.11 ADCCON MMR Bit Designations

Bit	Value	Description
15:13		Reserved
12:10		ADC clock speed.
	000	fADC/1.This divider is provided to obtain 1MSPS ADC with an external clock <41.78MHz
	001	fADC/2(default value)
	010	fADC/4
	011	fADC/8
	100	fADC/16
	101	fADC/32
9:08		ADC Acquisition time.
	00	2clocks
	01	4clocks
	10	8clocks
	11	16clocks
7		Enable start conversion Ser by the user to start any type of conversion command. Cleared by the user to disable a start conversion (cleaning this bit does not stop the ADC when continuously converting:
6		Enable ADCbusy Set by the user to enable the ADCbusy pin Cleared by the user to disable the ADCbusy pin.
5		ADC power control. Set by the user to place the ADC in normal mode (the ADC must be powered up for at least 5 us before it conerrts.correctly) Cleared by the user to place the ADC in power-down mode.

ตารางที่ 2.11 (ต่อ) ADCCON MMR Bit Designations

Bit	Value	Description
4:03		Conversion type
	00	Single-ended mode.
	10	pseudo differential mode
	11	Reserved
2:00		Conversion type
	000	Enable CONVstart pin as a conversion input.
	001	Enable timer1 as a conversion input.
	010	Enable timer0 as a conversion input.
	011	Single software conversion; sets to 000 after conversion (Bit 7 of ADCCON MMR should be cleared after starting a single software conversion to avoid further conversions triggered by the CONVstart pin)
	100	Continuous software conversion
	101	PLA conversion
	Other	Reserved

ตารางที่ 2.12 ADCCP Register

Name	Address	Default Value	Access
ADCCP	0xFFFF0504	0x00	R/W

ADCCP ย่อมาจาก ADC Positive Channel เป็นการเลือก Channel สำหรับการแปลงค่าสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอลที่เป็นสัญญาณแบบบวกซึ่งตารางสำหรับการเลือกดูได้จากตารางที่ 2.13

ตารางที่ 2.13 ADCCP MMR Bit Designation

Bit	Value	Description
7:05		Reserved
4:00		Positive channel selection bits
	00000	ADC0
	00001	ADC1
	00010	ADC2
	00011	ADC3
	00100	ADC4
	00101	ADC5
	00110	ADC6
	00111	ADC7
	01000	ADC8
	01001	ADC9
	01010	ADC10
	01011	ADC11
	01100	DAD0/ADC12
	01101	DAD1/ADC13
	01110	DAD2/ADC14
	01111	DAD3/ADC15
	10000	Temperature sensor
	10001	AGND (self-diagnostic feature)
	10010	Internal reference(self-diagnostic feature)
	10011	Avlo/2
	Others	reserved

ตารางที่ 2.14 ADCCN Register

Name	Address	Default Value	Access
ADCCN	0xFFFF0508	0x01	R/W

ADCCN ย่อมาจาก ADC Negative Channel เป็นการเลือก Channel สำหรับการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลที่เป็นสัญญาณแบบลบ ซึ่งสำหรับการเลือกดูได้จากตาราง

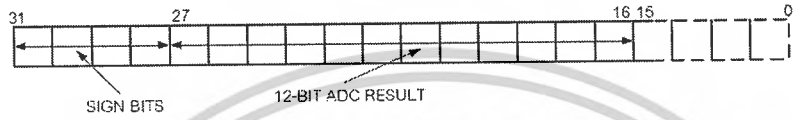
ตารางที่ 2.15 ADCCN MMR Bit Designation

Bit	Value	Description
7:05		Reserved
4:00		Negative channel selection bits
	00000	ADC0
	00001	ADC1
	00010	ADC2
	00011	ADC3
	00100	ADC4
	00101	ADC5
	00110	ADC6
	00111	ADC7
	01000	ADC8
	01001	ADC9
	01010	ADC10
	01011	ADC11
	01100	DAD0/ADC12
	01101	DAD1/ADC13
	01110	DAD2/ADC14
	01111	DAD3/ADC15
	10000	Internal reference(self-diagnostic feature)
	Others	Reserved

ตารางที่ 2.16 ADCDAT Register

Name	Address	Default Value	Access
ADCDAT	0xFFFF0510	0x00000000	R

ADCDAT คือ ADC Data result registerซึ่งมีขนาด 12 bit แสดงได้ดังภาพด้านล่างโดยที่ 4 บิตบน (27-31) ส่วนรูปแบบผลลัพธ์ของข้อมูลขนาด 12 บิต นั้นจะเริ่มตั้งแต่บิตที่ 16-27



ภาพที่ 2.5 การแสดงผลลัพท์ข้อมูล

ตารางที่ 2.17 ADCRST Register

Name	Address	Default Value	Access
ADCSTA	0xFFFF050C	0x00	R

ADCRST เป็นการรีเซทค่าของข้อมูลในรีจิสเตอร์และทำการเขียนข้อมูลค่าเริ่มต้นลงไปยังรีจิสเตอร์

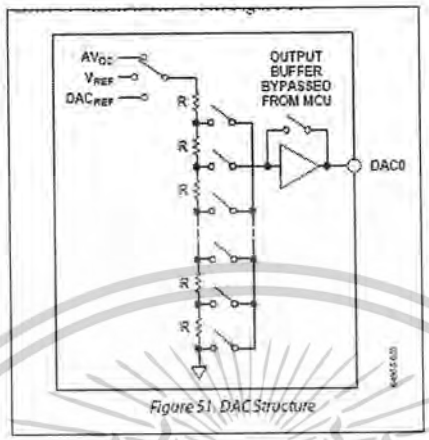
ตารางที่ 2.18 ADCGN Register

Name	Address	Default Value	Access
ADCGN	0xFFFF0530	0x0200	R/W

ADCGN ใช้สำหรับสอบเทียบ (Calibration) รีจิสเตอร์

2.1.3 การติดต่อและการเขียนโปรแกรมสั่งงานของวงจรแปลงดิจิตอลเป็นอนาลอก (D/A)

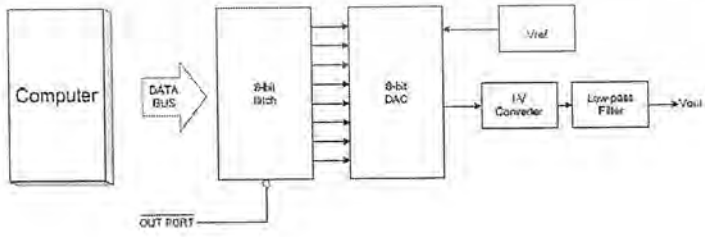
มี D/A จำนวน 2 ช่อง อินพุตเป็นสัญญาณอนาลอกขนาด 12 บิต ขนาดของแรงดันเอาต์พุตอยู่ที่ $0 - V_{REF}$, $0 - A_{VDD}$, $0 - DAC_{REF}$แล้วแต่จะเลือกใช้



ภาพที่ 2.6 วงจรแปลงดิจิตอลเป็นอนาลอก (D/A)

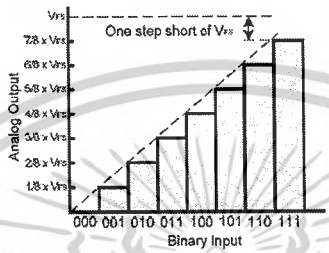
หลักการทำงานเมื่อมีลอจิก 1 เข้ามาเหมือนกับการสับสวิทช์ทำให้เกิดการแบ่งแรงดันส่งต่อผ่านออปแอมป์ ทำให้เกิดแรงดันที่เอาต์พุตลอจิกที่มีนัยสำคัญมากกว่าถูกสับอยู่ก็จะทำให้ได้รับค่าสัญญาณเอาต์พุตมากขึ้นตามไปด้วย

สัญญาณควบคุมอุปกรณ์ โดยทั่วๆ ไปเช่น มอเตอร์ นั้นจะเป็นสัญญาแบบอะนาล็อกที่ใช้กับค่า (แบบต่อเนื่อง) เช่น แรงดันไฟฟ้า, กระแสไฟฟ้า เป็นต้น ส่วนสัญญาณที่ได้จากคอมพิวเตอร์หรือ ไมโครคอนโทรลเลอร์ซึ่งจะเป็นสัญญาณแบบดิจิตอล (แบบไม่ต่อเนื่อง) เมื่อเราได้นำระบบดิจิตอล หรือใช้ไมโครคอนโทรลเลอร์มาใช้ควบคุมอุปกรณ์แบบอะนาล็อกเหล่านี้ จึงต้องมีวงจรในการแปลงสัญญาณทางดิจิตอลเป็นสัญญาณแบบอะนาล็อก ตั้งแต่ศูนย์โวลต์จนระดับสูงสุดที่กำหนดไว้ เรียกว่าวงจร Digital to Analog Converter (DAC)



ภาพที่ 2.7 ระบบการแปลงสัญญาณดิจิตอลเป็นอะนาล็อก

ในภาพที่ 2.7 เป็นการแสดงถึงส่วนประกอบหลักของระบบ DAC โดยทั่วไปคอมพิวเตอร์จะมีเอาต์พุตเป็นค่าไบนารีซึ่ง วงจรแลทช์จะรับค่าไบนารีเข้ามาเพื่อส่งไปยัง DAC ในวงจรนี้จะใช้แหล่งกำเนิดแรงดัน หรือกระแสคงที่เพื่ออ้างอิงในการแปลงข้อมูลไบนารีเป็นระดับกระแสต่อมาจะมีวงจรแปลงจากกระแสเป็นระดับแรงดัน (current-to-voltage converter) ซึ่งปกติจะใช้ออปแอมป์ สัญญาณอะนาล็อกที่ได้จะผ่านวงจร low-pass filter เพื่อกำจัดสัญญาณความถี่สูงที่แฝงอยู่ในรูปของสัญญาณที่ถูกสร้างขึ้นมา



ภาพที่ 2.8 Transfer Curve ในอุดมคติของ DAC 3 บิต

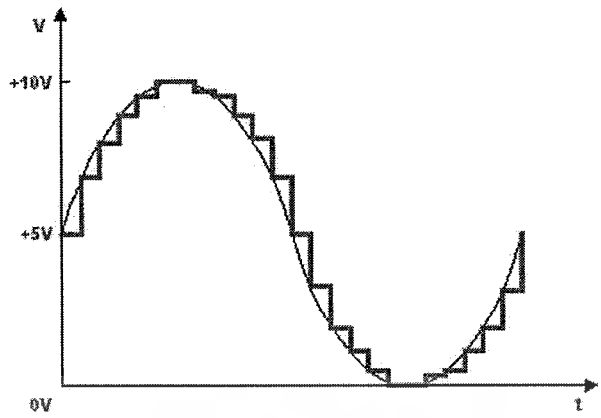
ในภาพที่ 2.8 เป็นกราฟแสดงถึงความสัมพันธ์ระหว่าง เอาต์พุตที่เป็นอะนาล็อกกับอินพุตที่เป็นดิจิตอลขนาด 3 บิตเรียกว่า transfer curve สังเกตว่าเมื่ออินพุตไบนารีเพิ่มขึ้นเอาต์พุตอะนาล็อกจะเพิ่ม ในลักษณะขั้นบันได ขนาดของแต่ละขั้นจะ หาได้จาก

$$\text{Step size} = VFS/2n \qquad (2.1)$$

เมื่อให้ VFS คือ ระดับแรงดันเอาต์พุตสูงสุด

n คือ จำนวนบิตของอินพุต

เนื่องจากเอาต์พุตของ DAC จะเพิ่มเป็นขั้น ๆ รูปคลื่นสัญญาณที่ได้จาก DACจึงมีลักษณะไม่เรียบดังตัวอย่าง ในภาพที่ 2.9 ซึ่งแสดงถึงสัญญาณซายน์ ที่สร้างจาก DAC



ภาพที่ 2.9 คลื่นซายน์ที่สร้างจาก DAC

ถ้าทำการเพิ่มจำนวนบิตของความละเอียดของ DAC จะเพิ่มขึ้นเช่น เมื่อเราใช้ DAC 12 บิต และ V_{FS} = 5.0 V ความละเอียดคือ 5.0 V / 4096 = 1.22 mV ซึ่งจะ ละเอียดกว่า DAC 8 บิตถึง 16 เท่า

ความถูกต้องของค่า DAC ขึ้นอยู่กับองค์ประกอบต่างๆ หลายส่วน ดังต่อไปนี้

1. Quantization error DAC บิต VFS = 5.0 V เอาต์พุตจะมีความละเอียด 19.53 mV ถ้าจะต้องการให้เอาต์พุตมี 4.00 V DAC จะให้เอาต์พุตได้ใกล้เคียงที่สุดคือ 4.04 V (19.53 mV x 205) ค่าผิดพลาด 4 mV โดยทั่วไปค่าผิดพลาดจะเท่ากับ +/- 0.5 LSB (least significant bit) ตัวอย่างเช่นถ้า DAC 8 บิต ความผิดพลาดจะเป็น 1 ใน 512 หรือ +/- 0.195 %

2. Offset and gain errors เมื่ออินพุตไบนารีมีค่าเท่ากับ 0 แต่ถ้าเอาต์พุตของ DAC ไม่เป็นค่า 0 เรียกว่า offset error และอาจเกิดร่วมกับ gain error ความผิดพลาดเหล่านี้อาจจะทำให้ค่าของ transfer curve ในภาพที่ 2 .9 จะโค้งขึ้นหรือลงขึ้นอยู่กับความไม่สมดุลย์ภายใน DAC อย่างไรก็ตาม offset error และ gain error จะแก้ได้โดยใช้ความต้านทานปรับค่าได้ต่อไว้ภายนอก

3. Nonlinearity คือค่าความคลาดเคลื่อนสูงสุดของ transfer curve เทียบกับเส้นตรงจากจุดศูนย์และจุดสูงสุด ซึ่งมันจะขึ้นอยู่กับความผิดพลาดของส่วนประกอบภายใน DACใน data sheet ของ DAC จะระบุเป็นเปอร์เซ็นต์เทียบกับค่าสูงสุด หรือในการที่จะระบุเป็นเศษส่วนของค่า LSB (โดยทั่วไปคือ +/- 0.5 LSB)

4. Settling time คือช่วงเวลานับแต่ให้อินพุตจนกระทั่ง DAC ให้ เอาต์พุต วัดเมื่อเอาต์พุตที่ได้ผิดพลาดจากค่าจริงน้อยกว่า 0.5 LSB ค่าเวลานี้อาจน้อยกว่า 100 ns สำหรับ DAC มีความเร็วสูงและอาจมากกว่า 100us สำหรับ DAC ราคาถูก

สำหรับการใช้งาน DAC ในไมโครคอนโทรลเลอร์ ARM7 เบอร์ ADuC7024 นั้นจะมีขนาด 12-bit 2 Channel มีย่านการใช้งาน 2 ย่าน คือ 0 V - VREF และ 0 V – AVDD ตัวอย่างการกำหนดรีจิสเตอร์ DAC0CON (ดูจากตารางที่2.19) และ DAC0DAT (ดูจากตารางที่ 2.4) ส่วน ตารางที่ 2.19 และ 2.20 นั้นจะเป็นตารางสำหรับบอกถึงรีจิสเตอร์ DAC x CONและ DAC x DAT ทั้งหมดที่มีใช้และบอกแอด เดรสของรีจิสเตอร์แต่ละตัว

ตารางที่ 2.19 DACxCON Registers

Name	Address	Default Value	Access
DAC0CON	0xFFFF0600	0x00	R/W
DAC1CON	0xFFFF0608	0x00	R/W
DAC2CON	0xFFFF0610	0x00	R/W
DAC3CON	0xFFFF0618	0x00	R/W

ตารางที่ 2.20 DAC0CON MMR Bit Designations

Bit	Value	Name	Description
6			Reserved
5		DACCLK	DAC Update Rate. Set by user to update the DAC using Timer1.Cleared by user to update the DAC using HCLK(core clock).
4		DACCLR	DAC Clear Bit. Set by user to enable normal DAC operation. Cleared by user to reset data register of the DAC to 0.
3			Reserved. This bit should be left at 0.
2			Reserved. This bit should be left at 0.
1:0			
	00		Power-Down Mode.The DAC output is in tristate
	01		0V to DACref Range.
	10		0V to Vref(2.5V)Range.
	11		0V to Avdd Range.

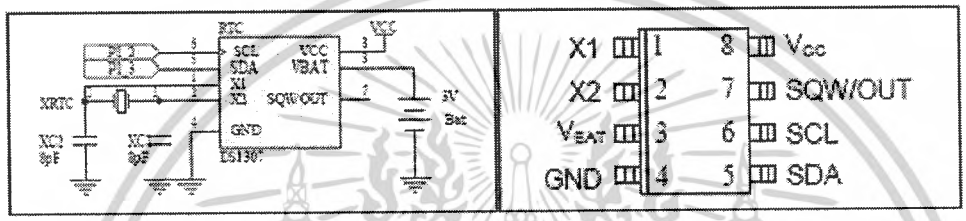
ตารางที่ 2.21 DACxDAT Registers

Name	Address	Default Value	Access
DAC0DAT	0xFFFF0604	0x00000000	R/W
DAC1DAT	0xFFFF060C	0x00000000	R/W
DAC2DAT	0xFFFF0614	0x00000000	R/W
DAC3DAT	0xFFFF061C	0x00000000	R/W

ตารางที่ 2.22 DAC0DAT MMR Bit Designations

Bit	DESCRIPTION
31:24	Reserved.
27:16	12-bit data for DAC0
15:0	Reserved

2.1.4 เขียนโปรแกรมติดต่อกับ REAL TIME CLOCK

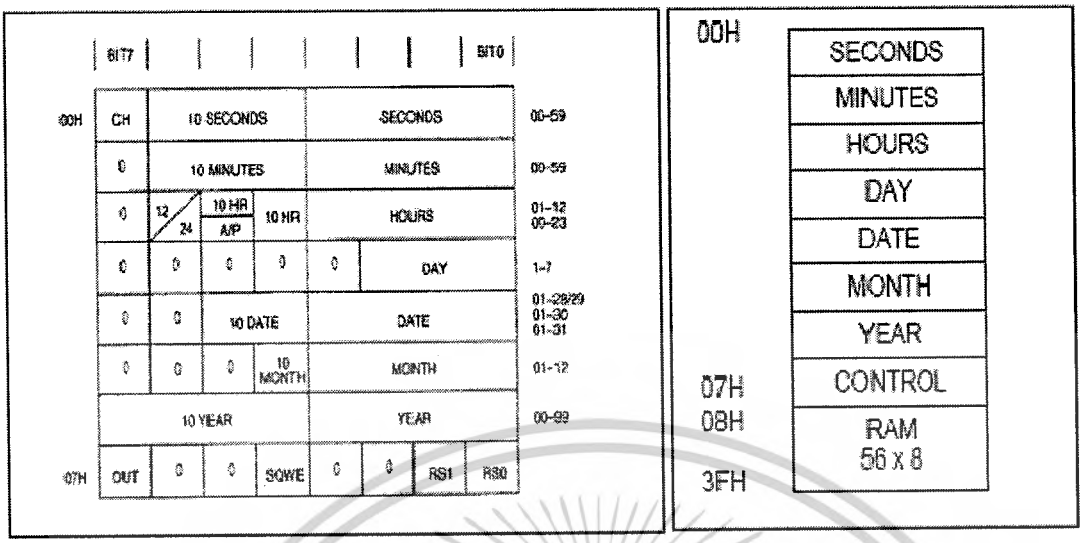


ภาพที่ 2.10 ลักษณะวงจรการต่อสายและรายละเอียดขาสัญญาณต่างๆ

หน้าที่การทำงานของแต่ละขา

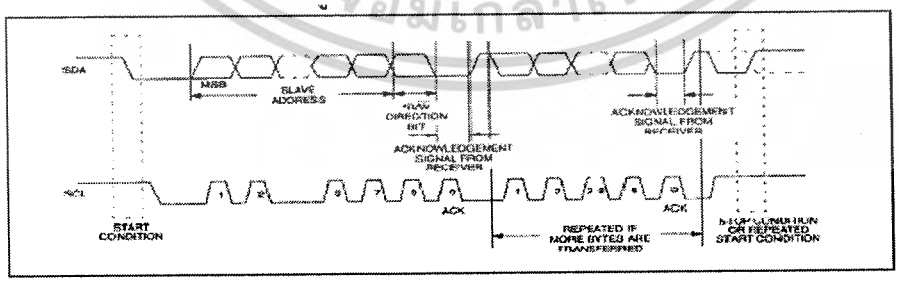
1. Vcc - แหล่งจ่ายไฟฟ้า
2. X1,X2 - ขาต่อคริสตอลความถี่ 32.768 kHz
3. Vbat - +3V แหล่งจ่ายไฟสำรอง
4. GND - กราวด์
5. SDA - Serial data line ขาสัญญาณข้อมูลอนุกรม
6. SCL - Serial clock line ขาสัญญาณนาฬิกา
7. SQW/OUT – ขาสัญญาณรูปคลื่นสี่เหลี่ยมเอาต์พุท

ตารางที่ 2.23 แสดงแอดเดรสของ Register ต่างๆ ใน DS1307



DS1307 มีขนาด 64 * 8 ในแต่ละแอดเดรสของข้อมูลจะมีขนาด 8 บิต เริ่มตั้งแต่แอดเดรสที่ 00H-06H มีไว้สำหรับเขียนหรือ การอ่านข้อมูลที่เป็น วินาที, นาที, ชั่วโมง, วัน, วันที่, เดือน, ปี ไปตามลำดับ แต่ในส่วนการใช้งานนั้นถ้าต้องการเขียนปี 07 ลงไปใน register จะต้องอ้างอิงแอดเดรสก่อนคือ ให้แอดเดรสที่ 06H ต่อจากนั้นก็เขียนข้อมูลลงไปได้เลย คือ 07H ซึ่งตัวอย่างของคำสั่งนี้คือ WriteDS1307(0x06,0x07); ลำดับการทำงานของคำสั่งคือจะต้องส่ง 06H ออกไปก่อนต่อจากนั้นเราก็จะทำการส่งข้อมูล 07H ตามออกไปแอดเดรสที่ 07H นั้นซึ่งเป็นส่วนของการควบคุมการผลิตของสัญญาณพัลซ์ ซึ่งเราสามารถที่จะต่อออกไปใช้งานภายนอกได้ส่วนแอดเดรสตั้งแต่ 08H-3FH นั้นเป็นส่วนของแรมสามารถที่จะเก็บข้อมูลได้ขนาด 56*8 บิต

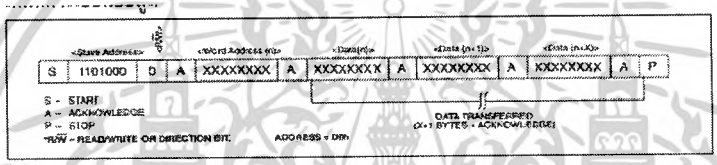
TIMING DIAGRAM ของการส่งถ่ายข้อมูลแบบ *I 2C*



ภาพที่ 2.11 Timing Diagram การส่งข้อมูล

จากภาพที่ 2.11 สังเกตได้ว่าถ้า SCL เป็นลอจิก 1 แบบค้างสภาวะแล้ว SDA มีการเปลี่ยนแปลงข้อมูลจะถือว่าการเปลี่ยนแปลงนั้นไม่ใช่ข้อมูล ในช่วงเริ่มต้น (start condition) SCL เป็นลอจิก 1 SDA เป็นช่วงขอบขาลงหลังจากผ่านสัญญาณ start แล้วตัว master จะส่งแอดเดรสไปให้กับตัว slave (DS1307) ว่าจะติดต่อกับตัวไหนในบอร์ดทดลองนี้ DS1307 อยู่ที่ 0xD หลังจากนั้นจะส่งบิตให้ R/W เพื่อบอกว่าต้องการเขียนหรืออ่าน 1=อ่าน, 0=เขียน เช่น 0xD0 เป็นการติดต่อเพื่อทำการเขียนเป็นต้น ซึ่งหลังจากนั้นตัว master จะหยุดส่ง SDA แต่ส่ง SCL ออกไปหนึ่งลูกและรอรับสัญญาณจาก DS1307 ว่าทราบข้อมูลหรือ ยังถ้าตัวลูกส่ง ACK ตอบกลับมาว่าทราบแล้ว master จะส่งข้อ มูลไบต์ถัดไปตามมาถ้าส่งครบแล้วตัว master จะทำให้ SDA,SCLเป็นลอจิก 1 คือช่วงของ stop condition

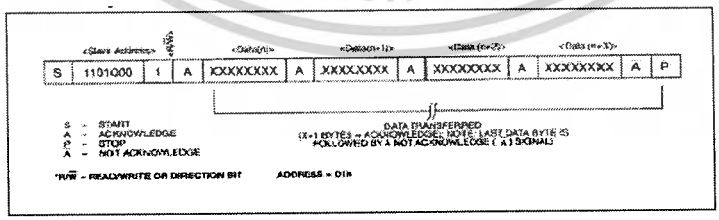
โหมดการเขียนข้อมูล



ภาพที่ 2.12 โหมดการเขียนข้อมูล

ในโหมดนี้ตัวไมโครคอนโทลเลอร์ (Master) ต้องการติดต่อกับ DS1307 (slave) เพื่อทำการเขียนข้อมูลหลังจากผ่าน start condition ไปแล้วตัว master จะส่งข้อมูล 0xD0 เพื่ออ้างอิงถึงตำแหน่งและการเขียนต่อจากนั้นก็ส่ง ACK รอการตอบกลับตามด้วยข้อมูลและ stop bit เป็นการสิ้นสุดโหมดการเขียนข้อมูล

โหมดการอ่านข้อมูล



ภาพที่ 2.13 การอ่านข้อมูล

ในโหมดนี้ตัวไมโครคอนโทลเลอร์ (Master) ต้องการติดต่อกับ DS1307 (slave) เพื่อทำการอ่านข้อมูลหลังจากผ่าน start condition ไป แล้วตัว master จะทำการส่งข้อมูล 0xD1 เพื่อใช้ในการอ้างอิงถึงตำแหน่งและการอ่าน ต่อจากนั้นก็ส่ง ACK รอการตอบกลับตามด้วยข้อมูลและ stop bit เป็นการสิ้นสุดโหมดการเขียนข้อมูล

2.1.5 การอินเตอร์รัปต์ (Interrupt)

การอินเตอร์รัปต์ คือ การขัดจังหวะการทำงานของโปรแกรมกล่าวคือ ถ้าโปรแกรมทำงานอยู่ในภาวะปกติแล้วเกิดมีการร้องขอให้ไปทำงานอื่นก่อน แล้วจึงจะกลับมาทำงานในสภาวะปกติ ลักษณะเช่นนี้เรียกว่า "การอินเตอร์รัปต์" การอินเตอร์รัปต์ของ ADuC7024 นั้นจะมีการใช้งานของอินเตอร์รัปต์จากแหล่งต่าง ๆ ได้ถึง 23 แหล่ง เช่น อินเตอร์รัปต์ ADC และการอินเตอร์รัปต์ UART เป็นต้น นอกจากนี้แล้วยังมีอินเตอร์รัปต์อีก 4 ตัว ที่มาจากภายนอกผ่านขา IRQ0, IRQ1 , IRQ2, และ IRQ3 การอินเตอร์รัปต์ของ ARM7TDMI มีการอินเตอร์รัปต์อยู่ 2 ชนิด คือ การอินเตอร์รัปต์ปกติ (IRQ) การอินเตอร์รัปต์แบบเร็ว (FIQ) ซึ่งในการอินเตอร์รัปต์ทั้งหมดสามารถทำงานแยกกันได้โดยรีจีสเตอร์ต่าง ๆ ของอินเตอร์รัปต์แสดงได้ดังตารางที่ 2.24

ตารางที่ 2.24 IRQ/FIQ MMRs Bit Description

Bit	Description
0	All Interrupts OR'ed
1	SW1
2	Timer0
3	Timer1
4	Wake-Up Timer-Timer2
5	Watchdog Timer-Timer3
6	Flash Control
7	ADC Channel
8	PLL Lock
9	I2C0 Slave
10	I2C0 Master
11	I2C1 Master
12	SPI Slave

ตารางที่ 2.24 (ต่อ) IRQ/FIQ MMRs Bit Description

Bit	Description
13	SPI Master
14	UART
15	External IRQ0
16	Comparator
17	PSM
18	External IRQ1
19	PLA IRQ0
20	PLA IRQ1
21	External IRQ2
22	External IRQ3
23	PWM Trip (IRQ only)/PWM Sync(FIQ only)

ตารางที่ 2.25 IRQSTA Register

Name	Address	Default Value	Access
IRQSTA	0xFFFF0000	0x00000000	R

IRQSTA เป็นรีจีสเตอร์สำหรับอ่านอย่างเดียว ซึ่งจะแสดงสถานะของแฟล็กซ์ ถ้าเซ็ทบิทใด บิทหนึ่งเป็น 1 จะทำให้การอินเตอร์รัปต์เกิดขึ้น

ตารางที่ 2.26 IRQSIG Register

Name	Address	Default Value	Access
IRQSIG	0xFFFF0004	0x00XXX000	R

IRQSIG เป็นรีจีสเตอร์ที่ใช้อ่านค่าอย่างเดียว จะแสดงแฟล็กซ์อินเตอร์รัปต์

ตารางที่ 2.27 IRQEN Register

Name	Address	Default Value	Access
IRQEN	0xFFFF0008	0x00000000	R/W

IRQEN เป็นรีจีสเตอร์ที่อ่านและเขียน ถ้าเซทบิทให้เป็น 1 หมายถึงการ Enables ถ้าเซทให้เป็น 0 หมายถึงการ Disable

ในไมโครคอนโทรลเลอร์ ADuC7024 มี Timer สำหรับให้เลือกใช้งานมี 4 ตัวด้วยกัน คือ

1. Timer0 (RTOS Timer), count-down
2. Timer1 (General Use Timer), count-down i count-up
3. Timer2 (Wake Up / general use timer), count-down i count-up
4. Timer3 (Watchdog timer), count-down, count-up

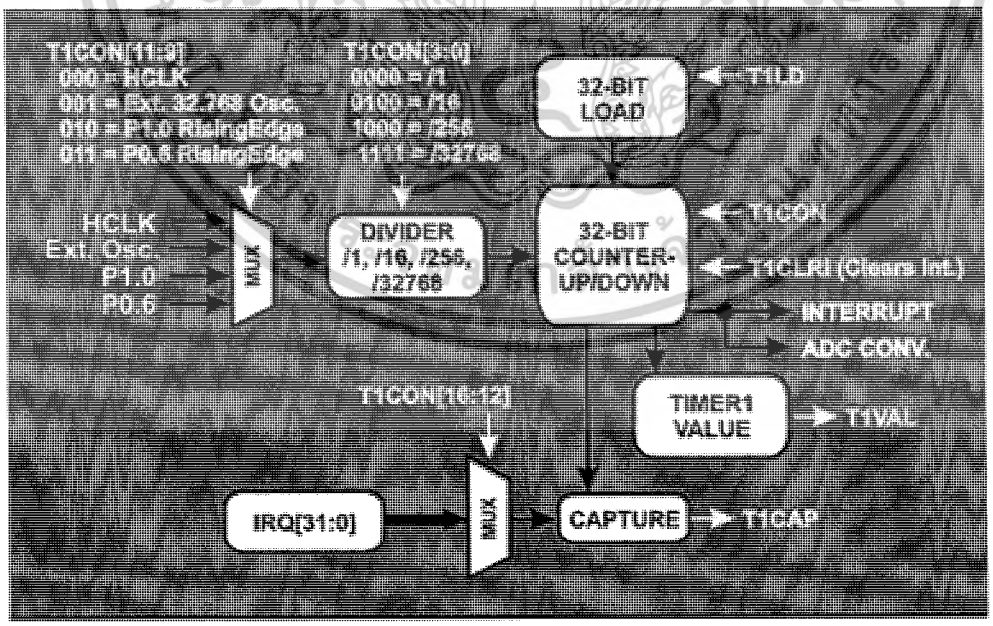
ใน Timers หนึ่งตัวสามารถทำงานได้สองโหมดคือ

1. freerun สำหรับในกรณีนี้ เป็น Timer ซึ่งจะทำการเริ่มนับจากค่าสูงสุดลงมาจนถึงค่าต่ำสุด(timer count down) จากนั้นจะเริ่มนับจากค่าต่ำสุดไปจนถึงค่าสูงสุด (timer count-up)

2. periodic (okresowy) ในกรณีนี้ Timer ซึ่งจะเริ่มนับจากค่าสูงสุดใน Load Register ให้ (TxLD) ลงมาจนถึงค่าต่ำสุด (timer count - down) จากนั้นให้ทำการเริ่มนับจากค่าต่ำสุดใน Load Register (TxLD) ไปจนถึงค่าสูงสุด(timer count-up) สำหรับโหมด Periodic จะทำงานตามสมการ

$$Interval = \frac{(TxLD)\,\mathrm{Pr}escaler}{SourceClock} \qquad (2.2)$$

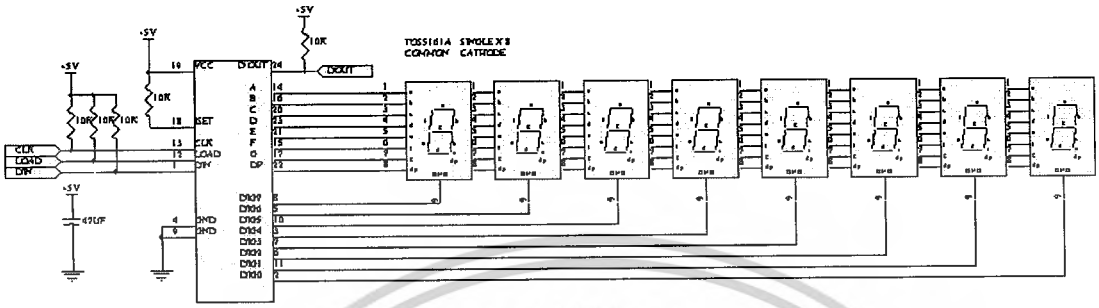
Timer1 (General purpose timer)



ภาพที่ 2.14 ภาพโครงสร้างของ Timer

รีจิสเตอร์ของ T1 คือ: T1LD, T1VAL, T1CON, T1CLRI, T1CAP

2.1.6 การติดต่อและสั่งงาน Display 7-Segment 8 Digit กับไมโครคอนโทรลเลอร์ ARM7



ภาพที่ 2.15 วงจร Display 7-Segment 8 Digit กับไมโครคอนโทรลเลอร์ ARM7

สำหรับการควบคุมการทำงานของอุปกรณ์แสดงผลแบบ 7 ส่วน (7-Segment) นั้นโดยปกติแล้วเราจะใช้สัญญาณที่ได้จากไมโครคอนโทรเลอร์ของเราไปทำให้ 7 Segment ทำงานโดยตรงเลยไม่ได้ เนื่องจาก 7 Segment นั้นต้องการกระแสในการทำให้ติดค่อนข้างสูงด้วยเหตุนี้เองจึงจำเป็นจะต้องมีไอซีอีกอย่างน้อย1ตัวที่จะคอยทำหน้าที่ในเปลี่ยนสัญญาณที่ได้รับจากไมโครคอนโทรลเลอร์และขยายสัญญาณให้สามารถที่จะไปขับ 7 Segment ติดได้ ซึ่งในบอร์ดนี้ใช้ไอซีเบอร์ MAX7219 เป็นบอร์ดที่ใช้ในการทดลองนี้จะสังเกตเห็นได้ว่าจะมีไอซีเบอร์ MAX7219 ที่ทำหน้าที่ควบคุมการทำงานของ 7 Segment ซึ่งลักษณะการต่อวงจรใช้งานของ MAX 7219 กับ 7 Segment แสดงภาพที่2.15

คุณสมบัติของวงจรที่ใช้ IC MAX 7219

1. ทำงานที่ความเร็วสูงถึง 10 เมกะเฮิตซ์

2. สามารถควบคุมการแสดงผลของแผงแสดงผลแบบตัวเลข 7 ส่วนชนิดคอมมอนคาโทดได้ 8 หลัก ต่อวงจรควบคุม 1 วงจร

3. สามารถควบคุมความสว่างของการแสดงผลได้ 16 ระดับ ด้วย Soft-Ware

4. ใช้การเชื่อมต่อแบบอนุกรมโดยใช้สัญญาณในการเชื่อมต่อแค่เพียง 3 เส้นทำให้ประหยัด พอร์ตควบคุมได้เป็นอย่างมาก

5. ใช้การสั่งงานให้แสดงผลเพียงครั้งเดียว หลังจากนั้นจะทำการควบคุมการแสดงผลของแผงแสดงผลทุกหลักเอง จนกว่าเราจะต้องการเปลี่ยนค่าการแสดงผลเป็นค่าใหม่จึงจะส่งข้อมูลและคำสั่งใหม่ให้กับ IC อีก ทำให้ง่ายต่อการเขียนโปรแกรมควบคุมเพราะเราสามารถให้ CPU ไปใช้ทำงานอย่างอื่นได้โดยไม่ต้องกลัวว่าส่วนแสดงผลจะหยุดทำงาน

การทำงาน มีสัญญาณในการเชื่อมต่อเพียง 4 เส้นดังนี้ คือ

1. DIN เป็นขาของสัญญาณอินพุตใช้สำหรับรับข้อมูลอินพุตแบบอนุกรม โดยที่ข้อมูลจะถูกโหลดเข้าไปอยู่ในชิพต์รีจิสเตอร์ ขนาด 16 บิต โดยสัมพันธ์กับสัญญาณนาฬิกา ที่ส่งเข้ามาทางขาCLK โดยข้อมูลจะเริ่มจาก D15..D0

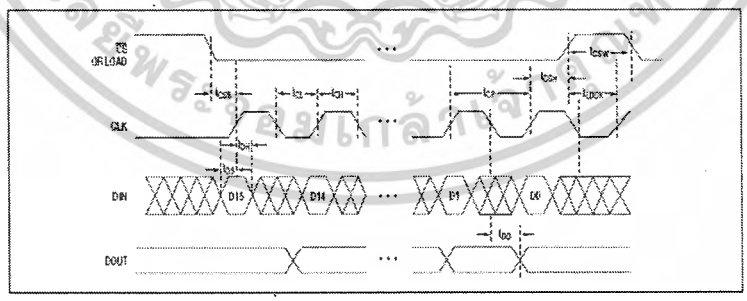
2. CLK เป็นขาสัญญาณอินพุตใช้สำหรับรับสัญญาณนาฬิกาอินพุตจากภายนอก ซึ่งจะมีอัตราความเร็วสูงสุด 10 เมกะเฮิตซ์ โดยเมื่อสัญญาณนาฬิกาเป็นช่วงขอบขาขึ้นสัญญาณ ข้อมูลจาก DIN จะถูกเลื่อนเข้าสู่ชิพต์รีจิสเตอร์ภายในเมื่อสัญญาณนาฬิกาเป็นช่วงขอบขาลงข้อมูลที่อยู่ในชิพต์รีจิสเตอร์ ตำแหน่งสุดท้าย(D15) จะถูกเลื่อนออกมาที่ขาสัญญาณ DOUT

3. LOAD เป็นขาสัญญาณอินพุต ใช้สำหรับควบคุมการโหลดข้อมูลที่ถูกเลื่อนเข้ามารอไว้ในชิพ รีจิสเตอร์เข้าไปยังหน่วยความจำของ IC MAX 7219 โดยค่าในชิพต์รีจิสเตอร์ ทั้งหมดจะถูกโหลดเมื่อสัญญาณนาฬิกาที่ขา LOAD เป็นขอบขาขึ้น

4. DOUT เป็นขาสัญญาณเอาต์พุต ซึ่งเป็นข้อมูลแบบอนุกรมชุดเดียวกันกับสัญญาณ DIN โดยสัญญาณ DOUT นี้เราใช้สำหรับต่อเข้ากับสัญญาณ DIN ของ IC MAX ตัวถัดไป เมื่อต้องการเพิ่งจำนวนหลักแสดงผลให้มากขึ้น

จากข้อมูลข้างต้น ใช้สัญญาณเชื่อมต่อกับไมโครคอนโทรลเลอร์เพียง 3 เส้น จึงได้เลือกใช้พอร์ต 1 คือ P1.4, P1.6 และ P1.7

การเขียนโปรแกรมติดต่อกับ MAX 7219 จำเป็นที่จะต้องศึกษาถึงคู่มือการใช้งาน MAX 7219 โดยละเอียดจึงจะสามารถเขียนโปรแกรมติดต่อกับ MAX 7219 ได้ถูกต้อง ซึ่งในใบงานนี้จะกล่าวถึงเนื้อหาที่จำเป็นในคู่มือของMAX 7219 ที่ถูกนำมาใช้เป็นตัวช่วยในการเขียนโปรแกรมรวมไปถึงตัวอย่างโปรแกรมที่ใช้ในการเขียนติดต่อกับMAX 7219 เพื่อควบคุม 7 Segment



ภาพที่ 2.16 Timing Diagram ของ MAX 7219

สำหรับการเขียนโปรแกรมควบคุมการทำงาน 7 Segment โดยใช้ MAX 7219 นั้นจำเป็นจะต้องทำความเข้าใจวงจรในภาพที่ 2.15 เสียก่อนว่าพอร์ตต่างๆ ของไมโครคอนโทรลเลอร์ ARM7 ADuC7024 ต่ออยู่กับขาใดของMAX 7219 ซึ่งจากวงจรจะเห็นได้ว่า P1.4, P1.5 และ P1.6 นั้นต่ออยู่

กับขาของสัญญาณ CLK, Load และ DIN ตามลำดับส่วน Timing Diagram ของการทำงานของ MAX7219 แสดงดังในภาพที่2.16

รูปแบบการส่งข้อมูลแบบอนุกรม ซึ่งการส่งข้อมูลจะส่งเป็น Packets ครั้งละ 16 บิตโดยในหนึ่ง packets จะประกอบไปด้วย ข้อมูล 8 บิต (D0-D7) แอดเดรส 4 บิต (D8-D11) ส่วนบิตที่ 12-15 (D12-D15) นั้นไม่สนใจซึ่งรูปแบบการส่งข้อมูล 1 packets แสดงดังตารางที่ 2.28

ตารางที่ 2.28 รูปแบบการส่งข้อมูลแบบอนุกรม (16 บิต)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
x	x	x	x	ADDRESS				MSB			DATA				LSB

หลังจากที่เราศึกษารูปแบบการส่งข้อมูลแบบอนุกรม แล้วต่อไปจะเป็นการศึกษาวิธีการ Map Address ของรีจิสเตอร์ ในตารางที่ 2.28 ซึ่งจะอธิบายวิธี Map Address ของ MAX 7219จะพิจารณาที่บิต D8-D11 ซึ่งเป็นบิตที่ใช้สำหรับกำหนด Address ของ 7 Segment ส่วนบิตที่ D0-D7 จะเป็นบิตข้อมูล และ D12-D15 จะเป็นบิตที่ไม่สนใจเพื่อความเข้าใจเราจะกำหนดให้เป็นศูนย์ได้ ในตารางที่ 2.28 จะอธิบายถึงรหัสเลขฐาน 16 ที่จะนำไปเขียนในโปรแกรมเพื่อกำหนดว่าจะให้ข้อมูลออกที่ 7 Segment หลักใด (0-7) ยกตัวอย่างเช่น ต้องการส่งข้อมูลไปแสดงยังหลักที่ 0 รหัสคือ 0x01, หลักที่ 1 รหัสคือ 0x02 นอกจากนี้แล้วยังมีรหัสอื่นๆ ที่เกี่ยวข้อง เช่น กำหนดให้อยู่ในโหมด Decode รหัสคือ 0x09, การปรับความเข้มของ 7 Segment รหัสคือ 0x0A เป็นต้น รายละเอียดอื่นๆ สามารถดูได้จากตารางที่ 2.29

ตารางที่ 2.29 Register Address Map

REGISTER	**ADDRESS**					**HEX**
	D15-D12	**D11**	**D10**	**D9**	**D8**	**CODE**
No-Op	x	0	0	0	0	0xX0
Digit0	x	0	0	0	1	0xX1
Digit1	x	0	0	1	0	0xX2
Digit2	x	0	0	1	1	0xX3
Digit3	x	0	1	0	0	0xX4
Digit4	x	0	1	0	1	0xX5

ตารางที่ 2.29 (ต่อ) Register Address Map

Digit5	x	0	1	1	0	0xX6
Digit6	x	0	1	1	1	0xX7
Digit7	x	1	0	0	0	0xX8
Decode Mode	x	1	0	0	1	0xX9
Intensity	x	1	0	1	0	0xXA
Scan Limit	x	1	0	1	1	0xXB
Shutdown	x	1	1	0	0	0xXC
Display Test	x	1	1	1	1	0xXF

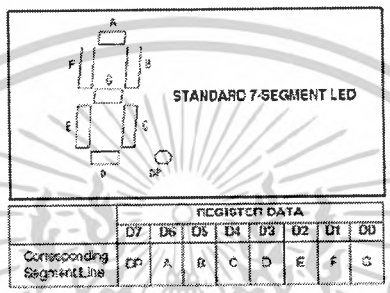
ตารางที่ 2.30 Decode-Mode (Address(HEX)=x0X9)

DECODE MODE	**REGISTER DATA**								**Hex**
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Code**
No Decode digits7-0	0	0	0	0	0	0	0	0	**0x00**
Code B decode for digit0 No decode digits7-1	0	0	0	0	0	0	0	1	**0x01**
Code B decode for digit 3-0 No decode digits 7-4	0	0	0	0	0	1	1	1	**0x0F**
Code B decode for digit 7-0	1	1	1	1	1	1	1	1	**0xFF**

ในตารางที่ 2.30 จะเป็นรายละเอียดการกำหนด Register Data เมื่อ MAX 7219 ทำงานอยู่ในโหมด Decode (Address(HEX) =x0X9) ซึ่งการกำหนดโหมดการทำงานที่ถูกกำหนดโดยรหัส Address(HEX)=x0X9) แล้วเรายังสามารถกำหนดได้ว่าจะเลือกให้ 7 Segment หลักใดทำงานอยู่ในโหมด Decode ยกตัวอย่าง เช่น ในตารางที่ 2.30 เมื่อคำสั่ง Register Address Map ในตารางที่ 2.29 ถูกกำหนดให้เป็น (0x09) นั้นแสดงว่า MAX 7219 จะทำงานอยู่ในโหมด Decode ซึ่งการ Decode นั้นเรายังสามารถที่จะกำหนดได้อีกว่าจะเลือก Decode ที่หลักใดบ้างใน ตารางที่ 2.30 จะแสดงตัวอย่างวิธีการ Decode อยู่ 4 ตัวอย่างคือถ้ากำหนดคำสั่งเป็น (Address(HEX)=x009, 0x00) แสดงว่าหลักที่ 7-0 ของ 7 Segment ไม่ได้ทำงานอยู่ในโหมด Decode กำหนดคำสั่งเป็น (Address (HEX)= x009,0x01) แสดงว่าหลักที่ 0 ของ 7 Segment จะทำงานอยู่ในโหมด Decode ส่วนหลักที่ 7-1 ทำ

งานในโหมดปกติกำหนดคำสั่งเป็น (Address (HEX) = x009 , 0x0F) ซึ่งแสดงได้ว่าหลักที่ 3-0 ของ 7 Segment จะทำงานอยู่ในโหมด Decode ส่วนหลักที่ 7- 4 ทำงานในโหมดปกติกำหนดคำสั่งเป็น (Address(HEX)=x009, 0xFF) แสดงว่าทุกหลักของ 7 Segment จะทำงานอยู่ในโหมดDecode

ข้อดีของการสั่งให้โปรแกรมทำงานอยู่ในโหมด Decode คือ ทำให้เราเข้าใจคำสั่งได้ง่าย และสะดวกในการเขียนโปรแกรม ยกตัวอย่างเช่น ถ้าเราต้องการส่งค่าตัวเลข 3 ออกไปยังหลักที่ 1 ของ 7 Segment ตัวอย่างรูปแบบของคำสั่งคือ MAX7219(0x01, 0x03) นั่น หมายความว่าถ้าเราต้องการให้ตัวเลขตัวใดแสดงผลที่ 7 Segment ก็ให้ใส่ข้อมูลตัวเลขนั้นตรงๆ ในโค้ดคำสั่งได้เลย



ภาพที่ 2.17 โหมดการทำงานปกติ

ถ้าเรากำหนดให้โปรแกรมเป็น MAX7219(0x09, 0x00) แสดงว่าการทำงานของ MAX7219 อยู่ในโหมดปกติ ในการที่เราจะส่งข้อมูลไปแสดงผลยัง 7 Segment นั้นเราจะส่งข้อมูลออกไปให้เหมือนโหมด Decode ไม่ได้ ในตัวอย่างเช่นถ้าเราจะส่งค่าข้อมูลเลข 1 ไปแสดงผลที่ 7 Segment ถ้า MAX7219 อยู่ในโหมด Decode เราจะใช้คำส่งเป็น MAX7219(0x09, 0x01) ได้เลยแต่ถ้า MAX7219 อยู่ในโหมดปกติค่าของเลข 1 ที่จะส่งออกไปจะต้องคำนวณจากภาพที่ 2.17 หมายถึงว่าถ้าต้องการที่ให้ค่า 7 Segment ซึ่งจะแสดงเลข 1 ตำแหน่งของหลอด LED ที่จะติดคือ ตำแหน่ง B และ C เมื่อเรานำไปเทียบกับภาพที่ 2.17 จะได้ว่าถ้าจะส่งข้อมูลเลข 1 ไปแสดงผลที่ 7 Segment จะต้องส่งคำสั่งให้ MAX7219 (0x09, 0x30) ในตารางที่ 2.30 จะแสดงรายละเอียดการกำหนดค่าของข้อมูลเมื่อเราให้ MAX 7219 ทำงานในโหมด Decode ในรูปรหัสเลขฐานสองและ แสดงให้เห็นว่าหลอดของ LED ที่ตัว 7 Segment ติดสัมพันธ์กับข้อมูลที่ส่งออกไปอย่างไร ตัวอย่างเช่น ถ้าเราจะส่งข้อมูลเลข 0 ให้ออกไป หลอด LED ที่ 7 Segment ก็จะติดที่ตำแหน่ง A, B, C, D, E, และ F ส่วนตำแหน่ง G จะไม่ติดเพื่อที่จะแสดงเป็นค่าตัวเลข และถ้าจะให้ตำแหน่ง DP ของ 7 Segment ติดก็กำหนดให้ D7 = 1

ตารางที่ 2.31 โหมด Decode แบบรหัสเลขฐานสอง

7-SEGMENT	REGISTER DATA						ON SEGMENTS=1							
CHARACTER	D7*	D6-D4	D3	D2	D1	D0	DP*	A	B	C	D	E	F	G
0		x	0	0	0	0		1	1	1	1	1	1	0
1		x	0	0	0	1		0	1	1	0	0	0	0
2		x	0	0	1	0		1	1	0	1	1	0	1
3		x	0	0	1	1		1	1	1	1	0	0	1
4		x	0	1	0	0		0	1	1	0	0	1	1
5		x	0	1	0	1		1	0	1	1	0	1	1
6		x	0	1	1	0		1	0	1	1	1	1	1
7		x	0	1	1	1		1	1	1	0	0	0	0
8		x	1	0	0	0		1	1	1	1	1	1	1
9		x	1	0	0	1		1	1	1	1	0	1	1
-		x	1	0	1	0		0	0	0	0	0	0	1
E		x	1	0	1	1		1	0	0	1	1	1	1
H		x	1	1	0	0		0	1	1	0	1	1	1
L		x	1	1	0	1		0	0	0	1	1	1	0
P		x	1	1	1	0		1	1	0	0	1	1	1
blank		x	1	1	1	1		0	0	0	0	0	0	0

ในตารางที่ 2 .31 จะเป็น รายละเอียดของคำสั่งเพื่อกำหนดความเข้มของ 7 Segment ซึ่งจะแสดงเป็นคำสั่งรหัสเลขฐาน 16 การเขียนคำสั่งสามารถทำได้โดย กำหนด Register Address Map ให้เป็น (Address (HEX)=0x0A) ซึ่งดูได้จากตารางที่ 2.29 จากนั้นในใส่ข้อมูลในตารางที่ 2.31 ในช่อง Hex Code ซึ่งจะเป็นรหัสเลขฐาน 16 ตั้งแต่ 0x00 ไปจนถึง 0x0F ซึ่งจะสัมพันธ์กับช่อง Duty Circle ซึ่งจะมีค่าตั้ง 1/32 ไปจนถึง 31/32 ซึ่งจะบอกถึงความเข้มจากน้อยไปหามากตามลำดับตัวอย่างการเขียนโปรแกรม เช่น MAX7219(0x0A,0x0F) ความหมายของโปรแกรมคือกำหนดให้หลอด LED ของ 7 Segment มีความเข้มสูงสุดนั่น เอง

ตารางที่ 2.32 โหมดการปรับความเข้มของ 7 Segment (Address (HEX) =0xXA)

Duty Cycle		D7	D6	D5	D4	D3	D2	D1	D0	HEX CODE
MAX7219	MAX7221									
1/32 (min on)	1/16 (min on)	x	x	x	x	0	0	0	0	0xX0
3/32	2/16	x	x	x	x	0	0	0	1	0xX1
5/32	3/16	x	x	x	x	0	0	1	0	0xX2
7/32	4/16	x	x	x	x	0	0	1	1	0xX3
9/32	5/16	x	x	x	x	0	1	0	0	0xX4
11/32	6/16	x	x	x	x	0	1	0	1	0xX5
13/32	7/16	x	x	x	x	0	1	1	0	0xX6
15/32	8/16	x	x	x	x	0	1	1	1	0xX7
17/32	9/16	x	x	x	x	1	0	0	0	0xX8
19/32	10/16	x	x	x	x	1	0	0	1	0xX9
21/32	11/16	x	x	x	x	1	0	1	0	0xXA
23/32	12/16	x	x	x	x	1	0	1	1	0xXB
25/32	13/16	x	x	x	x	1	1	0	0	0xXC
27/32	14/16	x	x	x	x	1	1	0	1	0xXD
29/32	15/16	x	x	x	x	1	1	1	0	0xXE
31/32	15/16(maxon)	x	x	x	x	1	1	1	1	0xXF

คุณสมบัติอีกอย่างหนึ่งที่ MAX7219 ทำได้คือการกำหนดการแสดงผลของ 7 Segment โดยสามารถกำหนดให้แสดงได้ตั้งแต่ 1 หลักจนถึง 8 หลัก ซึ่งคำสั่งที่ใช้คือ (Address (HEX) = 0xXB) แล้วตามด้วยรหัสเลขฐาน16 ตั้งแต่ 0x00 ไปจนถึง 0x07 ตัวอย่างเช่น

1. MAX7219 (0x0B,0x00) หมายถึง 7 Segment จะแสดงผลหลักที่หนึ่งหลักเดียว
2. MAX7219(0x0B,0x03) หมายถึง 7 Segment จะแสดงผลหลักที่ 1 ไปจนถึงหลักที่ 4
3. MAX7219(0x0B,0x07) หมายถึง 7 Segment จะแสดงผลทุกหลัก

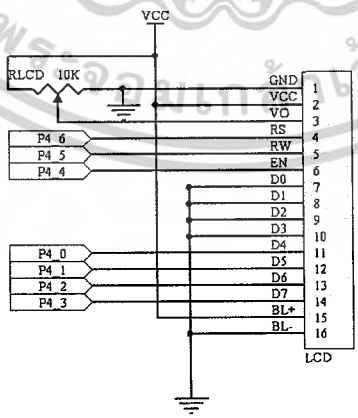
ซึ่งรายละเอียดต่างๆ แสดงดังตารางที่ 2.33

ตารางที่ 2.33 โหมด Scan-Limit (Address (HEX) =0xXB)

SCAN LIMIT	Register Data								HEX
	D7	D6	D5	D4	D3	D2	D1	D0	CODE
Display digits 0 only*	x	x	x	x	x	0	0	0	0xX0
Display digits 0 & 1 *	x	x	x	x	x	0	0	1	0xX1
Display digits 0 1 2*	x	x	x	x	x	0	1	0	0xX2
Display digits 0 1 2 3	x	x	x	x	x	0	1	1	0xX3
Display digits 0 1 2 3 4	x	x	x	x	x	1	0	0	0xX4
Display digits 0 1 2 3 4 5	x	x	x	x	x	1	0	1	0xX5
Display digits 0 1 2 3 4 5 6	x	x	x	x	x	1	1	0	0xX6
Display digits 0 1 2 3 4 5 6 7	x	x	x	x	x	1	1	1	0xX7

2.1.7 คำสั่งควบคุมโมดูลตัวแสดงผลแบบผลึกเหลว (LCD interface)

จอแสดงผลที่ใช้ในใบงานนี้เป็นแบบ DOT MATRIX LCD MODULE เป็นอุปกรณ์แผงแสดงผลตัวอักษรหรือตัวเลข เหมาะสำหรับงานแสดงผลการทำงานเป็นข้อความตัวอักษรหรือข้อความต่างๆ ซึ่ง LCD Module มีส่วนประกอบใหญ่ๆแบ่งได้เป็น Dot Matrix LCD เป็นส่วนของกระจกผลึกสำหรับการแสดงผล Driver เป็นตัวรับสัญญาณจากตัวควบคุมมาขับ LCD อีกที่หนึ่ง Controller เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาและจัดการควบคุม LCD Module ให้ทำงานแสดงผลต่างๆ เช่น ลบจอภาพ การเกิดตัวอักษร โดยตัว LCD MODULE นี้ สามารถต่อเข้ากับบอร์ดของไมโครคอนโทรลเลอร์ ได้ทาง DATA BUS หรือทาง I/O PORT



ภาพที่ 2.18 Schematic ของการต่อจอแสดงผล LCD กับไมโครคอนโทรลเลอร์ ADuC 7024

จากภาพที่ 2.18 Schematic การต่อจอแสดงผล LCD กับไมโครคอนโทรลเลอร์ ADuC 7024 ซึ่งจะเห็นได้ว่าเป็นการต่อแบบ I/O port และการทำงานของจอแสดงผล LCD อยู่ในโหมด 4 บิต ซึ่งจะต่อใช้งานที่ D7 – D4 โดยข้อมูลครั้งแรกที่ส่งนั้นจะถือเป็นข้อมูล 4 บิตบนและข้อมูลที่ส่งต่อมานั้นเป็นข้อมูล 4 บิตล่าง โดยรายละเอียดการต่อใช้งานขาต่างๆของ LCD โมดูล ดังต่อไปนี้

ตารางที่ 2.34 การใช้งานขาต่างๆ

No.	Symbol	Function
1	VSS	0 V Power Supply(GND Level)
2	VCC	Power supply for Logic circuit
3	V0	Power supply for Driving the LCD
4	RS	Data/Instruction select
5	R/W	Read/Write select
6	E	Enable signal
7-14	DB0-DB7	Data Bus line

1. RS (Register Selection) จะเป็นขาเลือก Register ภายใน ซึ่งมีอยู่ 2 ตัวคือ Instruction Register (IR) และ DataRegister (DR) โดยถ้าเป็น 1 จะเป็นการเลือก Data และถ้าใช้เป็น 0 จะเป็นการเลือก Instruction

2. R/W (Read/Write) เป็นตัวเลือกว่าจะเขียนหรืออ่านข้อมูลจาก IC โดยอ่านข้อมูล = 1 และเขียนข้อมูล = 0

3. EN (Enable Signal) เป็นขากำหนดสภาพการรับเขียนอ่านข้อมูล

ตารางที่ 2.35 การรับเขียนอ่านข้อมูล

RS	R/W	EN	Operation
0	0	↓	IR write as internal operation (Display clear, etc.)
0	1	⊓	Read busy flag (DB_7) and Address counter (DB_0-DB_6)
1	0	↓	DR write as internal operation (DR to DD or CG RAM)
1	1	⊓	DR read as internal operation (DD or CG RAM to DR)

4. DB0-DB7 เป็นขารับส่งข้อมูลจากตัว IC

5. VDD ไฟเลี้ยงตัววงจร +5V

6. VSS เป็นขาGND

7. VO เป็นขารับ Voltage ในการขับ LCD .ให้สว่างหรือมืด

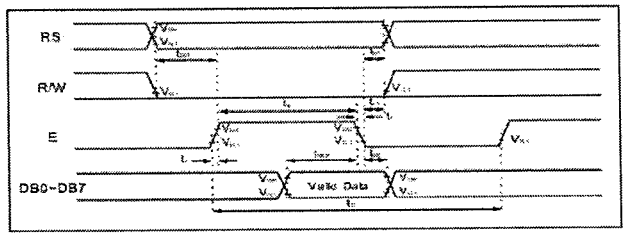
ในการควบคุมให้จอแสดงผล LCD แสดงตัวอักษรจะเริ่มต้นจากการเขียนคำสั่งให้ควบคุมการทำงานเพื่อกำหนดรูปแบบการแสดงผล จากนั้นก็จะส่งค่ารหัสตัวอักษรตามรหัส ASCII ให้กับจอแสดงผล LCD เพื่อนำไปแสดงผลบนหน้าจอ LCD โดยคำสั่งควบคุมแสดง

ตารางที่ 2.36 คำสั่งการควบคุม LCD

Instruction	Instruction Code										Description	Execution time (fosc= 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address to "00H" from AC	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 μs
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	39 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5×11dots/5×8 dots)	39 μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39 μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 μs

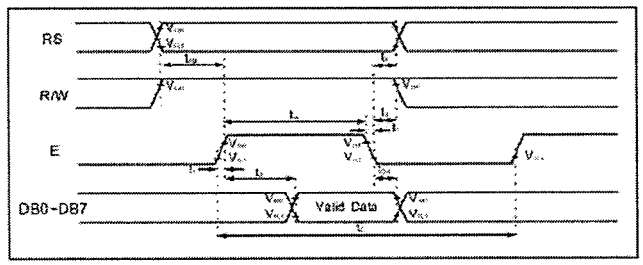
Timing Diagram ของโหมดการเขียน และการอ่าน คำสั่งหรือข้อมูลของ LCD Module อธิบายได้ ภาพที่ 2.19

Write Mode Timing Diagram



ภาพที่ 2.19 Timing Diagram แสดงการเขียนข้อมูล

Read Mode Timing Diagram



ภาพที่ 2.20 Timing Diagram แสดงการอ่านข้อมูล

2.2 การใช้งานโปรแกรม KEIL uVision 3

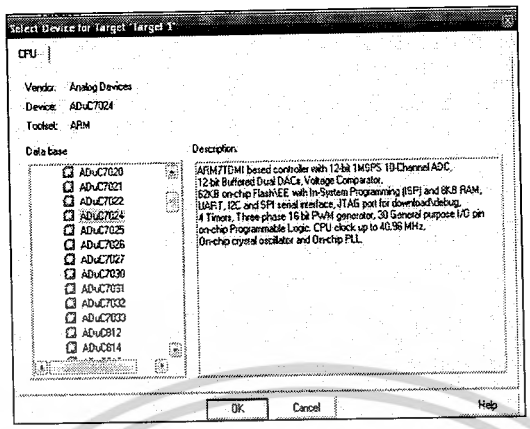
การเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ ด้วยภาษาแอสเซมบลีจะมีการทำงานที่รวดเร็วดี แต่กว่าจะเขียนได้แต่ละโปรแกรมนั้นค่อนข้างยุ่งยาก อีกทั้งยังไม่สามารถเขียนภาษาแอสเซมบลีในระบบใหญ่ๆ ได้อีกด้วย เมื่อศึกษาแล้วมีโปรแกรมที่ช่วยแปลงภาษา C เป็น HEX code ได้เลย โปรแกรมที่ว่านี้คือ KEIL uVision 3 อีกเหตุผลที่เลือก เขียนภาษา C เนื่องจากเป็นภาษาที่เข้าใจได้ง่าย แก้ไขได้สะดวก

ขั้นตอนในการใช้งาน โปรแกรม Keil uVision 3 สรุปพอสังเขปได้ดังนี้

1. เปิดโปรแกรม u Vision3,

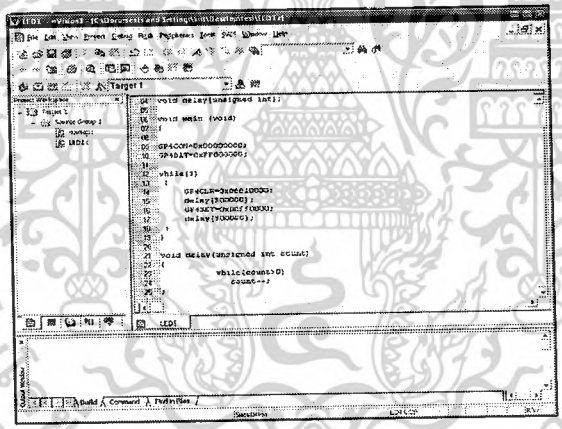
ภาพที่ 2.21 การเปิดโปรแกรม u Vision3

2. สร้างไฟล์โปรเจคและเลือก ARM 7 เบอร์ ADuV 7024



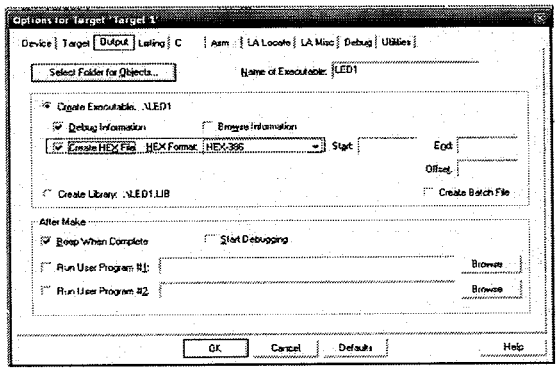
ภาพที่ 2.22 การสร้างไฟล์โปรเจคและเลือก ARM 7 เบอร์ ADuV 7024

3. เขียน source code และเพิ่มเข้าไปใน source file ของโปรเจค



ภาพที่ 2.23 การเขียน source code และเพิ่มเข้าไปใน source file

4. สร้างโปรเจคและ HEX ไฟล์สำหรับ PROM



ภาพที่ 2.24 การสร้างโปรเจคและ HEX ไฟล์สำหรับ PROM

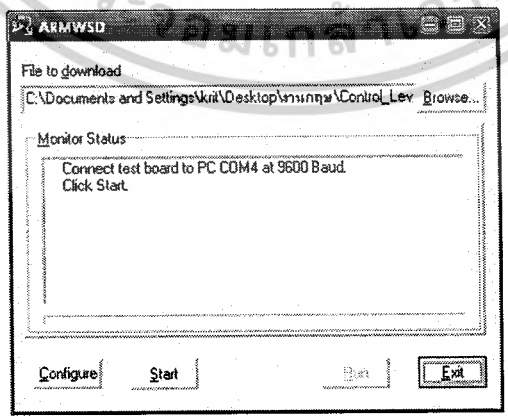
2.3 การ Download Hex file ให้กับ MCU ของบอร์ด

การ Download Hex File ให้กับหน่วยความจำให้กับ Flash ของ MCU ในบอร์ดนั้นจะใช้โปรแกรม ชื่อ "**ARMWSD**" ของ Analog Device ซึ่งจะติดต่อกันกับ MCU ผ่าน Serial Port ของคอมพิวเตอร์ PC ในการเชื่อมต่อกับระบบฮาร์ดแวร์ที่ใช้ในการเชื่อมต่อแบบ RS232 โดยโปรแกรม RMWSD จะใช้สำหรับ Download ข้อมูลให้กับหน่วยความจำของไมโครคอนโทรลเลอร์ ในการที่จะสั่ง Resetให้กับ CPU เข้าทำงานใน Monitor Mode เพื่อสั่ง Download HEX File จาก PC ให้กับบอร์ดสามารถทำได้ ตามขั้นตอนต่อไปนี้คือ

1. ต่อสายสัญญาณ RS232 จาก Com Port ของเครื่องคอมพิวเตอร์ PC เข้ากับขั้วของ RS232 แบบ 4 Pin ของบอร์ด

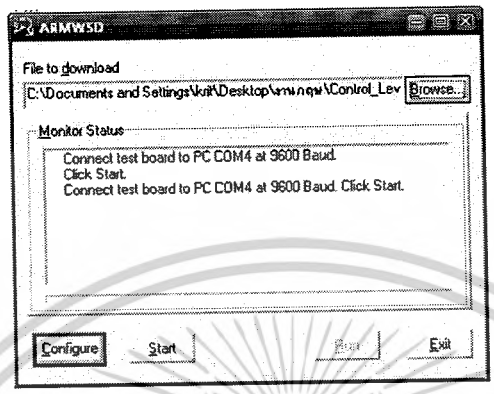
2. จ่ายไฟเลี้ยงวงจรให้บอร์ด ซึ่งจะสังเกตเห็น LED แสดงสถานะของ PWR สีแดงติดสว่างให้เห็น

3. สั่ง Run โปรแกรม "ARMWSD" ซึ่งจะได้ผลดังภาพที่ 2.25

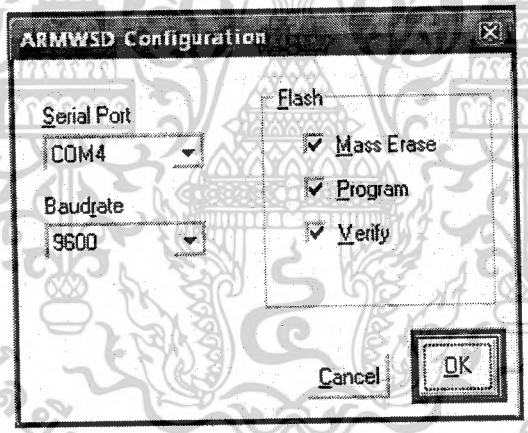


ภาพที่ 2.25 ผลการสั่ง Run โปรแกรม "ARMWSD"

4. สั่งเลือกกำหนดการเชื่อมต่อกับพอร์ตสื่อสารอนุกรม RS232 ให้ตรงตามเงื่อนไขที่ใช้อยู่จริง โดยให้คลิกเมาส์ที่ปุ่ม “Configure” แล้วเลือกกำหนดหมายเลข Comport ให้ถูกต้องดังภาพที่ 2.26

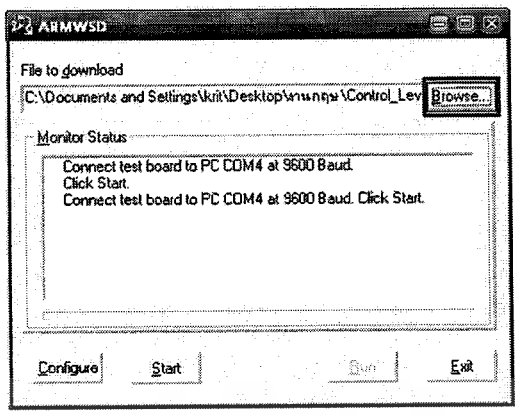


ภาพที่ 2.26 การเลือกกำหนดหมายเลข Comport

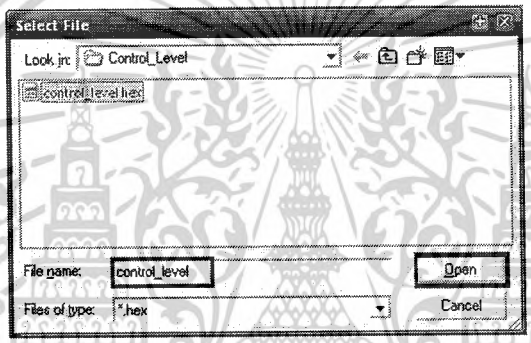


ภาพที่ 2.27 การเลือกกำหนดหมายเลข Comport

5. สั่งเปิด Hex File ที่ต้องการจะ Download ให้กับ MCU มารอไว้ในBuffer ของโปรแกรม โดยให้คลิกเมาส์ที่ปุ่ม“Browse…”แล้วเลือกกำหนดชื่อ HEX File ให้ถูกต้องเรียบร้อยดังภาพที่ 2.28

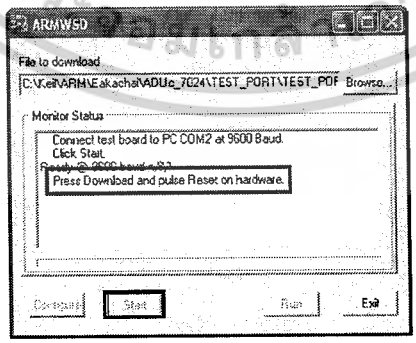


ภาพที่ 2.28 การจะ Download ให้กับ MCU มารอไว้ใน Buffer



ภาพที่ 2.29 การเลือกกำหนดชื่อ HEX File

6. คลิกเมาส์ที่ปุ่ม Start เพื่อสั่งโปรแกรมให้เริ่มการ Download โดยโปรแกรมจะตอบรับการทำงานโดยจะมีข้อความ บอกให้ผู้ใช้รีเซ็ต MCU ให้ทำงานใน Monitor Mode ดังภาพที่ 2.30

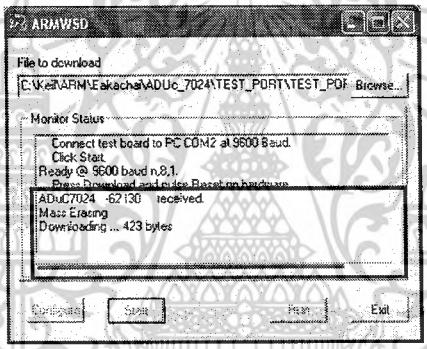


ภาพที่ 2.30 ข้อความ บอกให้ผู้ใช้รีเซ็ต MCUให้ทำงานใน Monitor Mode

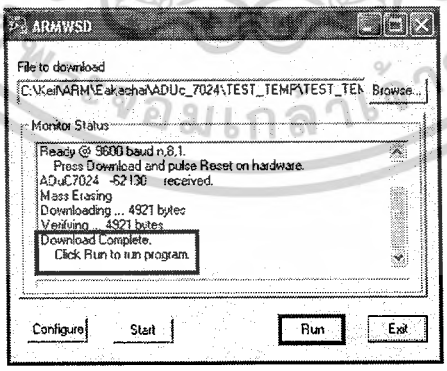
7. ให้ผู้ใช้ทำการรีเซ็ต MCU ให้เข้าทำงานใน Monitor โดยมีลำดับขั้นตอนดังนี้

- กดสวิตช์ LOAD ค้างไว้เพื่อกำหนดสถานะขาสัญญาณ P0.0 (BM) ให้เป็น “0”
- กดสวิตช์ RESET เพื่อส่งสัญญาณ RESET ให้กับ MCU โดยสวิตช์ LOAD ต้องกดค้างอยู่เช่นเดิม
- ปล่อยสวิตช์ RESET เพื่อปล่อยให้ MCU พ้นจากสภาวะการ Reset (สวิตช์ LOAD ยังกดค้างอยู่)
- ปล่อยสวิตช์ LOAD เป็นลำดับสุดท้าย

8. ถ้าทุกอย่างถูกต้องตามขั้นตอน โปรแกรมจะเริ่มต้นทำการ Download HEX File ให้กับ MCU ทันทีซึ่งในขั้นตอนนี้ให้รอจนการทำงานของโปรแกรมเสร็จเรียบร้อย แล้วจึง คลิกเมาส์ที่ปุ่ม “Run” หรือกดสวิตช์ RESET ที่บอร์ด เพื่อให้ MCU เริ่มต้นทำงานตามโปรแกรมที่ Download ให้ ดังภาพที่ 2.31



ภาพที่ 2.31 การเริ่มต้นทำการ Download HEX File



ภาพที่ 2.32 การคลิกเมาส์ที่ปุ่ม “Run”เพื่อให้ MCU เริ่มต้นทำงาน

บทที่ 3
ทฤษฎีตัวควบคุม PID

เครื่องควบคุมหรือตัวควบคุม เป็นส่วนประกอบที่สำคัญของระบบควบคุมแบบอัตโนมัติ การควบคุมอัตโนมัติก็คือ การใช้ตัวควบคุม (Controller) ทำหน้าที่เปรียบเทียบตัดสินและปรับแทนมนุษย์ให้มีผลตอบสนองเป็นไปตามต้องการ ซึ่งสัญญาณดังกล่าวจะเป็นไปตามกฎ และรูปแบบของการควบคุมที่ผู้ควบคุมได้เลือก และกำหนดไว้ล่วงหน้า การควบคุมไม่ว่าจะเป็นการควบคุมด้วยมือหรืออัตโนมัติ ผลการควบคุมจะถูกวัดและป้อนกลับไปยังตัวควบคุมเพื่อทำการเปรียบเทียบ ตัดสิน และ ออกคำสั่งปรับใหม่การกระทำจะดำเนินไปซ้ำๆกัน เช่นนี้โดยลำดับ

ปัจจุบันเครื่องควบคุมสามารถแบ่งตามโครงสร้างการทำงานได้ 3 ชนิด คือเครื่องควบคุมแบบนิวเมตริกส์ที่ทำงานโดยใช้สัญญาณลม เครื่องควบคุมแบบอิเล็กทรอนิกส์ที่ทำงานโดยใช้วงจรอิเล็กทรอนิกส์เชิงเส้น (Linear Circuit) ในการสร้างสัญญาณควบคุม และเครื่องควบคุมแบบดิจิตอลทำงานโดยใช้วงจรตรรกะ(Logic Circuit) หรือไมโครโปรเซสเซอร์สร้างสัญญาณควบคุม โดยเครื่องควบคุม ที่ถูกนำมาใช้และเป็นที่รู้จักกันดีมากที่สุดในวงการอุตสาหกรรมการผลิตก็คือ ตัวควบคุมแบบ PID

3.1 การควบคุมแบบป้อนกลับด้วยตัวควบคุมพีไอดี

ระบบควบคุมแบบ Closed-loop นั้นเป็นระบบควบคุมแบบหนึ่งซึ่งสัญญาณเอาท์พุทจะมีผลโดยตรงต่อการควบคุม ดังนั้นระบบควบคุมแบบ Closed-loop ก็คือ ระบบควบคุมป้อนกลับนั่นเอง สัญญาณค่าความคลาดเคลื่อนซึ่งเป็นสัญญาณแตกต่างระหว่างสัญญาณอินพุทกับสัญญาณป้อนกลับจะถูกป้อนให้ตัวควบคุมเพื่อที่จะได้ลดความคลาดเคลื่อนให้น้อยลง และ ทำให้เอาท์พุทของระบบมีค่าตามที่ต้องการ ซึ่งสัญญาณที่ใช้ป้อนกลับนี้อาจเป็นสัญญาณเอาท์พุทโดยตรงหรือเป็นสัญญาณที่เป็นฟังค์ชั่นของสัญญาณเอาท์พุท

ระบบควบคุมป้อนกลับโดยทั่วไปประกอบด้วยอุปกรณ์ ดังนี้

1. ตัวควบคุม (Controller) คือ ตัวรับค่าสัญญาณวัดเพื่อนำมาเปรียบเทียบเป็นค่าเป้าหมายแล้วคำนวณหาค่าที่เหมาะสม เพื่อส่งเป็นสัญญาณควบคุมออกไปควบคุมโปรเซสซึ่งเราสามารถจะตั้งเป้าหมายให้กับตัวควบคุมนี้ได้ ซึ่งการควบคุมมีหลายแบบ เช่น ON-OFF Control, P Control, PI Control, PID Control เป็นต้น

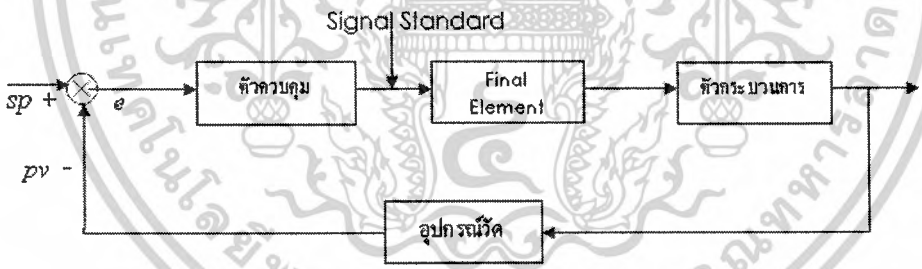
2. อุปกรณ์วัด (Measuring Instruments) หมายถึง อุปกรณ์ได้แก่ Sensor, Transducer หรืออุปกรณ์แปลงสัญญาณ (Converter) หรือวัดสัญญาณอื่นๆ ที่มีเอาท์พุทตามสัญญาณมาตรฐานเอาท์พุท

2. อุปกรณ์วัด (Measuring Instruments) หมายถึง อุปกรณ์ได้แก่ Sensor, Transducer หรืออุปกรณ์แปลงสัญญาณ (Converter) หรือวัดสัญญาณอื่นๆ ที่มีเอาท์พุทตามสัญญาณมาตรฐานเอาท์พุท

3. อุปกรณ์ปรับกระบวนการ (Final Control Element) เป็นอุปกรณ์ที่ทำหน้าที่ปรับสภาวะของกระบวนการด้วยสภาวะของกระบวนการเปลี่ยนแปลงตามค่าสัญญาณควบคุม (Manipulated Variable) ของกฎการควบคุมอุปกรณ์พวกนี้ได้แก่วาล์วควบคุม (Control Valve) , inverter, Actuator ต่างๆเป็นต้น

4. ตัวกระบวนการ (Plant or process) คือ กระบวนการที่ถูกควบคุมหรือกระบวนการทางฟิสิกส์ที่เราต้องการควบคุมให้มีสภาวะตามต้องการ ขณะที่สภาวะการทำงานหรือสภาพแวดล้อมอาจเปลี่ยนแปลงอยู่ตลอดเวลากระบวนการซึ่ง ได้แก่ อุณหภูมิ , ความดัน , อัตราการไหล , ระดับค่าความเป็นกรดด่าง เป็นต้น

5. สัญญาณมาตรฐาน (Standard Signal) ในการที่เราต้องการเชื่อมต่ออุปกรณ์ในระบบให้ควบคุมอัตโนมัติในทำงานได้ตามต้องการซึ่งจำเป็นต้องมีมาตรฐานรองรับ ซึ่งวิวัฒนาการตั้งแต่เริ่มมีระบบควบคุมอัตโนมัติมานั้นก็มีการเปลี่ยนแปลงระบบตัวอุปกรณ์ทางเครื่องมือวัดมาตั้งแต่ยุคการใช้ (Pneumatic), ไฟฟ้า (Electrical) แล้วปัจจุบันเริ่มมีการใช้สัญญาณดิจิตอลกันแล้ว สัญญาณลม 3-15 PSI, สัญญาณไฟฟ้า 1-5 Vd.c หรือ 4-20 mA



ภาพที่ 3.1 แสดงโครงสร้างของระบบควบคุมแบบป้อนกลับโดยทั่วไป

การควบคุมกระบวนการทางอุตสาหกรรม โดยทั่วไปนิยมใช้ตัวควบคุมแบบพีไอดี (PID) เพราะรูปแบบของตัวควบคุมเป็นตัวควบคุมที่สามารถควบคุมกระบวนการต่างๆ ได้อย่างกว้างขวาง เนื่องจากมีโครงสร้างการทำงานที่ไม่ซับซ้อน สามารถเข้าใจได้ง่าย การใช้งานตัวควบคุมพีไอดีนี้ขึ้นอยู่กับการปรับค่าพารามิเตอร์ของตัวควบคุม PID ให้เหมาะสม เพื่อให้ได้ผลตอบสนองของกระบวนการตามต้องการ

ตัวควบคุม PID ประกอบด้วยตัวควบคุมแบบ Proportional (P) ตัวควบคุมแบบ Integral (I) ตัวควบคุมแบบ Derivative (D) ซึ่งมีฟังก์ชั่นถ่ายโอน (Transfer function) ดังนี้

$$m(t) = Kp\left(e(t) + \frac{1}{Ti}\int_0^t e(t)dt + Td\frac{de(t)}{dt} \right) \quad (3.1)$$

โดยที่ Kp = ค่าอัตราขยายของตัวควบคุมแบบ P (Proportional Gain)

Ti = ค่าเวลา Integral (Integral Time)

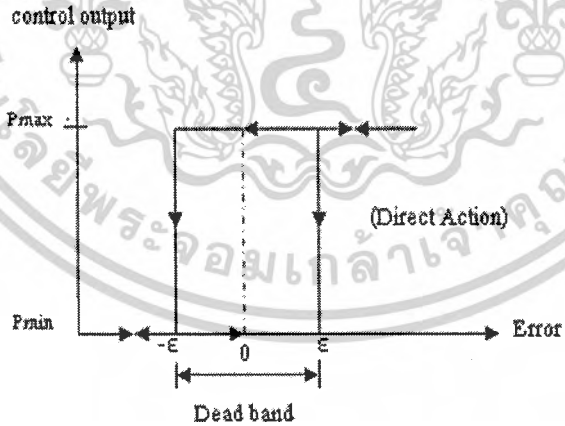
Td = ค่าเวลา Derivative (Derivative Time)

จากภาพที่ 3.1 จะเห็นได้ว่าสัญญาณควบคุมหรือ ตัวแปรปรับกระบวนการ (mv) ที่ได้จากตัวควบคุม PID จะถูกกำหนดด้วยความสัมพันธ์ระหว่างค่าสัญญาณขาเข้าด้วยตัวควบคุมกับตัวแปรกระบวนการ (pv) กับสัญญาณอ้างอิงหรือค่าเป้าหมาย (sp) โดยที่ความสัมพันธ์ดังกล่าวจะขึ้นอยู่กับกฎเกณฑ์การควบคุมที่ผู้ควบคุมปรับแต่งไว้ล่วงหน้า (ค่าพารามิเตอร์ของตัวควบคุม) จะเป็นไปตามกริยาควบคุมแบบต่างๆดังที่กล่าวต่อไปนี้

3.2 กริยาการควบคุมแบบป้อนกลับ

3.2.1 กริยาการควบคุมแบบ ON-OFF

การควบคุมแบบ ON-OFF เป็นการควบคุมที่ง่ายที่สุด และนิยมที่จะใช้เพื่อการควบคุมทางกระบวนการที่ไม่ต้องการความเที่ยงตรงสูง โดยการควบคุมนั้นจะทำงานเพียง 2 สถานะ คือให้เปิด (100%) กับปิด (0%)กริยาการควบคุมแบบ ON-OFF ดังแสดง ภาพที่ 3.2



ภาพที่ 3.2 แสดงกริยาการควบคุมแบบ ON-OFF

จากภาพที่ 3.2 จะเห็นว่าถ้าค่าความคลาดเคลื่อนมากกว่าค่าวิกฤต (+ε) ค่าเอาท์พุทของตัวควบคุมจะเปลี่ยนจาก 0% เป็น 100% เมื่อค่าความคลาดเคลื่อนน้อยกว่าค่าวิกฤต (-ε)ค่าเอาท์พุทของตัวควบคุมจะเปลี่ยนจาก 100% เป็น 0%ค่าเอาท์พุทที่อยู่ในช่วงเดทแบนด์ (Dead Band) จะไม่มีการ

เที่ยงตรงของการควบคุมลดลง กริยาการควบคุมแบบ ON-OFF สามารถเขียนเป็นสมการ(3.2) ได้ดังนี้

$$m(t) = \begin{cases} 0\% & , e < - \in \\ 100\% & , e > + \in \end{cases} \qquad (3.2)$$

เมื่อ $m(t)$ = สัญญาณควบคุมหรือเอาท์พุทของตัวควบคุม

$e(t)$ = ค่าความคลาดเคลื่อน

$\in$ = $\frac{1}{2}$ ของค่าเดทแบนด์

3.2.2 กริยาการควบคุมแบบ Proportional (P)

กริยาการควบคุม P นั้น ค่าเอาท์พุทของตัวควบคุมจะแปรผันตรงกับค่าความคลาดเคลื่อน กล่าวคือ ถ้าค่าความคลาดเคลื่อนมีค่ามากขึ้น ค่าเอาท์พุทของตัวควบคุมก็จะมีค่ามากขึ้นตาม และถ้าค่าความคลาดเคลื่อนมีค่าน้อยลง ค่าเอาท์พุทของตัวควบคุมก็จะมีค่าน้อยลงตามกริยาการควบคุมแบบ P สามารถเขียนสมการ(3.3)ได้ดังนี้

$$mp(t) = Kpe(t) + \overline{m} \qquad (3.3)$$

$mp(t)$ = ค่าเอาท์พุทของตัวควบคุมแบบ Proportional

Kp = อัตราขยายของตัวควบคุมแบบ Proportional

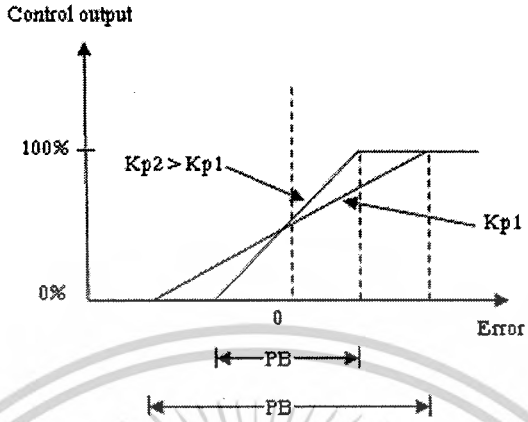
$\overline{m}$ = ค่าเอาท์พุทของตัวควบคุมที่ค่าความคลาดเคลื่อนเท่ากับศูนย์

ตัวควบคุมแบบ P บางตัวอาจจะใช้ค่า Proportional Band(PB) แทนการใช้ค่า Kp ซึ่ง PB คือช่วงของค่าความคลาดเคลื่อนในระหว่างที่เอาท์พุทของตัวควบคุมมีค่า 0-100% ดังสมการ (3.4)

$$PB = \frac{100\%}{Kp} \qquad (3.4)$$

ข้อเสียของกริยาการควบคุมแบบ Proportional คือ ไม่สามารถกำจัดค่าออฟเซ็ตได้

ข้อเสียของกริยาการควบคุมแบบ Proportional คือ ไม่สามารถกำจัดค่าออฟเซ็ตได้



ภาพที่ 3.3 แสดงคุณสมบัติของกริยาการควบคุมแบบ Proportional

3.2.3 กริยาการควบคุมแบบ Integral (I)

กริยาการควบคุมแบบ I หรือ เรียกอีกอย่างหนึ่งว่า การควบคุมแบบรีเซ็ต (Reset Control) โดยค่าเอาท์พุทของตัวควบคุมซึ่งหาได้จากค่าพื้นที่ทั้งหมดภายใต้กราฟของค่าความคลาดเคลื่อนต่อเวลาคูณกับค่าคงที่ ที่เรียกว่า อัตราขยายของตัวควบคุมแบบ I (Integral Gain) ดังสมการ (3.5) ต่อไปนี้

$$m_I(t) = K_I \int_0^t e(t)dt + \bar{m}_I(0) \qquad (3.5)$$

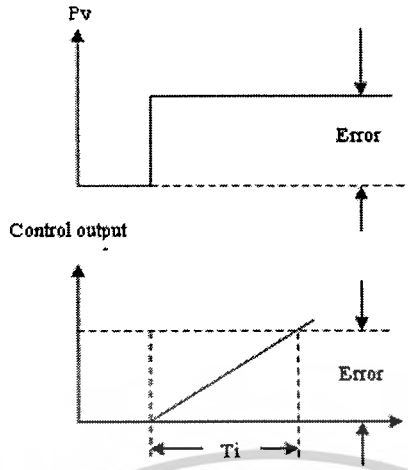
เมื่อ $m_I(t)$ = ค่าเอาท์พุทของตัวควบคุมแบบ Integral

K_I = อัตราขยายของตัวควบคุมแบบ Integral

$\int_0^t e(t)dt$ = พื้นที่ทั้งหมดของค่าความคลาดเคลื่อน

$\bar{m}_{I(0)}$ = ค่าเอาท์พุทของตัวควบคุมที่เวลา t เท่ากับศูนย์

ผลของกริยาการควบคุมแบบ I นี้จะทำให้ไม่เกิดค่าของออฟเซ็ตขึ้นในระบบ และ ลดค่าพุ่งเกิน (Overshoot) ของระบบลงได้ แต่ถ้ากริยาการควบคุมมีค่าสูงเกินไปจะทำให้ผลตอบสนองของกระบวนการช้าลง ผลตอบสนองของกริยาการควบคุมแบบ I ดังแสดงในภาพที่ 3.4



ภาพที่ 3.4 แสดงผลตอบสนองของกริยาการควบคุมแบบ Integral

3.2.4 กริยาการควบคุมแบบ Derivative (D)

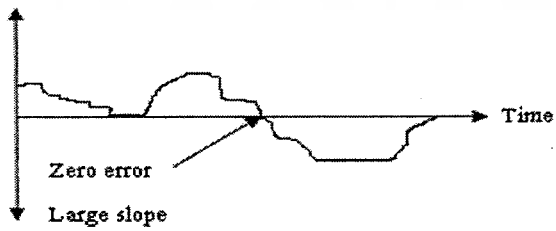
กริยาการควบคุมแบบ D เรียกอีกชื่อหนึ่งว่า การควบคุมแบบอัตราส่วน (Rate Action) โดยสัญญาณเอาท์พุทของตัวควบคุมจะขึ้นอยู่กับอัตราการเปลี่ยนแปลงของค่าความคลาดเคลื่อนต่อเวลา จะเห็นว่าค่าความคลาดเคลื่อนนี้มีโอกาสเป็นศูนย์ได้ และค่าเอาท์พุทก็สามารถเปลี่ยนแปลงให้มีค่าสูงขึ้น เมื่อความคลาดเคลื่อนเปลี่ยนแปลง ซึ่งเรียกการกระทำดังกล่าวว่า อัตราการกระทำ (Rate Action)ดังสมการ(3.6) ต่อไปนี้

$$m_D(t) = K_D \frac{de(t)}{dt} \tag{3.6}$$

เมื่อ $m_D(t)$ = ค่าเอาท์พุทของตัวควบคุมแบบ Derivative

K_D = อัตราขยายของตัวควบคุมแบบ Derivative

$\frac{de(t)}{dt}$ = ค่าความคลาดเคลื่อนที่เวลา t



ภาพที่ 3.5 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ Derivative

3.2.5 กริยาการควบคุมแบบ Proportional- Integral (PI)

ตามที่กล่าวมาแล้วว่ากริยาการควบคุมแบบ P นั้น จะมีออฟเซ็ทเกิดขึ้น ซึ่งการกำจัดค่าของออฟเซ็ทนี้ สามารถทำได้โดยการเพิ่มกริยาการควบคุมแบบ I เข้าไป ดังสมการต่อไปนี้

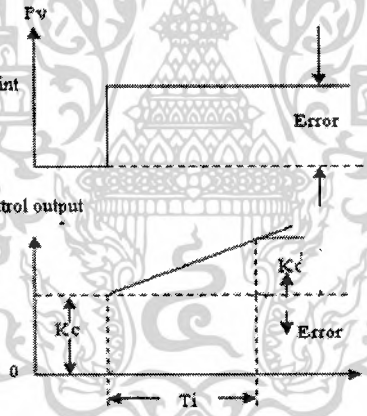
$$m_{PI}(t) = \overline{m} + K_P e(t) + K_P K_I \int_0^t e(t)dt \tag{3.7}$$

หรือ

$$m_{PI}(t) = \overline{m} + K_c e(t) + \frac{K_c}{T_i} \int_0^t e(t)dt \tag{3.8}$$

เมื่อ $K_c = K_P$ และ $K_I = \frac{1}{T_i}$

T_i = ค่าเวลา Integral



ภาพที่ 3.6 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบPI (Direct action)

3.2.6 กริยาการควบคุมแบบ Proportional-Derivative (PD)

การประยุกต์ใช้กริยาการควบคุมแบบ P ร่วมกับกริยาการควบคุมแบบ D เพื่อทำให้ผลตอบสนองของระบบรวดเร็วขึ้น แต่จะไม่มีผลโดยตรงต่อผลตอบสนองของระบบที่สภาวะคงที่ไว้ซึ่งสมการเอาท์พุตของกริยาการควบคุมแบบ PD แสดงดังสมการ(3.9)ต่อไปนี้

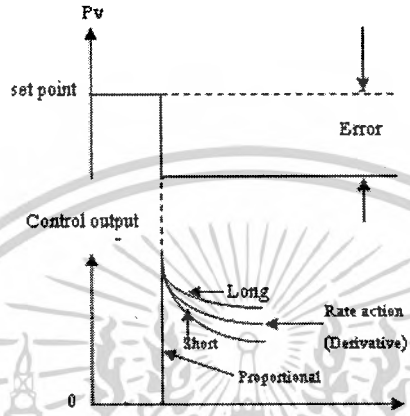
$$m_{PD}(t) = \overline{m} + Kpe(t) + K_p K_D \frac{de(t)}{dt} \tag{3.9}$$

หรือ

$$m_{PD}(t) = \overline{m} + K_c e(t) + K_c T_d \frac{de(t)}{dt}$$

เมื่อ $K_D = T_d$

T_d = ค่าเวลา Derivative



ภาพที่ 3.7 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ PD

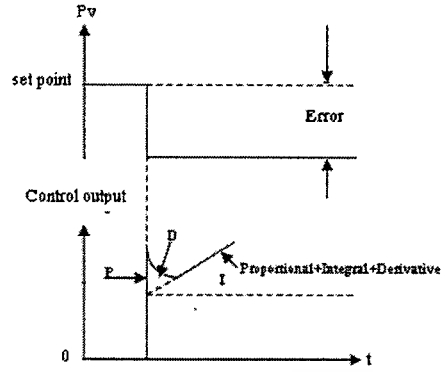
3.2.7 กริยาการควบคุมแบบ Proportional – Integral- Derivative (PID)

เพื่อให้ผลตอบสนองของระบบควบคุมมีสมรรถนะเป็นไปตามต้องการ จึงใช้กริยาในการควบคุมทั้ง 3 แบบ ร่วมกัน ซึ่งจะทำให้ได้กริยาการควบคุมแบบ PID ที่มีสมการดัง (3.10) ต่อไปนี้

$$m_{PID}(t) = \overline{m} + Kpe(t) + KpK_I \int_0^t e(t)dt + KpK_D \frac{de(t)}{dt} \qquad (3.10)$$

หรือ

$$m_{PID}(t) = \overline{m} + K_c e(t) + \frac{K_c}{T_i} \int_0^t e(t)dt + K_c T_d \frac{de(t)}{dt}$$



ภาพที่ 3.8 แสดงตัวอย่างผลตอบสนองของกริยาการควบคุมแบบ PI (Direct action)

3.3 การคำนวณหาค่าพารามิเตอร์ของตัวควบคุม PID จากผลตอบสนองขอกระบวนการ

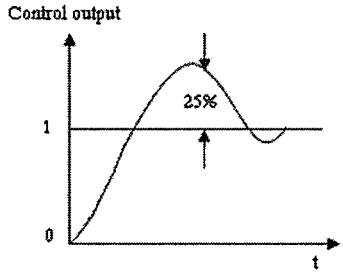
การนำตัวควบคุม PID ไปใช้ในการควบคุมกระบวนการต่างๆ จำเป็นที่จะต้องทำการปรับค่าพารามิเตอร์ของตัวควบคุม PID ให้เหมาะสมเพื่อให้ได้ผลตอบสนองที่ดีตามความต้องการซึ่งในการหาค่าพารามิเตอร์ของตัวควบคุม PID นี้สามารถทำได้หลายวิธี ซึ่งบางวิธีอาจพิจารณาได้จากผลตอบสนองของกระบวนการจากการทดสอบทางกระบวนการ หรือจะอาศัยประสบการณ์จากการควบคุม บางวิธีอาจจะอาศัยทฤษฎีทางคณิตศาสตร์ และส่วนใหญ่แล้วจะพิจารณาจากเงื่อนไขของรูปแบบของโดเมนเวลามากกว่าในโดเมนความถี่ซึ่งวิธี Ziegler-Nichols นี้เป็นวิธีที่ได้รับความนิยมมากที่สุดในทางปฏิบัติ

3.3.1 การปรับค่าพารามิเตอร์ของตัวควบคุม PID โดยวิธีของ Ziegler-Nichols

การหาค่า K_p, T_i และ T_d จะขึ้นอยู่กับค่าทางคุณลักษณะของผลตอบสนองชั่วครู่ซึ่งในกระบวนการที่ถูกควบคุมมี 3 วิธี คือ

1. วิธี Process Reaction Curve Method (Open Loop Method)
2. วิธี Ultimate Method (Close Loop)
3. วิธี Trial and Error (วิธีการลองผิดลองถูก)

ซึ่งแต่ละวิธีมีจุดมุ่งหมายที่จะทำให้ผลตอบสนองต่อเวลาของกระบวนการต่ออินพุทแบบขั้วบันได มีค่าพุ่งเกินสูงสุดไม่เกิน 25% ดังแสดงในภาพที่ 3.9



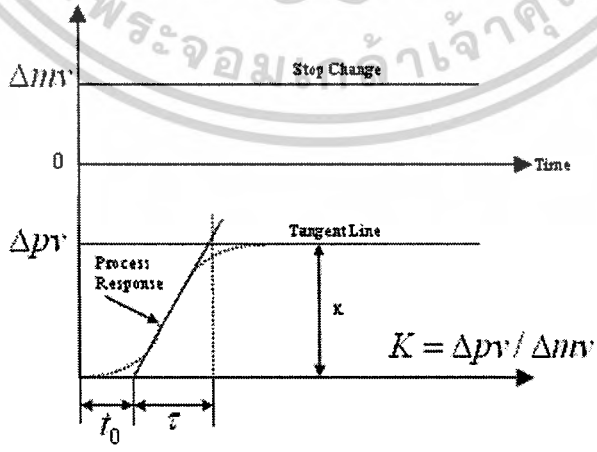
ภาพที่ 3.9 แสดงค่าพุ่งเกินที่ต้องการเมื่อใช้การปรับด้วยวิธี Ziegler-Nichols

3.3.1.1 วิธี Process Reaction Curve(การทดลองแบบลูปเปิด)

วิธีนี้จะหาค่าพารามิเตอร์ของตัวควบคุม PID จากผลตองสนองของกระบวนการแบบลูปเปิดที่ถูกควบคุมด้วยสัญญาณอินพุทแบบขั้นบันได โดยที่กระบวนการในกรณีนี้จะไม่มี Pole ที่จุด Origin หรือไม่มี Dominant complex-conjugate poles ดังนั้นผลตอบสนองของเวลาจะได้ในลักษณะเป็นรูปตัว S (ถ้าผลตอบสนองของเวลาไม่มีลักษณะเป็นรูปตัว S วิธีนี้จะใช้ไม่ได้) จากผลตอบสนองต่อเวลาสามารถแยกพิจารณาเป็นค่าเวลาหน่วงของกระบวนการ(Delay time: t_0) และเวลาคงที่ของกระบวนการ (Time Constant:τ) โดยลากเส้นสัมผัสกับจุดที่มีอัตราการเบี่ยงเบนสูงสุด เส้นนี้จะตัดกับแกนเวลา และเส้นตรงของเอาท์พุทC(t) ที่ C (t) =K

ดังนั้นฟังก์ชั่นถ่ายโอนจะประมาณได้ด้วยระบบอันดับหนึ่ง แบบมีการหน่วงเวลา (FOPDT) ดังสมการที่ 3.11

$$G(s) = \frac{C(s)}{U(s)} = \frac{Ke^{-t_0 s}}{\tau s + 1} \tag{3.11}$$



ภาพที่ 3.10 แสดงผลตอบสนองรูปตัว S เมื่อปรับโดยใช้วิธี Process Reaction Curve

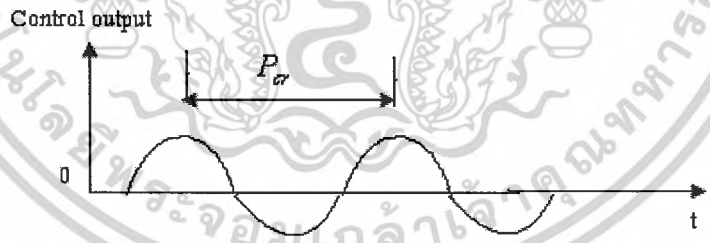
Ziegler-Nichols ได้กำหนดค่าของ K_p, T_i และ T_d สำหรับตัวควบคุมแบบต่างๆดังตาราง

ตารางที่ 3.1 แสดงค่าพารามิเตอร์ของตัวควบคุมแบบต่างๆตามวิธี Process Reaction Curve

Controller Type	**Proportional Gain** K_c	**Integral Time** T_i	**Derivative Time** T_d
Proportional only P	$\frac{1}{K}\left(\frac{\tau}{t_0}\right)$	-	-
Proportional – Integral PI	$\frac{0.9}{K}\left(\frac{\tau}{t_0}\right)$	$3.33t_0$	-
Proportional-Integral-Derivative PID	$\frac{1.2}{K}\left(\frac{\tau}{t_0}\right)$	$2.0t_0$	$0.5t_0$

3.3.1.2 วิธี Ultimate Method (Close Loop)

วิธีนี้จะหาค่าพารามิเตอร์ของตัวควบคุม PID จากผลตอบสนองของกระบวนการแบบลูปปิดที่ถูกควบคุมด้วยตัวควบคุมแบบ P ต่อสัญญาณอินพุทแบบขั้นบันได โดยปรับค่า K_p ไปเรื่อยๆ จนผลตอบสนองของกระบวนการเกิดการแกว่งอย่างต่อเนื่อง (Sustained Oscillations) ถ้าผลตอบสนองเวลาไม่เกิดการแกว่งอย่างต่อเนื่อง วิธีการนี้จะใช้ไม่ได้ ดังแสดงในภาพที่ 3.11



ภาพที่ 3.11 แสดงผลตอบสนองเวลาเกิดการแกว่งอย่างต่อเนื่อง
เมื่อปรับโดยใช้วิธี Ultimate method

จากการหาค่าของ

K_{cr} (Critical Gain) คือ อัตราการขยายที่ทำให้ผลตอบสนองเวลาเกิดการแกว่งได้อย่างต่อเนื่อง

P_{cr} (Oscillation Period) คือ คาบเวลาของการแกว่งอย่างต่อเนื่อง

Ziegler-Nichols ได้กำหนดค่าของ K_p, T_i และ T_d สำหรับตัวควบคุมแบบต่าง ๆ ดังตาราง

ตารางที่ 3.2 แสดงค่าพารามิเตอร์ของตัวควบคุมแบบต่างๆตามวิธี Ultimate Method

Controller Type	**Proportional Gain** K_c	**Integral Time** T_i	**Derivative Time** T_d
Proportional only P	$\frac{K_{cr}}{2}$	-	-
Proportional – Integral PI	$\frac{K_{cr}}{2.2}$	$\frac{T_{cr}}{1.2}$	-
Proportional-Integral-Derivative PID	$\frac{K_{cr}}{1.7}$	$\frac{T_{cr}}{2}$	$\frac{T_{cr}}{8}$

การคำนวณหาค่าพารามิเตอร์ของตัวควบคุมของ PID โดยวิธีของ Ziegler-Nichols นั้นไม่ใช่เป็นค่าที่เที่ยงตรงที่จะนำไปใช้งานได้ทันที และมีค่าพุ่งเกินสูงสุดไม่เกิน 25 % ตามที่ได้กล่าวไว้ แต่ค่าที่ได้อาจจะมีเพียงแค่ใกล้เคียงเท่านั้น หรือ ไม่ใกล้เคียงก็ตามผู้ควบคุมจะต้องทำการปรับค่าของพารามิเตอร์เหล่านี้แบบละเอียด (Fine Tuning) อีกครั้งหนึ่ง

3.3.1.3 วิธี Trial and Error (วิธีการลองผิดลองถูก)

เป็นวิธีการที่เหมาะสำหรับผู้ที่มีประสบการณ์ในด้านการปรับพารามิเตอร์ PID แล้ว โดยการควบคุมจะต้องเป็นระบบปิดและตำแหน่งการควบคุมจะต้องอยู่ในตำแหน่งAutomatic Control ส่วนวิธีการต่อไปนี้ก็คือ

วิธีการปรับแบบ PI

1. ให้ตัวควบคุมทำงานในรูปแบบ Proportional Control เพียงอย่างเดียว
2. ต่อไปทำการปรับค่า Kc จนกระทั่งค่า PV เข้าใกล้ค่าเป้าหมายซึ่งโดยทั่วไปก็จะต่ำกว่าค่า เป้าหมายเพียงเล็กน้อย
3. ต่อไปค่อยๆปรับค่า Ti เพิ่มขึ้นเพื่อชดเชยค่า Offset ที่เกิดขึ้นแล้วดูผลว่าค่า PV เข้าสู่ค่าเป้าหมายตามที่ต้องการหรือยังถ้ายังก็ค่อยๆปรับขึ้นอีกจนกระทั่งได้ตามต้องการ โดยช่วงนี้ก็พยายามลด Kc เพื่อลดการแกว่ง

วิธีการปรับแบบ PID

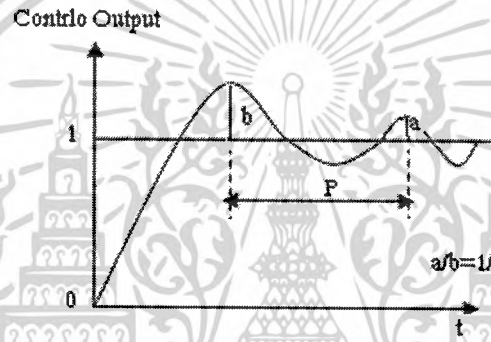
1. ทำตามวิธีการแบบ PI ทั้ง 3 ข้อ

2. ถ้าต้องการผลตอบสนองที่รวดเร็วขึ้นมักจะใช้ค่าอัตราส่วนระหว่างค่า Integral Time กับ Derivative Time เป็น 4/1 เป็นต้นไปพร้อมกับการเพิ่มลดของค่า Kc ไปด้วยเพื่อที่จะใช้ป้องกันการเกิด Overshoot และการเกิด Hunting

3. เนื่องด้วยวิธีการนี้เป็นความพึงพอใจเฉพาะบุคคลจึงยากที่จะกำหนดให้เป็นไปตามกฎเกณฑ์ตายตัว แต่วิธีการเริ่มต้นในการปรับนั้นมักเหมือนกัน

3.3.2 การปรับค่าพารามิเตอร์ของตัวควบคุม PID โดยวิธี Damped Oscillation

วิธีปรับปรุงมาจากวิธี Ultimate Method โดย Harriott เพื่อใช้ในกรณีที่การปรับค่าของ K_p ไปอย่างไรก็ตาม แต่ผลตอบสนองต่อเวลาไม่เกิดการแกว่งอย่างต่อเนื่อง ดังแสดงในภาพ



ภาพที่ 3.12 แสดงค่าอัตราการเสื่อม 1/4เมื่อปรับโดยวิธี Damped Oscillation

วิธี Damped Oscillation นี้จะปรับค่าของ K_p ไปจนถึงผลตอบสนองต่อเวลาของระบบที่ใช้ควบคุมแบบลูปปิดมีอัตราเสื่อม1/4 ดังภาพ จากนั้นวัดค่าของ P และใช้ค่า K_p (1/4) เพื่อคำนวณหาค่า K_p, T_i และ T_d ดังนี้

$$K_p = (1/4)$$

$$T_i = P/1.5$$

$$T_d = P/6$$

3.4 ข้อกำหนด (Specifications) ของผลตอบสนองชั่วครู่ของระบบ

สมรรถนะของระบบควบคุม จะแสดงในเทอมของปริมาณต่างๆในรูปของโดเมนเวลาโดยจะทำการวิเคราะห์สมรรถนะจากผลตอบสนองชั่วครู่ของระบบต่ออินพุทที่เป็น Unit step สำหรับข้อกำหนดต่างๆซึ่งจะประกอบด้วย

1. Maximum Overshoot (M_p) ซึ่งบางครั้งนั้นจะทำการแสดงแบบในเทอมของ Perfect Maximum Overshoot เป็นค่าแตกต่างระหว่างเอาท์พุทที่มีค่าสูงสุดของระบบกับเอาท์พุทที่มีค่าคงที่ นั่นคือถ้า

$C_{\max}$ คือ ค่าเอาท์พุทที่มีค่าสูงสุด

C_{ss} คือ ค่าเอาท์พุทที่มีค่าคงที่

จะได้ว่า

$$M_p = C_{\max} - C_{ss}$$

และ Perfect Maximum Overshoot :

$$\%M_p = \left(\frac{M_p}{C_{ss}}\right) \times 100 \tag{3.12}$$

Maximum Overshoot จะแสดงถึงเสถียรภาพสัมพัทธ์ของระบบโดยทั่วไปแล้วในระบบที่มี Overshoot มากนั้นจะไม่เป็นที่ต้องการ นอกจากนี้แล้วค่า Maximum Overshoot ยังเป็นข้อกำหนดสำหรับการออกแบบระบบควบคุมด้วย

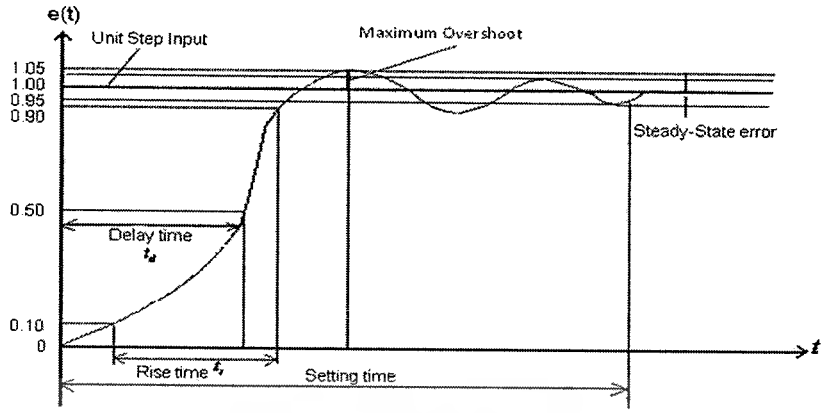
2. Delay Time (t_d) เป็นช่วงเวลาที่ผลตอบสนองชั่วครู่ของระบบที่จะมีค่าเข้าสู่ 50% ของค่าที่ภาวะคงที่

3. Rise Time (t_r) เป็นช่วงเวลาที่ผลตอบสนองชั่วครู่ของระบบจะมีค่าเพิ่มขึ้นจาก 10% ถึง 90% ของค่าที่ภาวะคงที่ในบางครั้งอาจจะถือว่า Rise Tim เป็นช่วงเวลาที่ให้ผลตอบสนอง 10% ถึง 90%, 5% ถึง 95%, 0% ถึง 100% ของค่าที่ภาวะคงที่ก็ได้

4. Settling Time (t_s) เป็นช่วงเวลาที่ผลตอบสนองชั่วครู่ของระบบที่จะมีค่าเข้าสู่ช่วง±2% หรือ ±5% ของค่าที่ภาวะคงที่และมีค่าอยู่ในช่วงนี้ตลอด หรือหมายถึง ค่าเวลาที่ผลตอบสนองต่อเวลาเปลี่ยนสภาพจากผลตอบสนองชั่วครู่ไปเป็นผลตอบสนองที่ภาวะคงที่

5. Peak Time $(t_p$ หรือ $t_{\max})$ เป็นช่วงเวลาที่ตอบสนองชั่วครู่ของการเกิดระบบ.Maximum Overshoot

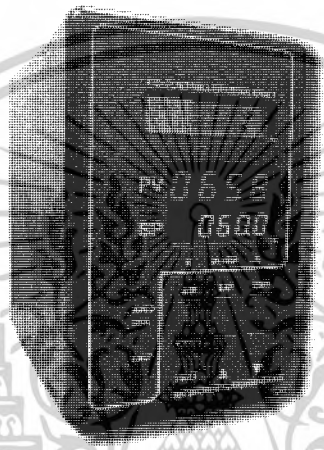
ข้อกำหนดต่าง ๆ เหล่านี้จะใช้วัดคุณลักษณะของผลตอบสนองแบบชั่วครู่ โดยอินพุทแบบ Unit step เท่านั้น แต่จะไม่สามารถนำไปใช้ในทุกกรณีได้ เช่น ระบบที่ใช้เป็นแบบ Critical Damped และ Over Damped นั้นจะไม่มีค่าของ t_p และ M_p นอกจากนี้ข้อกำหนดเหล่านี้จะใช้กับระบบที่มีเสถียรภาพเท่านั้น เพราะระบบที่ไม่มีเสถียรภาพนั้นผลตอบสนองของระบบจะมีที่ขนาดใหญ่ขึ้นเรื่อยๆ และควบคุมไม่ได้



ภาพที่ 3.13 แสดงข้อกำหนดของผลตอบสนองชั่วครู่ของระบบ

บทที่ 4
การออกแบบตัวควบคุม

ในบทนี้จะทำการกล่าวถึงวิธีการ และ ขั้นตอนของการออกแบบตัวควบคุม PID โดยใช้ตัวไมโครคอลโทรลเลอร์ ARM7 ซึ่งจะทำการแบ่งการออกแบบในส่วนนี้เป็น ฮาร์ดแวร์ และส่วนของซอฟท์แวร์ดังนี้



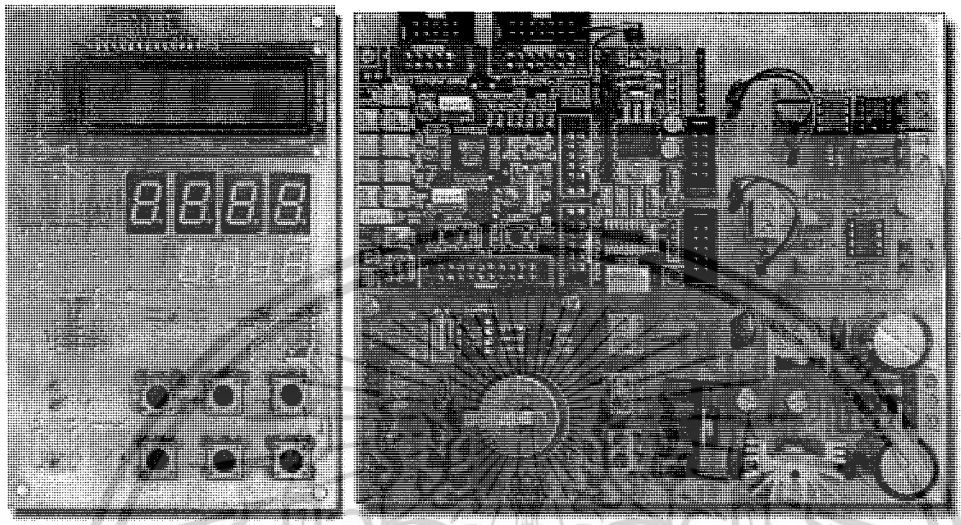
ภาพที่ 4.1 แสดงตัวควบคุม PID

1. Hardware จะประกอบด้วยวงจรต่าง ๆรวมทั้งในส่วนของการควบคุมและแสดงผล ดังนี้
   - วงจรเพาเวอร์ซัพพลาย
   - วงจร V-V
   - วงจร V-I
   - การออกแบบคีย์บอร์ด(SW)
   - การออกแบบตัวแสดงผล (Segment , LCD , LED)

2. Software ในส่วนนี้จะกล่าวถึงทฤษฎีต่างๆที่ใช้ในการออกแบบตัวควบคุม รวมไปถึงการเขียนโปรแกรมควบคุมเพื่อให้ไมโครคอนโทรลเลอร์ทำงานได้ตามที่ต้องการ เนื้อหาประกอบด้วย
   - การออกแบบอัลกอริทึม
   - การกำหนดช่วงเวลาของการแซมปลิ้ง(interrupt)
   - การแปลงอนาลอกเป็นดิจิตอล

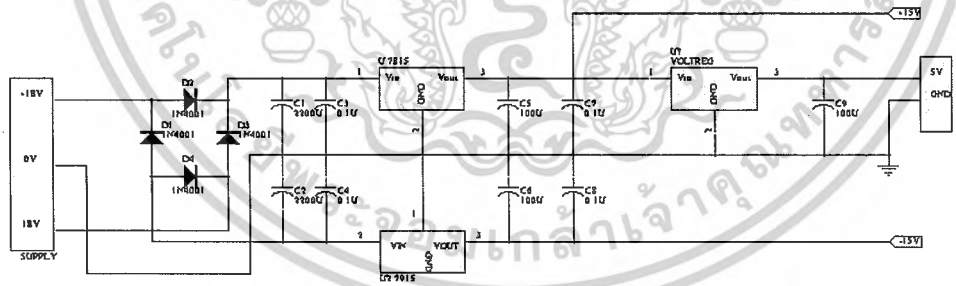
- การแปลงดิจิตอลเป็นอนาลอก
- การ Scaling

4.1 การออกแบบ HardWare



ภาพที่ 4.2 แสดงฮาร์ดแวร์ตัวควบคุมโดยรวม

4.1.1 เพาเวอร์ซับพลายชนิดสองขั้ว



ภาพที่ 4.3 แสดงวงจรเพาเวอร์ซับพลาย

จากภาพที่ 4.3 เป็นเพาเวอร์ซับพลายชนิดสองขั้ว โดยที่แต่ละขั้วจะให้แรงดัน ± 15โวลต์ D2 และ D4 จะมีหน้าที่ป้อนแรงดันที่เป็นบวกเมื่อเทียบกับกราวด์ให้กับขั้ว 1 และ D1 และ D3 มีหน้าที่ป้อนแรงดันที่เป็นลบเมื่อเทียบกับกราวด์ให้กับขั้ว 2 สำหรับตัวเก็บประจุทั้งสองตัวนั้นจะทำหน้าที่กรองแรงดันริปเปิลให้ลดลง โดย c+ จะทำหน้าที่กรองแรงดันในฝั่งบวก c- ทำหน้าที่กรองแรงดันในฝั่งลบ สำหรับวิธีการคำนวณสามารถทำได้ดังนี้

ตารางที่ 4.1 การหาขนาดของหม้อแปลง

ชนิดของเรกติไฟ	แรงดันทางด้านขดลวดทุติยภูมิ (Vsec rms)	กระแสทางด้านขดลวดทุติยภูมิ (Isec rms)
HalfWave	(V0+0.7)/1.414	1.8Io
FWCT	(V0+0.7)/1.414	0.9Io
FWB	(V0+0.7)/1.414	1.8Io
Dual FWCT	(V0+0.7)/1.414	1.8Io

ในที่นี้เราใช้เรกติไฟแบบฟลูเวฟเรกติไฟแบบมีเซนเตอร์แท็ป จากตารางจะเห็นได้ว่า

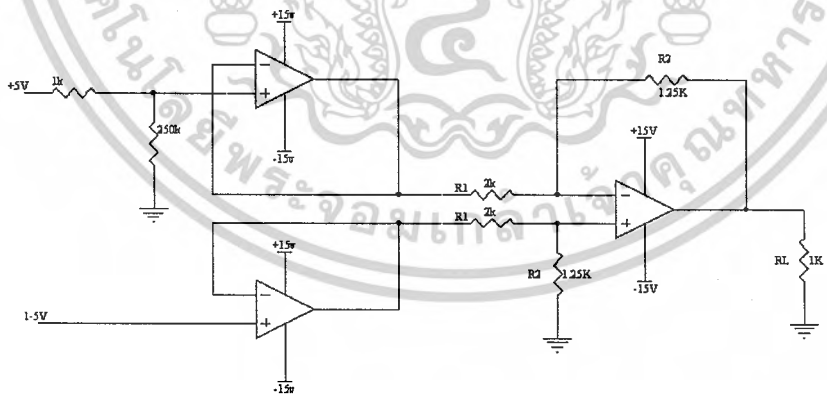
$$V\sec_{rms} = \frac{Vo + 0.7}{1.41} = \frac{15 + 0.7}{1.41} = 11.13 \quad \text{Volts}$$

$$I\sec_{rms} = 1.8Io = 1.44A$$

การหาค่า Em ที่บวก15 กับจุด CT

$$Em = 1.4(15) = 21 \quad v$$

4.1.2 วงจรปรับระดับแรงดัน (Vin = 1-5V, Vout = 0-2.5V)



ภาพที่ 4.4 แสดงวงจรปรับระดับแรงดัน

จากวงจรค่าแรงดันเอาท์พุต Vout จะขึ้นอยู่กับผลต่างแรงดันอินพุต V1 และ V2 ถ้าแรงดันอินพุต V1 และ V2 มีค่าเท่ากันจะได้ค่าแรงดันเอาต์พุต Vo จะเท่ากับศูนย์ จะเรียกสัญญาณแรงดันอินพุต V1 และ V2 นี้ว่า สัญญาณร่วม **(Common Mode Voltage)** ค่าความต้านทาน R1 และ R2 แต่

ละคู่ในวงจรจะต้องมีความสมนัยกันมาก เพื่อให้ได้อัตราขยายของวงจรต่อสัญญาณร่วมเท่ากับศูนย์ ส่วนวงจรขยายความแตกต่างที่เพิ่มวงจรตามศักดาหรือวงจรบัฟเฟอร์ (Buffer) เข้ามายังส่วนหน้าของวงจรนี้ เรียกว่า วงจรขยายแบบอินสตรูเมนท์แบบพื้นฐาน **(Basic Instrumentation Amplifier)** ซึ่งการคำนวณหาค่าความต้านทาน R1และ R2 มีดังนี้

หาค่าอัตราขยาย Av

$$A_v = \frac{Vout}{Vin} = \frac{2.5-0}{5-1} = 0.625 \tag{4.1}$$

ทำการคำนวณหาค่า R1 และ R2 จากสมการ Differential Amplifier

$$Vout = \frac{R2}{R1}(V2 - V1) \tag{4.2}$$

$$Av = \frac{Vout}{V2 - V1} = \frac{R2}{R1} \tag{4.3}$$

$$0.625R1 = R2$$

เลือกค่า $R1 = 2k\Omega$ เพื่อให้ได้คำตอบของ $R1$ และ $R2$ เท่ากับ Av

$$0.625(2k\Omega) = R2$$

$$R2 = 1.25k\Omega$$

หาค่าแรงดัน V1 จาก $Vin = 1 \quad Vout = 0$ แทนสมการที่ (4.2)

$$Vout = \frac{R2}{R1}(V2 - V1)$$

$$0 = \frac{1.25k\Omega}{2k\Omega}(1 - V1)$$

$$0 = 0.625 - 0.625V1$$

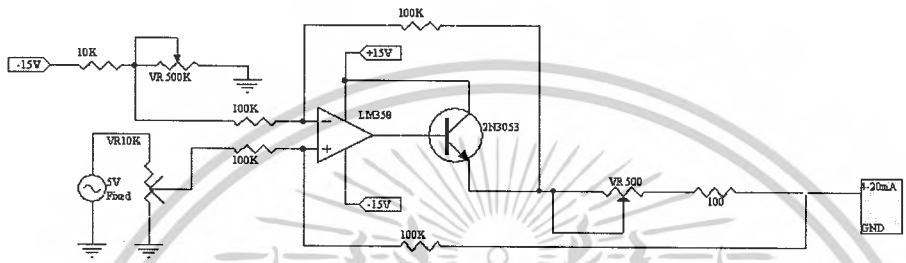
$$V1 = 1V$$

นำค่าแรงดันอินพุตแทนในสมการ (4.2)

$$Vout = \frac{1.25k\Omega}{2k\Omega}(V2 - V1)$$

จากทฤษฎีตามสมการ (2) จะได้ $Vout = 0.625(V2 - 1)$

4.1.3 วงจรแปลงสัญญาณแรงดันไฟฟ้าเป็นสัญญาณกระแสไฟฟ้า (Voltage to Current Converter Circuit)



ภาพที่ 4.5 วงจรแปลงสัญญาณแรงดันไฟฟ้าเป็นสัญญาณกระแสไฟฟ้า

เป็นวงจรที่สำคัญวงจรหนึ่งที่ใช้ในระบบควบคุมกระบวนการทางอุตสาหกรรม ทั้งนี้เนื่องจากบางครั้งอุปกรณ์ตัวส่งสัญญาณในระบบ ให้สัญญาณเอาต์พุตเป็นแรงดันไฟฟ้าแต่ถ้าจำเป็นต้องส่งไปเป็นระยะทางไกลจึงต้องส่งเป็นกระแสไฟฟ้า

จากวงจรต้องการแปลงแรงดันไฟฟ้า 0-2.5 โวลท์ เป็น กระแสไฟฟ้า 4-20 มิลลิแอมป์ ซึ่งมีขั้นตอนการคำนวณดังนี้ วงจรขยายความแตกต่างแรงดันไฟฟ้า ดังรูป ค่าความต้านทานไฟฟ้า จากภาพที่ 4.4 ถ้า R1=R2=R3=R4=R เราสามารถคำนวณแรงดันเอาท์พุตของวงจรได้โดยใช้ทฤษฎีการทับซ้อน (Superposition)

$$Vout1 = -eref \tag{4.4}$$

$$Vout = (1 + \frac{R}{R})Va \tag{4.5}$$

ซึ่ง $Va = \left(\frac{ein - Vl}{2R}\right)R + Vl$

$$Vout2 = 2Va = 2\left[\left(\frac{ein - Vl}{2R}\right)R + Vl\right] = ein + Vl \tag{4.6}$$

และ

$$Vout = Vout1 + Vout2 = Vl + ein - eref \tag{4.7}$$

และจากรูป $V_{RS} = Vout - Vl = Vl + ein - eref - Vl = ein - eref$

ดังนั้นกระแสที่ไหลผ่าน Rs ซึ่งมีค่าเท่ากับกระแสเอาท์พุท คือ

$$irs = io = \frac{V_{RS}}{RS} = \frac{ein - eref}{Rs} \tag{4.8}$$

จากสมการจะเห็นว่ากระแสเอาท์พุทขึ้นอยู่กับค่าแรงดันอินพุท แรงดันอ้างอิงและค่าความต้านทาน Rs ดังนี้

$ein = 0v \quad ; \quad io = 4mA$

$4mA = \frac{0v - eref1}{RS}$

$eref = 0 - 4mA * RS$

ที่ $ein = 2.5V \quad ; \quad io = 20\,mA$

$20mA = \frac{0.5V - eref1}{RS}$

$$eref = 2.5 - 20mA * RS \tag{4.9}$$

แทนค่าลงในสมการ จะได้

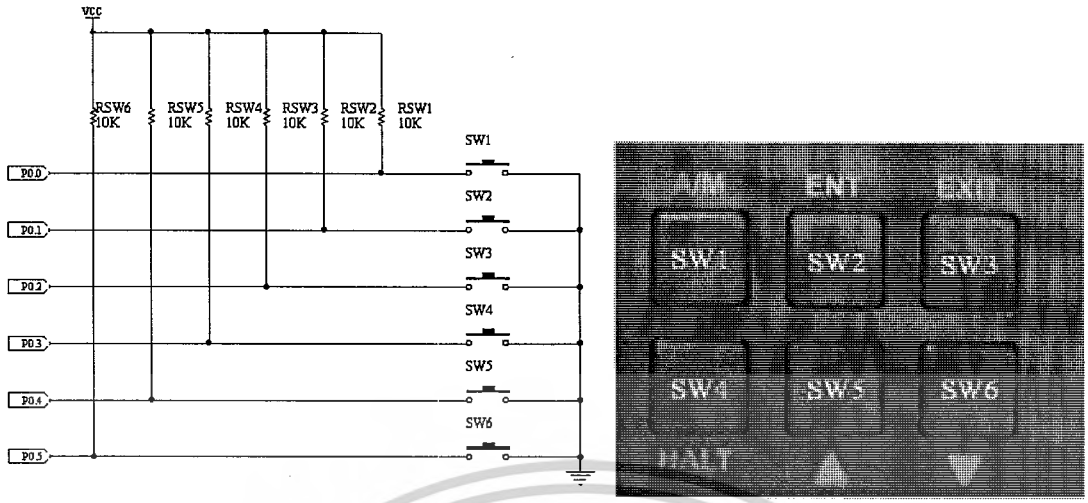
$$RS = \frac{2.5}{16mA} = 156.25$$

และ $eref = 0 - 4mA * 156.25 = -0.625$

ถ้าแทนลงในสมการ $io = \frac{ein - eref}{RS} = \frac{0 - (-0.625)}{156.25} = 4mA$

4.1.4 การออกแบบคีย์บอร์ดควบคุม(SW)

ในการกำหนดอินพุตให้กับระบบนั้นจะใช้วงจรเชื่อมต่อกับไมโครคอนโทรลเลอร์ ARM7 ดังแสดงในภาพที่ 4.6 เลือกใช้งานพอร์ต0 เป็นอินพุตพอร์ต มีดังนี้ P0.0, P0.1, P0.2, P0.3, P0.4 และ P0.5 จะเห็นว่าทุกขาจะมี R pull-up ต่อร่วมอยู่ ทั้งนี้เพื่อเป็นการกำหนดสภาวะเริ่มต้นขณะที่ไม่มีการกดคีย์ โดยจะมีสภาวะเป็นลอจิก 1 ตลอดเมื่อมีการกดคีย์สภาวะของขานั้นจะมีค่าเป็น 0



ภาพที่ 4.6 Schematic และการทำงานของ SW

หน้าที่การทำงานของสวิตซ์ต่างๆ

SW1 ทำหน้าที่เป็นปุ่ม ใช้ในการเลือกโหมดการทำงานของตัวควบคุมเป็นแบบ AUTO หรือ MAN ซึ่งจะแสดงสถานะที่ LED_AUTO และ LED_MAN

SW2 ทำหน้าที่เป็นปุ่ม Enter เลือกเข้าฟังก์ชั่น

SW3 ทำหน้าที่เป็นปุ่ม Exit ออกจากฟังก์ชั่น

SW4 ทำหน้าที่เป็นปุ่ม Halt ใช้ในการหยุดการทำงานของตัวควบคุมชั่วขณะ

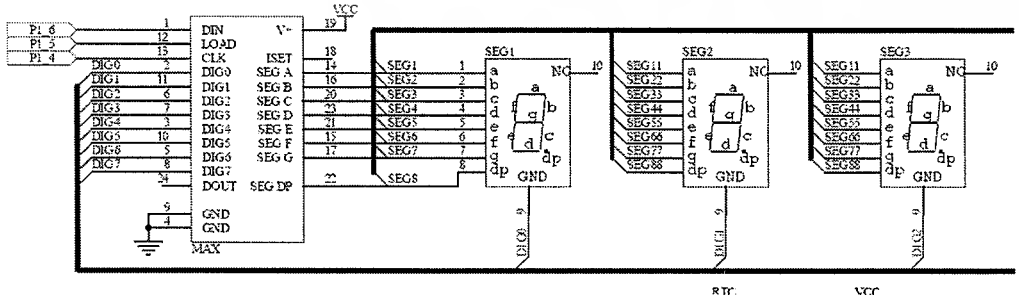
SW5 ทำหน้าที่เป็นปุ่ม ใช้ในการเลื่อนฟังก์ชั่นขึ้น, เพิ่มค่าของพารามิเตอร์ต่างๆ

SW6 ทำหน้าที่เป็นปุ่ม ใช้ในการเลื่อนฟังก์ชั่นลง, ลดค่าของพารามิเตอร์ต่างๆ

4.1.5 การออกแบบตัวแสดงผล

สำหรับการแสดงผลนั้นมีอยู่ด้วยกันหลายแบบขึ้นอยู่กับความเหมาะสมของการแสดงค่านั้นๆ ในโปรเจคนี้เลือกใช้ตัวแสดงผล 3 แบบ คือ ตัวแสดงผล7-Segment , LCD และ LED

4.1.5.1 ตัวแสดงผล 7-Segment



ภาพที่ 4.7 Schematic ของ 7-Segment

ในการควบคุมการแสดงผลของแผงแสดงผลแบบตัวเลข 7ส่วน จำนวน 8 หลักจะใช้ IC MAX 7219 เป็นตัวคุมการแสดงผล ทำให้วงจรมีขนาดเล็กและการเขียนโปรแกรมสั่งงานง่ายและสะดวกมาก

สำหรับการเขียนโปรแกรมควบคุมการทำงาน 7-segment โดยใช้ MAX 7219 นั้นจำเป็นจะต้องทำความเข้าใจวงจรใน ภาพที่ 4.6 เสียก่อนว่าพอร์ตต่างๆของไมโครคอนโทรลเลอร์ ARM7 ADuC7024 ต่ออยู่กับขาใดของ MAX 7219 ซึ่งจากวงจรจะได้ว่า

P1.4 ต่ออยู่กับขาสัญญาณ CLK

P1.5 ต่ออยู่กับขาสัญญาณ Load

P1.6 ต่ออยู่กับขาสัญญาณ DIN

ซึ่งในการออกแบบนั้นมี 7-Segment ทั้งหมด 8 หลัก เพื่อที่จะแสดงค่าการทำงานของกระบวนการดังนี้

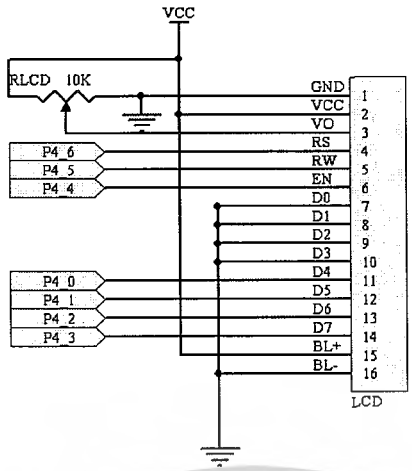
ภาพที่ 4.8 การแสดงผลของ 7-Segment บนตัวควบคุมPID

4 หลักบน ใช้แสดงค่า PV ที่ได้รับมาจากทรานซ์ดิวเซอร์

4 หลักล่าง ใช้แสดงค่าเป้าหมาย(setpoint)

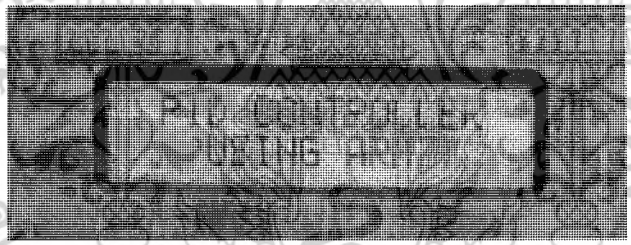
4.1.5.2 ตัวแสดงผลแบบผลึกเหลว(LCD)

จอแสดงผลที่ใช้นี้เป็นแบบ DOT MATRIX LCD MODULE เป็นอุปกรณ์แผงแสดงผลตัวอักษรหรือตัวเลข เหมาะสมสำหรับงานแสดงผลการทำงานเป็นข้อความตัวอักษร หรือข้อความต่างๆ



ภาพที่ 4.9 Schematic ของการต่อจอแสดงผล LCD กับไมโครคอนโทรลเลอร์ ADuC7024

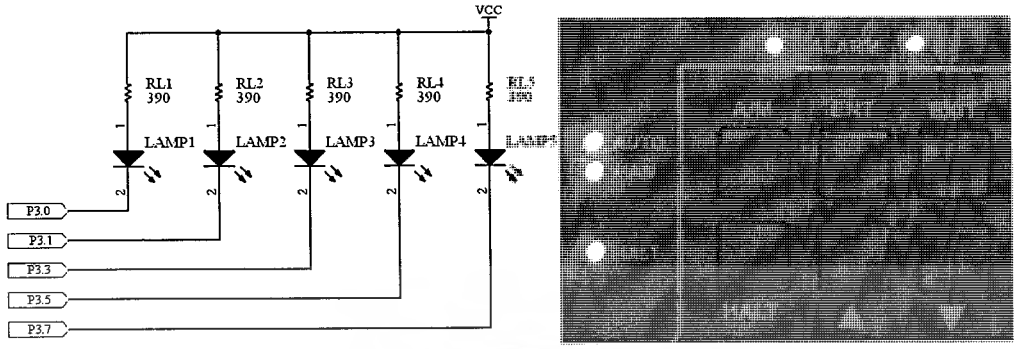
จากภาพที่ 4.9 Schematic ของการต่อจอแสดงผล LCD กับไมโครคอนโทรลเลอร์ ADuC 7024 ซึ่งจะเห็นได้ว่าเป็นการต่อแบบ I/O port และการทำงานของจอแสดงผล LCD อยู่ในโหมด 4 บิต ซึ่งจะต่อใช้งานที่ D7 – D4



ภาพที่ 4.10 การแสดงผลของ LCD บนตัวควบคุมPID

หน้าที่ของตัวแสดงผล LCD เพื่อใช้แสดงฟังก์ชั่นการทำงานของตัวควบคุมว่าในขณะนี้ตัวควบคุมอยู่ในฟังก์ชั่นการทำงานอะไร ซึ่งจะมีความสะดวกกว่าการแสดงชื่อฟังก์ชั่นบน 7-Segment ซึ่งจำเป็นต้องใช้คู่มือประกอบการใช้งานตัวควบคุมเสมอ

4.1.5.3 ตัวแสดงผล LED



ภาพที่ 4.11 Schematic และการแสดงผลของ LED

LED_H จะติดเพื่อเตือนว่า ขณะนี้ผลของ PV มากกว่าค่าของ SP มากเกินไปนั่นคือกระบวนการเกิดค่า error มากเกินไปนั่นเอง

LED_L จะติดเพื่อเตือนว่า ขณะนี้ผลของ PV น้อยกว่าค่าของ SP มากเกินไป

LED_AUTO จะติดเมื่อ ตัวควบคุมทำงานในโหมด Automatic

LED_MAN จะติดเมื่อ ตัวควบคุมทำงานในโหมด Manual

LED_HALT จะติดเมื่อ ตัวควบคุมมีการหยุดทำงานชั่วขณะ

4.2 การออกแบบ SoftWare

4.2.1 การออกแบบ อัลกอริทึม สำหรับตัวควบคุม PID ซึ่งไม่ต่อเนื่องทางเวลา

ก่อนที่จะกล่าวถึงการออกแบบ อัลกอริทึมของดิจิตอลคอนโทรลนั้น จะต้องพิจารณาว่าตัวควบคุมแบบไม่ต่อเนื่องมีคุณสมบัติเหมือนกันกับตัวควบคุมอนาลอก การควบคุมแบบอนาลอกได้อ้างถึงการออกแบบและ การทำให้เกิดผลของตัวควบคุมในโดเมนที่มีความต่อเนื่อง ซึ่งนั่นคือคอนโทรลเลอร์ที่สร้างจากอุปกรณ์อิเล็กทรอนิกส์นั่นเอง ถึงแม้ว่าธรรมชาติของความไม่ต่อเนื่องจะควบคุมโดยการจำลองการทำงานให้เหมือนกับธรรมชาติ ตามความต่อเนื่องของการควบคุมแบบอนาลอก ตัวอย่างซึ่งเป็น PI/PID อัลกอริทึมแบบอิเล็กทรอนิกส์จะมีขั้นตอนอยู่ด้วยกันหลายวิธี โดยคอนโทรลเลอร์นี้มีให้เห็นอยู่ทั่วไปล้วนแล้วถูกสร้างขึ้นมาจากรูปแบบที่ไม่ต่อเนื่องทั้งสิ้น

การออกแบบตัวควบคุม PI/PID จาก time domain

พิจารณาตัวควบคุม PID ในทางอุดมคติ เขียนเป็นรูปแบบโดเมนเวลาดังนี้

$$u(t) = K_c e(t) + \frac{K_c}{T_i}\int e(t)dt + K_c T_d \frac{de(t)}{dt} + u_0 \qquad (4.10)$$

สำหรับคอนโทรลเลอร์ที่ไม่ต่อเนื่อง ที่จำต้องมีการประมาณค่าในเทอมของการอินทิกรัล และderivative เพื่อให้เป็นรูปแบบที่เหมาะสำหรับการคำนวณโดยคอมพิวเตอร์ สามารถสรุปได้ว่า

$$\frac{de(t)}{dt} \approx \frac{e(t)-e(t-1)}{T_s} \tag{4.11}$$

และ

$$\int_0^t e(t)dt \approx T_s \sum_0^t e(i) \tag{4.12}$$

อัลกอริทึมของ PID ซึ่งไม่ต่อเนื่องของเวลาจะได้ว่า

$$u(t) = K_c e(t) + \frac{K_c T_s}{T_i} \sum_{i=0}^{t} e(i) + \frac{K_c T_d (e(t)+e(t-1))}{T_s} + u_0 \tag{4.13}$$

ฟอร์มที่ได้มานี้เป็นสมการความแตกต่าง ซึ่งเหมาะสำหรับการแปลรหัสในการประมาณค่าของการเขียนโปรแกรม ซึ่งเป็นที่รู้จักกันในชื่อ “positional” อันเนื่องมาจากสัญญาณควบคุมที่ได้จากการคำนวณอ้างอิงค่าจาก U_0

การออกแบบตัวควบคุม PI/PID จาก Laplace domain

เราสามารถกำหนดตัวควบคุม PID ซึ่งไม่ต่อเนื่องทางเวลาได้โดยตรงจากลาปลาซโดเมน อัลกอริทึมสามารถเขียนได้ดังนี้

$$\frac{U(s)}{E(s)} = K_c \left[1 + \frac{1}{T_i s} + T_d s \right] \tag{4.14}$$

จากนั้นเราก็สามารถใช้วิธี backward difference หรือ bilinear ในการสมมูลตัวควบคุม PID ยกตัวอย่างการประยุกต์ใช้วิธี backward difference เป็นดังนี้

$$\frac{U(s)}{E(s)} = K_c \left[1 + \frac{1}{T_i s} + T_d s \right] \xrightarrow{backward-difference} u(t) = e(t) K_c \left[1 + \frac{T_s}{T_i (1-z^{-1})} + T_d \frac{(1-Z^{-1})}{T_s} \right]$$

หรือเขียนให้เข้าใจได้ง่ายว่า

$$u(t) = u(t-1) + K_c \left[(e(t)-e(t-1)) + \frac{K_c T_s}{T_i} e(t) + \frac{K_c T_d}{T_s} \left[e(t) - 2e(t-1) + e(t-2) \right] \right]$$

อัลกอริทึมตัวควบคุม PID นี้มีโครงสร้างที่แตกต่างไปจากที่ได้รับจาก time domain เรียกอัลกอริทึมลักษณะนี้ว่า "Velocity" ซึ่งจะค่อนข้างแตกต่างกับการคอนโทรลที่ระบุค่าอ้างอิงคงที่ ที่ใช้ในอัลกอริทึม position การคำนวณตัวควบคุมในปัจจุบันใช้ค่าที่ควบคุมก่อนหน้าในการอ้างอิง การควบคุมจะถูกคำนวณขณะที่เกิดการเปลี่ยนแปลง เพราะฉะนั้นในเทอมของ "velocity form" ให้ผลลัพธ์ที่ได้จาก Bilinear Transform จะเหมือนกับอัลกอริทึมของ Velocity เนื่องจากการประพฤติตัวของ $(1-z^{-1})$ ที่ได้จากการประมาณค่า

อัลกอริทึม PID แบบ Positional และ Velocity

ถึงแม้ว่าโครงสร้างของอัลกอริทึมแบบ positional และ velocity จะมีความต่างกัน แต่ในความเป็นจริงแล้วมีความสัมพันธ์กัน ดังนี้

$$u(t) = K_c e(t) + \frac{K_c T_s}{T_i}\sum_{i=0}^{t} e(i) + \frac{T_d\left(e(t)-e(t-1)\right)}{T_s} + u_0 \tag{4.15}$$

เมื่อย้อนกลับไปหนึ่งครั้งของช่วงเวลาsampling จะได้ว่า

$$u(t-1) = K_c e(t-1) + \frac{K_c T_s}{T_i}\sum_{i=0}^{t-1} e(i) + \frac{T_d\left(e(t-1)-e(t-2)\right)}{T_s} + u_0 \tag{4.16}$$

นำทั้งสองสมการมาลบกัน จะได้ฟอร์มสุดท้ายคือ

$$u(t) = u(t-1) + K_c\left[e(t)-e(t-1)\right] + \frac{K_c T_s}{T_i}e(t) + \frac{K_c T_d}{T_s}\left[e(t)-2e(t-1)+e(t-2)\right] \tag{4.17}$$

หรือเขียนได้ดังนี้

$$U(kT) = U(kT-T) + K_C\left[e(kT)-e(kT-T)\right] + \frac{K_C T}{T_i}e(kT) + \frac{K_C T_d}{T}\left[e(kT)-2e(kT-T)+e(kT-2T)\right]$$

กำหนดให้ $a = K_C$

$$b = \frac{K_C T}{T_i}$$

$$c = \frac{K_C T_d}{T}$$

จะได้

$$U(kT) = U(kT-T) + a\left[e(kT)-e(kT-T)\right] + be(kT) + c\left[e(kT)-2e(kT-T)+e(kT-2T)\right]$$

สามารถเขียนสมการของแต่ละเทอมได้ดังนี้

p(kT) = ae(kT)-ae(kT-T)

i(kT) = be(kT)

d(kT) = ce(kT)-c2e(kT-T)+ce(kT-2T)

เราสามารถนำมาพัฒนาสร้างเป็นอัลกอริทึมเพื่อให้ได้ผลสำเร็จได้ดังนี้(โดยที่ตัวแปร a , b ,c กำหนดได้ตามสมการข้างต้น)

BEGIN

DO FOREVER

Get set point:	r(kT)
Get system output:	y(kT)
Calculate error:	e(kT) = r(kT) - y(kT)
Calculate P term:	p(kT) = ae(kT)-ae(kT-T)
Calculate I term:	i(kT) = be(kT)
Calculate D term:	d(kT) = ce(kT)-c2e(kT-T)+ce(kT-2T)
Calculate PID output:	u(kT) = p(kT)+i(kT)+d(kT)+u(kT-T)
Send control to actuator	
Save variables:	e(kT-2T) = e(kT-T)
	e(kT-T) = e(kT)
	u(kT-T) = u(kT)
Wait for next sample	

END DO

END

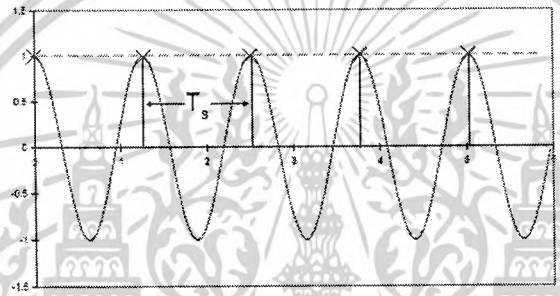
4.2.2 การเลือกช่วงเวลาในการสุ่มค่า (Choice of Sampling Interval)

ความสำคัญอื่นอีกในการสุ่มข้อมูลควบคุมจากระบบคือ การเลือกช่วงเวลาสำหรับการสุ่มค่าให้มีความเหมาะสม เพื่อให้สัญญาณที่คอนโทรลเลอร์ได้รับและส่งออกไปนั้นมีค่าใกล้เคียงกับตัวควบคุมที่มีความต่อเนื่องของเวลา อย่างง่ายดายคือช่วงเวลา sampling ต้องมีความเร็วที่มากพอ นั่นก็เพราะการประมาณค่าเพื่อใช้สร้างสมการความแตกต่าง ในการอธิบายคุณสมบัติตัวควบคุม หากช่วงเวลาในการ sampling น้อยก็จะให้คุณสมบัติของตัวควบคุมที่ได้ออกแบบนั้นบิดเบือนไปจากความเป็นจริง ดังนั้นเพื่อจะให้ได้เสถียรภาพที่ดีกว่าการสุ่มค่าที่ดีจะทำให้การทำงานของตัวควบคุมทำงานได้เต็มความสามารถ เมื่อช่วงเวลาในการsamplingมีค่าน้อยๆ

อย่างไรก็ตาม หากช่วงเวลาในการsampling เร็วมากจนเกินไป จะก่อให้เกิดความสิ้นเปลือง หรือทรัพยากรเสียหายได้ เช่น ต้นทุนจะเพิ่มมากขึ้นเมื่อต้องการองค์ประกอบที่ทำได้ดีมากขึ้น ,ความเร็วในการsamplingสูงจะหมายความว่าส่วนประกอบนั้นต้องมีความถี่สูงเช่นสัญญาณรบกวน ซึ่งจะมีผลครอบงำสัญญาณอีกด้วย อีกทั้งไม่ส่งให้เกิดประโยชน์อย่างจำเป็นสำหรับเสถียรภาพกับวงรอบของการควบคุม

ถ้าระยะเวลาในการ sampling นานเกินไป สัญญาณสูญเสียจะปรากฏ กรณีอย่างนี้จะเกิดปรากฏการณ์ที่เรียกว่า “aliasing”

ภาพประกอบด้านล่างแสดงให้เห็นเสถียรภาพระหว่างตัวควบคุมแบบ PID และ PYID ซึ่งมีช่วงเวลาของคลื่นเหมือนกัน(ค่าที่ได้จากการสุ่มคือตำแหน่งของเครื่องหมายดอกจัน)



ภาพที่ 4.12 แสดงให้เห็นเสถียรภาพระหว่างตัวควบคุมแบบ PID และ PYID

มีหลากหลายทฤษฎีซึ่งได้จากมาประสบการณ์สำหรับการเลือกช่วงเวลา sampling ซึ่งเป็นที่ยอมรับและได้จัดตีพิมพ์ รวมทั้งวิธีการที่จะกล่าวต่อไปด้วย

1. ถ้าเป็นระบบควบคุมลูปปิดซึ่งมีความถี่ธรรมชาติ (natural frequency) Wn ควรเลือกใช้ค่าความถี่ในการ sampling ของระบบเป็น

$$W_S > 10Wn$$

เมื่อ W_S คือความถี่ในการsampling $W_S = \frac{2\pi}{T}$

ซึ่งในการเลือกค่า เช่น เดียวกัน $T < T_S/10$ เมื่อ T_S คือ ช่วงเวลาเข้าสู่สภาวะคงที่ของระบบปิดใดๆ

2. ถ้าระบบมีค่า dominant time constant , T_{dtc} ดังนั้นช่วงเวลาsampling T หาได้ดังนี้

$$T < \frac{T_{dtc}}{10}$$

ในการออกแบบช่วงเวลาแซมปลิ้ง ได้เลือกช่วงเวลาเท่ากับ 250 msec ซึ่งในการกำหนดให้ไมโครคอนโทรลเลอร์แซมปลิ้งทุกๆ 250 ms นั้น จะต้องกำหนดการทำงานในส่วนของ Interrupt ซึ่งเลือกใช้ Timer1 ทำงานในโหมด periodic (okresowy)

ในกรณีนี้ Timer จะเริ่มนับจากค่าสูงสุดใน Load Register (TxLD) ลงมาจนถึงค่าต่ำสุด (timer count-down) จากนั้นจะเริ่มนับจากค่าต่ำสุดใน Load Register (TxLD) ไปจนถึงค่าสูงสุด (timer count-up) สำหรับโหมด Periodic จะทำงานตามสมการ

$$Interval = \frac{(TxLD)\,\mathrm{Pr}\,escaler}{SourceClock} \tag{4.18}$$

หรือ

$$TxLD = \frac{Interval(SourceClock)}{\mathrm{Pr}\,escaler}$$ เมื่อ Interval = 250 ms

SourceClock = HCLK = 41.78 MHz

Presaler = / 1

แทนค่าในสมการจะได้

$$TxLD = \frac{250ms(41.78MHz)}{1} = 10445042 \text{ Cycle}$$

กำหนด T1LD = 10445042 ; ในโปรแกรมก็จะอินเตอร์รัพธ์ ทุกๆ 250 ms

4.2.3 การแปลงสัญญาณอนาลอกเป็นดิจิตอล

จากที่กล่าวไว้แล้วข้างต้นว่า การแปลงสัญญาณอนาลอกเป็นดิจิตอล มีอยู่ด้วยกัน 3 โหมด คือ Single-ended mode, Differential mode และ Pseudo differential mode ซึ่งแต่ละโหมดจะมีลักษณะการทำงานที่แตกต่างกัน ระดับแรงดันที่ใช้มีค่าต่างกัน จึงต้องเลือกการทำงานให้เหมาะสมกับลักษณะของสัญญาณ

ในการออกแบบนั้น ได้เลือกใช้งาน Single-Ended mode เมื่อไมโครคอนโทรลเลอร์ของเราทำงานอยู่ในโหมดนี้ ย่านการทำงานที่สามารถรับอินพุตจากภายนอกจะอยู่ในช่วง 0V ถึง VREF ซึ่งได้เลือกค่า VREF ที่ 2.5V ดังนั้นถ้าแปลงให้เป็นรหัสเลขฐานสองทำได้โดย

- 1 LSB = FS/4096 หรือ
- 2.5 V/4096 = 0.61 mV หรือ

- 610 uV when VREF = 2.5 V

สำหรับการเขียนโปรแกรมเพื่อให้สามารถรับสัญญาณอนาลอกได้นั้นจำเป็นต้องอ่านคู่มือของ MCU เบอร์นั้นๆ เพราะเนื่องจากว่า MCU แต่ละตัวก็จะมีลักษณะการเข้าถึงที่ต่างกันเพราะฉะนั้นการอ่านคู่มือของ MCU จึงมีความสำคัญมาก หน้าที่ของ Register แต่ละตัวได้กล่าวไว้แล้วข้างต้น การกำหนดค่าเริ่มต้นเพื่อให้ MCU ทำหน้าที่แปลงสัญญาณอนาลอกเป็นดิจิตอลทำได้ดังนี้

ส่วนของการกำหนดการทำงาน ADC

```
ADCCON  = 0x00000000;                   // Reset ADC Config
ADCCON |= 0x00000020;                   // Power-ON ADC Function
delay(1000);                            // Wait ADC Power-on Ready
ADCCON |= 0x00001400;                   // ADC Clock = fADC/32
ADCCON |= 0x00000300;                   // Acquisition Time = 16 Cycle Clock
ADCCON &= 0xFFFFFFE7;                   // ADC = Single-End Mode
ADCCON |= 0x00000004;                   // Continue Software Convert
REFCON  = 0x00000001;                   // Used Internal 2.5V Reference
ADCCON |=   0x00000080;                 // ADC Start Conversion
```

ส่วนของการแปลงอนาลอกเป็นดิจิตอลของ MCU

```
ADCCP = 0;                              // Select Channel 0 to Conversion
delay(1000);                            // Wait Select Channel Ready
while (!ADCSTA){};                      // Wait ADC Conversion Complete (Bit0="1")
val = (ADCDAT >> 16)& 0x00000FFF;       // Shift ADC Result to Integer
```

ดังนั้นผลที่ได้จากการแปลงสัญญาณหลังจากการ Shift ADC Result คือ

เมื่ออินพุต 0V ให้สัญญาณที่เป็นดิจิตอลมีค่า 0x000 H

เมื่ออินพุต 2.5V ให้สัญญาณดิจิตอลมีค่า 0xFFF H

สัญญาณดิจิตอลที่ได้นี้ จะถูกนำไปใช้ในการประมวลผล ซึ่งค่านี้คือค่าของ PV หรือค่าที่ได้จากกระบวนการนั่นเอง

4.2.4 การแปลงสัญญาณดิจิตอลเป็นอนาลอก

สัญญาณที่ได้จากคอมพิวเตอร์ หรือไมโครคอนโทรลเลอร์จะเป็นสัญญาณแบบดิจิตอล (แบบไม่ต่อเนื่อง) เมื่อนำระบบดิจิตอลหรือไมโครคอนโทรลเลอร์มาใช้ควบคุมอุปกรณ์อนาลอกแล้ว จะต้องมีวงจรแปลงสัญญาณทางดิจิตอลเป็นสัญญาณอนาลอก ตั้งแต่ศูนย์โวลท์จนถึงระดับสูงสุดที่กำหนดไว้

สำหรับ DAC ในไมโครคอนโทรลเลอร์ ARM7 เบอร์ ADuC7024 นั้นจะมีขนาด 12-bit 2 Channel มีย่านการใช้งาน 2 ย่าน คือ 0V-VREF และ 0V-AVDD ในการออกแบบโปรเจคนี้ใช้ย่าน 0V-VREF และ Channel 0 ซึ่งกำหนดค่าเริ่มต้นการทำงานได้ดังนี้

```
// การกำหนดค่าเริ่มต้นการทำงาน DAC
DAC0CON &= 0xDF;                    // DAC0 Used System Clock
DAC0CON |= 0x10;                    // Enable DAC0
DAC0CON |= 0x02;                    // DAC0 Output Range = +Vref..AGND
REFCON   = 0x01;                    // Used Internal 2.5V Reference
// การส่งค่าข้อมูลออกสู่ DAC Channel 0
DAC0DAT = (MV << 16);               // Update DAC0 Output(0-2.5V)
```

4.2.5 การ Scaling

ในการสร้างอัลกอริทึมเพื่อคำนวณสมการควบคุม PID นั้น จำต้องตระหนักถึงการ Scaling ของค่าตัวแปรต่างๆเป็นสำคัญ เพื่อให้อยู่ในหน่วยการคำนวณเดียวกัน ในการออกแบบโปรเจคนี้ ได้ทำการScaling ให้อยู่ในหน่วยของเปอร์เซ็นต์(0-100%) ทำได้ดังนี้

```
// การ Scaling ตัวแปร Sp
    Sp กำหนดให้ปรับค่าได้จาก 0 ไปจนถึง 100
// การScaling ตัวแปร PV
    val = (ADCDAT >> 16)& 0x00000FFF;     // Shift ADC Result to Integer
    pv_volt = val * (2.50 / 4095.0);      // Volt = ADC Result x [2.5V / 4095]
    pv_percent=(pv_volt*100)/2.5  ;       // Convert to percent
    pv=pv_percent;
// การ Scaling ตัวแปร MV
    mv=U_1+Up+Ui+Ud;                      //calculate PID output
    U=(mv*4095)/100;
    if(U>4095)      { mv=100; }           //condition
    if(U<0)         { mv=0;   }
```

บทที่ 5

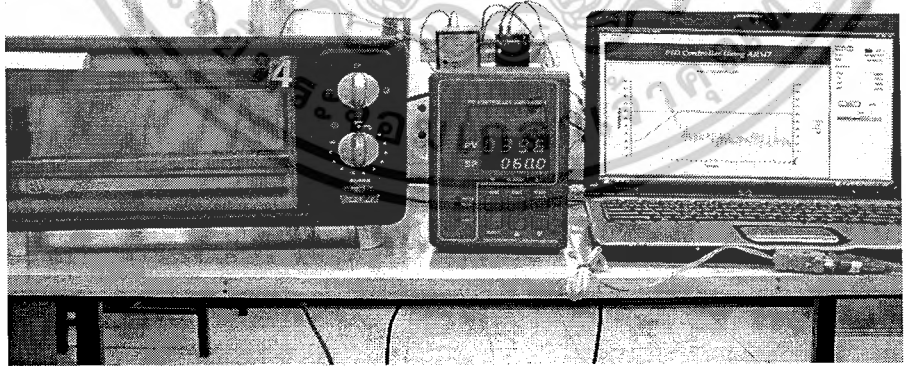
การทดลองและผลการทดลอง

ในบทนี้จะนำเอาตัวควบคุม PID ที่ได้ออกแบบไว้บนไมโครคอนโทรลเลอร์ ARM7 นำไปทดลองเพื่อวิเคราะห์ผลตอบสนอง ของกริยาควบคุมต่างๆ จากนั้นใช้วิธี Process Reaction Curve ของ Ziegler-NicholsและวิธีT rial&Errorในการหาค่าพารามิเตอร์ K_P , T_i และ T_d ที่มีค่าเหมาะสม ซึ่งใช้เตาอบเป็นกระบวนการที่ใช้ในการทดสอบ

ในการทดลองจะกำหนด setpoint (SP) ไปที่ค่า 60% เพื่อพิจารณากริยาควบคุมต่าง ๆ จากนั้นเปลี่ยนค่าเป้าหมาย setpoint จาก 60% ไปเป็น 80% เพื่อให้ดูผลตอบสนองของกระบวนการเมื่อค่าเป้าหมายมีการเปลี่ยนแปลง หลังจากนั้นสร้างสัญญาณรบกวน (disturbance) โดยการเปิดฝาของเตาอบทิ้งไว้เป็นเวลา 10 วินาที แล้วบันทึกผลการทดลอง

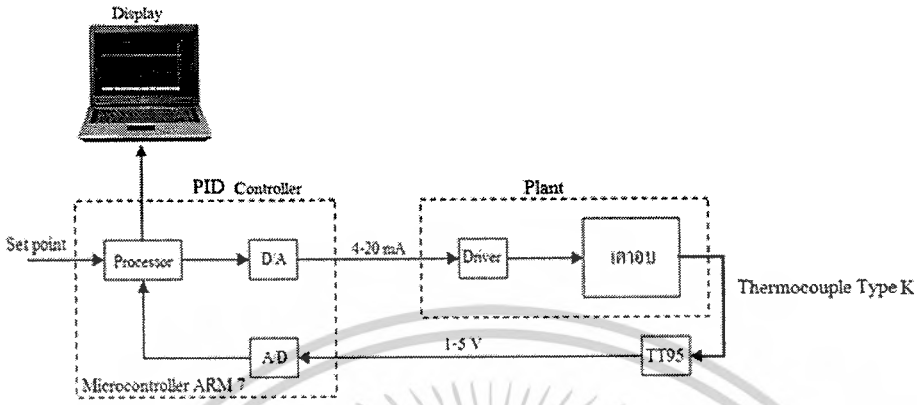
กราฟที่ได้จากการทดลองนั้นจะนำมาหาค่าพารามิเตอร์ต่าง ๆ เช่น Maximum Overshoot ($\%M_P$) , Rise time(t_r) และ Settling time(t_s) เพื่อเปรียบเทียบค่าของผลตอบสนองที่ได้จากค่าพารามิเตอร์ในแต่ละครั้ง

5.1 กระบวนการที่ใช้ในการทดสอบตัวควบคุม PID



ภาพที่ 5.1 แสดงกระบวนการเตาอบไฟฟ้า

การเชื่อมต่ออุปกรณ์ต่างๆและโครงสร้างไดอะแกรมของกระบวนการ ที่ได้กล่าวไว้ข้างต้น เมื่อนำมาออกแบบจะได้กระบวนการสำหรับควบคุมอุณหภูมิ ดังแสดงในภาพที่ 5.2



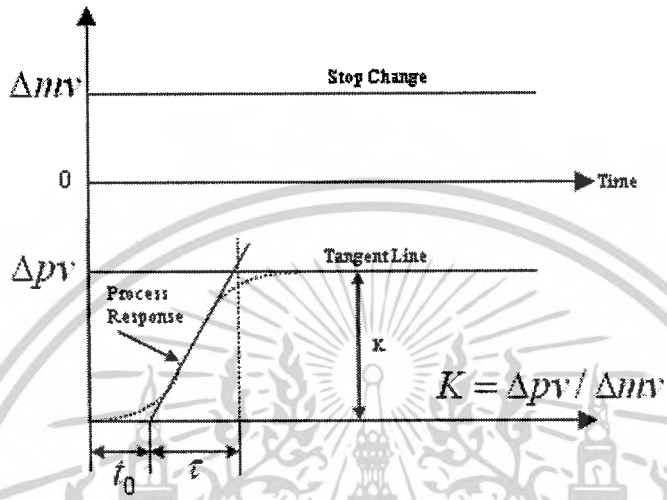
ภาพที่ 5.2 แสดงลักษณะการออกแบบกระบวนการ

การทำงานของกระบวนการนั้น แบ่งออกเป็น 2 ส่วน คือ ส่วนที่เป็นส่วนควบคุม กับส่วนที่เป็นกระบวนการเตาอบ การทำงานของกระบวนการเริ่มจากการกำหนดค่าเป้าหมาย (set point) ซึ่งกำหนดจากตัวควบคุมได้โดยตรง จากนั้นไมโครคอนโทรลเลอร์จะประมวลผลเพื่อสร้างสัญญาณควบคุมจากอัลกอริทึมที่ได้ออกแบบไว้ ซึ่งสัญญาณที่ออกจากตัวควบคุมจะเป็นสัญญาณมาตรฐาน 4-20mA ส่งผ่านสัญญาณไปยัง Final Control Element (heater) เพื่อจ่ายพลังงานความร้อนให้กับกระบวนการเตาอบโดยการทำงานของตัวทำความร้อนจะถูกควบคุมโดย Solid State Relay ซึ่งจะสร้างสัญญาณพัลส์ควบคุมตามสัญญาณที่ได้รับ (4-20 mA) อุณหภูมิของกระบวนการเตาอบจึงเกิดการเปลี่ยนแปลงขึ้น เทอร์โมคัปเปิ้ลจะทำการตรวจจับค่าความร้อนมาจากเตาอบออกมาในรูปแบบของแรงดันไฟฟ้า (mV) เนื่องจากแรงดันที่ได้ไม่เหมาะสมกับการนำไปใช้งานจึงต้องนำอุปกรณ์ Thermocouple Transmitter 95 (TT95) ซึ่งเป็นตัวแปลงสัญญาณที่รับสัญญาณจากเทอร์โมคัปเปิ้ลเพื่อแปลงให้เป็นสัญญาณมาตรฐาน 1-5V จากนั้นจึงนำสัญญาณที่ได้ส่งต่อไปยังตัวควบคุม สัญญาณจะถูกส่งเข้าไปยังส่วนของ A/D ในตัวไมโครคอนโทรลเลอร์ ARM7 เพื่อแปลงสัญญาณที่ได้รับซึ่งเป็นสัญญาณอนาลอกให้อยู่ในรูปของสัญญาณดิจิตอล และนำไปประมวลผลต่อไป

5.2 การทดลองตัวควบคุม PID โดยวิธี Process Reation Curve ของ (Ziegler-Nichols)

ปรับตัวควบคุมให้อยู่ในตำแหน่ง Manual Mode เพื่อทำลูปเปิด จากนั้นทำการปรับค่า Mv ไว้ที่ 20% รอจนกว่าเข้าสู่ steady state จากนั้นเปลี่ยนค่า Mv เป็น 60% สังเกตค่า PV จากกราฟว่าผลเป็น S Curve ดังภาพที่ 5.3หรือไม่ ถ้าได้ก็ทำการลากเส้นสัมผัส Tangent Line แล้วทำการหาค่า

อัตราขยายของกระบวนการ K, ค่าเวลาคงที่ของกระบวนการ τ (Time Constant) และค่า t_0(Deat Time) จากนั้นเลือกวิธีการควบคุมรูปแบบ PID โดยวิธี Process Reaction Curve ของ Ziegler-Nichols จะได้ค่าของ K_P, T_i และ T_d ซึ่งใช้ในการปรับค่าพารามิเตอร์ของตัวควบคุม PID ตามวิธี Process Reaction Curve ดังแสดงในภาพที่ 5.3



ภาพที่ 5.3 แสดงผลตอบสนองรูปตัวS เมื่อใช้วิธี Process Reaction Curve

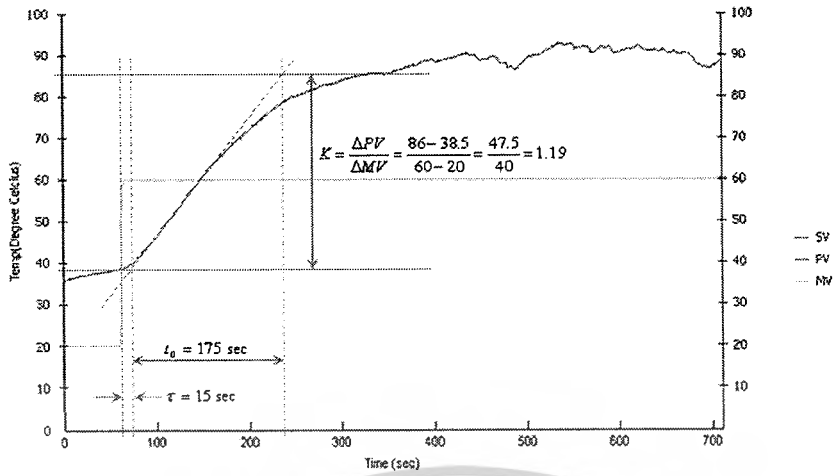
จากภาพที่ 5.3 จะได้ Transfer Function

$$G(s) = \frac{Ke^{-t_0 s}}{\tau s + 1}$$

K คือ อัตราขยายของกระบวนการ ซึ่งมีค่า $K = \frac{\Delta PV}{\Delta MV}$

t_0 คือ ค่าเวลาหน่วงของกระบวนการ

τ คือ ค่าเวลาคงที่ของกระบวนการ



ภาพที่ 5.4 แสดงผลตอบสนองรูปตัว S จากการทดลองลูปเปิดที่ Mv =20% ไปเป็น 60%

จากกราฟ สามารถจำลอง Transfer Function ของกระบวนการเตาอบ ได้ดังนี้

$$K = \frac{\Delta PV}{\Delta MV} = \frac{86-38.5}{60-20} = \frac{47.5}{40} = 1.19 \quad , \quad t_0 = 15 \text{ sec} \quad \tau = 175 \text{ sec}$$

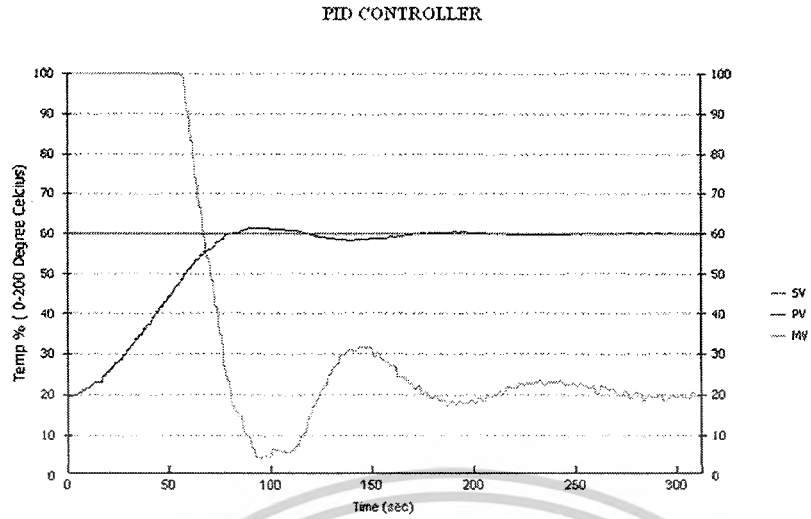
$$G(s) = \frac{1.19e^{-15s}}{175s+1}$$

นำค่าพารามิเตอร์ต่างๆ ที่ได้จากวิธี Process Reaction Curve มาคำนวณตามทฤษฎีของ Ziegler-Nichols ได้ค่าพารามิเตอร์ของตัวควบคุมคือ

พารามิเตอร์สำหรับ ตัวควบคุมแบบ PI คือ

$$K_P = \frac{0.9}{K}\left(\frac{\tau}{t_0}\right) = \frac{0.9(175)}{1.19(15)} = 8.8$$

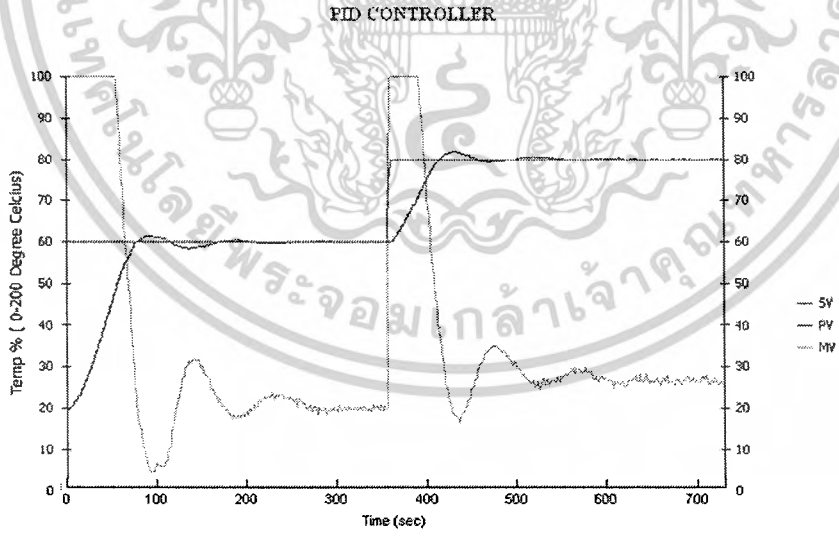
$$T_i = 3.33t_0 = 3.33(15) = 50$$



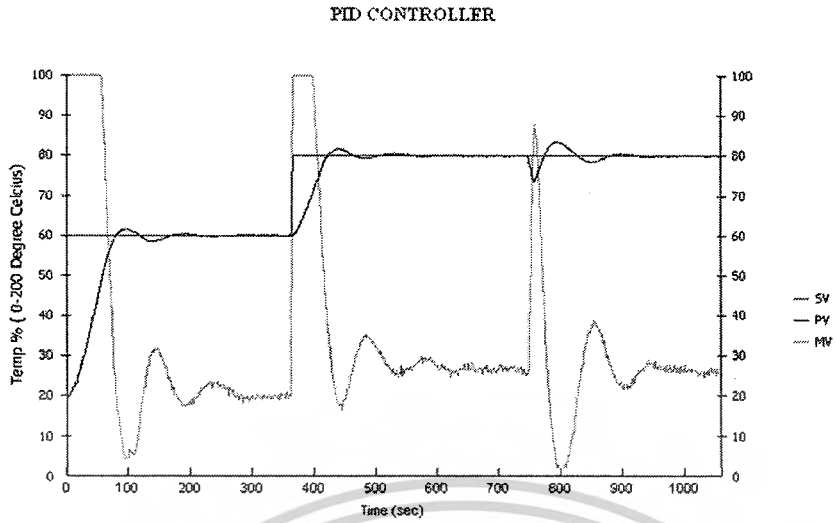
ภาพที่ 5.5 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) Setpoint = 60%

มีผลตอบสนองของกระบวนการเป็นดังนี้

- Percent Maximum Overshoot: $M_p = 3$ %
- Settling Time: $t_s = 180$ sec
- Rise Time: $t_r = 53$ sec



ภาพที่ 5.6 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols)
เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%



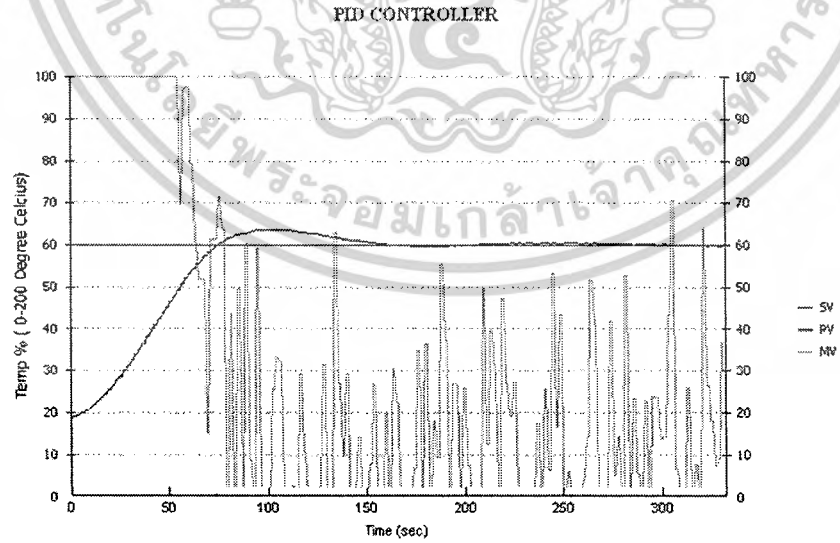
ภาพที่ 5.7 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance

พารามิเตอร์สำหรับ ตัวควบคุมแบบ PID

$$K_P = \frac{1.2}{K}\left(\frac{\tau}{t_0}\right) = \frac{1.2(175)}{1.19(15)} = 11.7$$

$$T_i = 2.0t_0 = 2.0(15) = 30$$

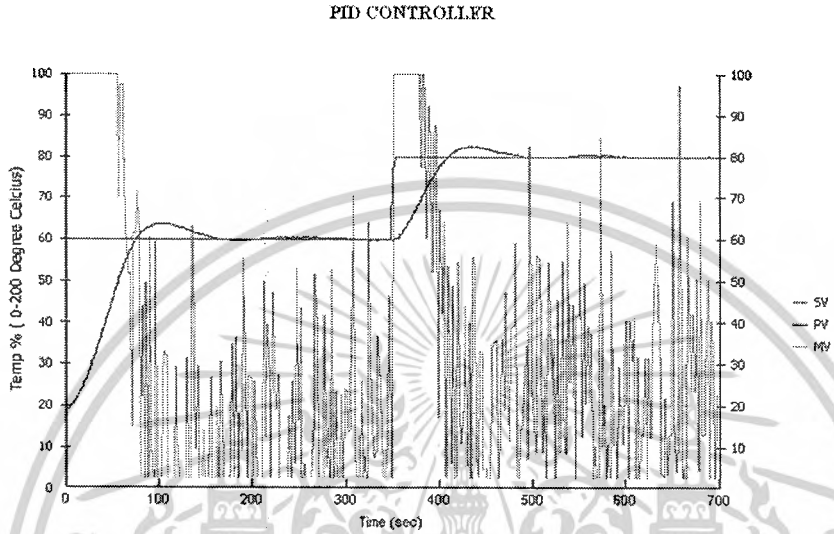
$$T_d = 0.5t_0 = 0.5(15) = 7.5$$



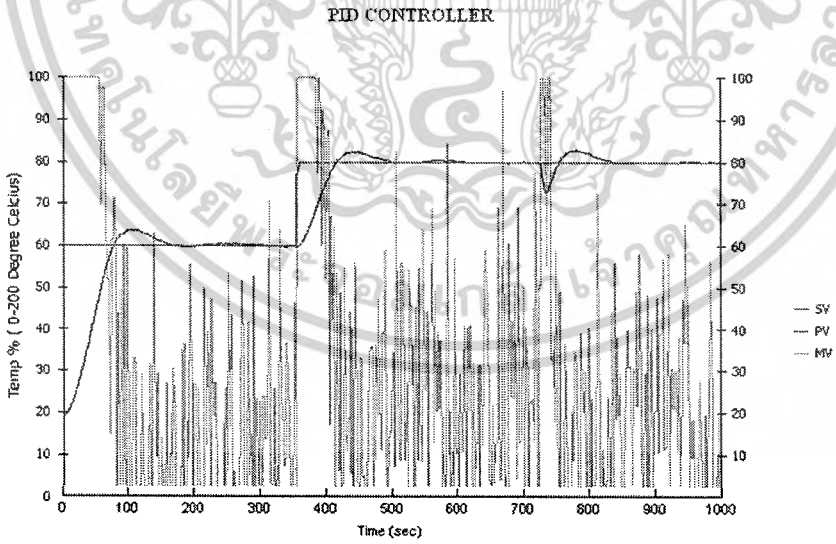
ภาพที่ 5.8 แสดงผลตอบสนองของตัวควบคุม PID (Ziegler-Nichols) Setpoint = 60%

ผลตอบสนองของกระบวนการเป็นดังนี้

- Percent Maximum Overshoot: $M_p = 5.83$ %
- Settling Time: $t_s = 160$ sec
- Rise Time: $t_r = 48$ sec



ภาพที่ 5.9 แสดงผลตอบสนองของตัวควบคุม PID (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%

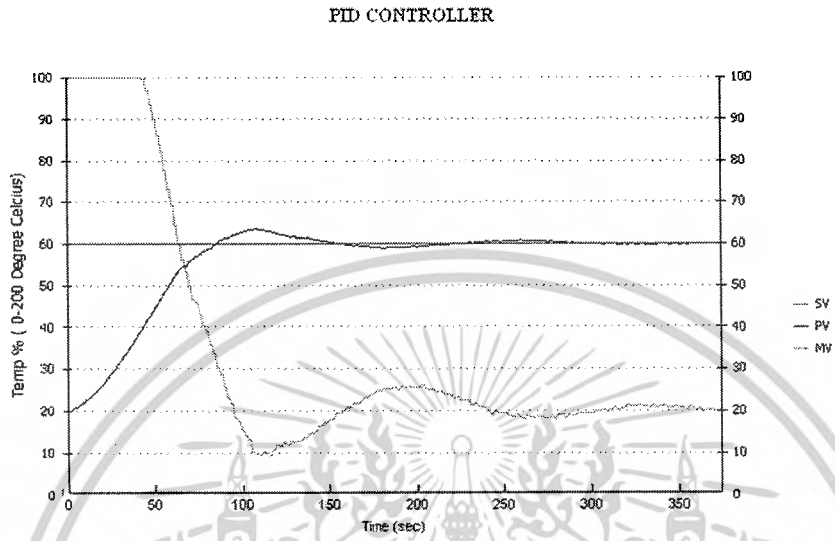


ภาพที่ 5.10 แสดงผลตอบสนองของตัวควบคุม PI (Ziegler-Nichols) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance

5.3 การทดลองตัวควบคุม PID โดยวิธี Trial&Error

กำหนดค่าพารามิเตอร์สำหรับตัวควบคุมแบบ PI ดังนี้

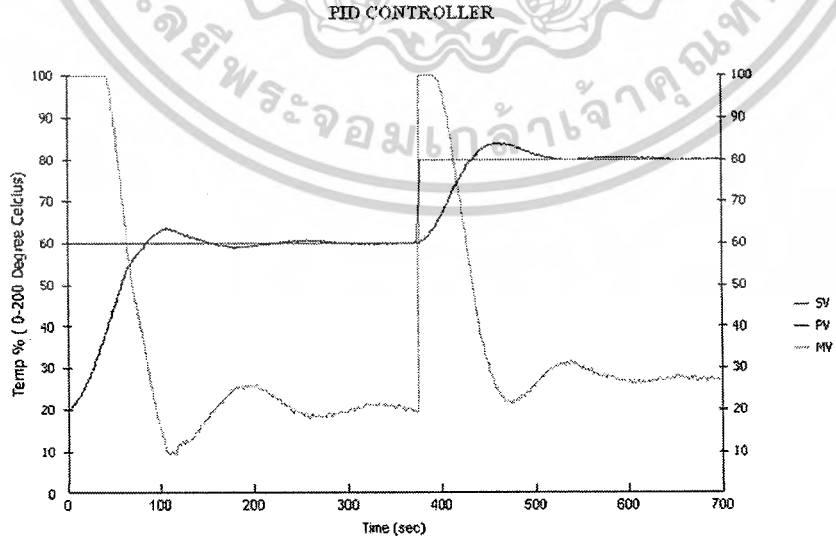
$K_P = 5$, $T_i = 50$ และ $T_d = 0$



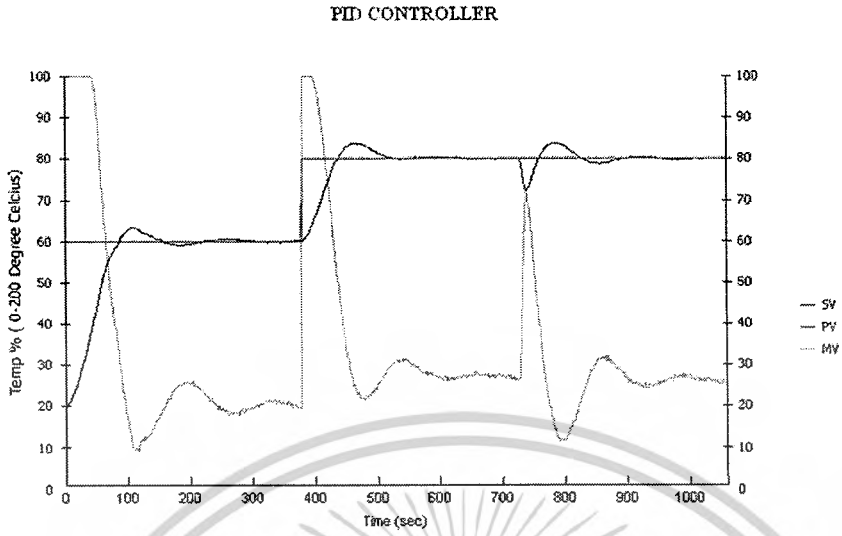
ภาพที่ 5.11 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error) Setpoint = 60%

ผลตอบสนองของกระบวนการเป็นดังนี้

- Percent Maximum Overshoot: $M_p = 5.85$ %
- Settling Time: $t_s = 228$ sec
- Rise Time: $t_r = 53$ sec



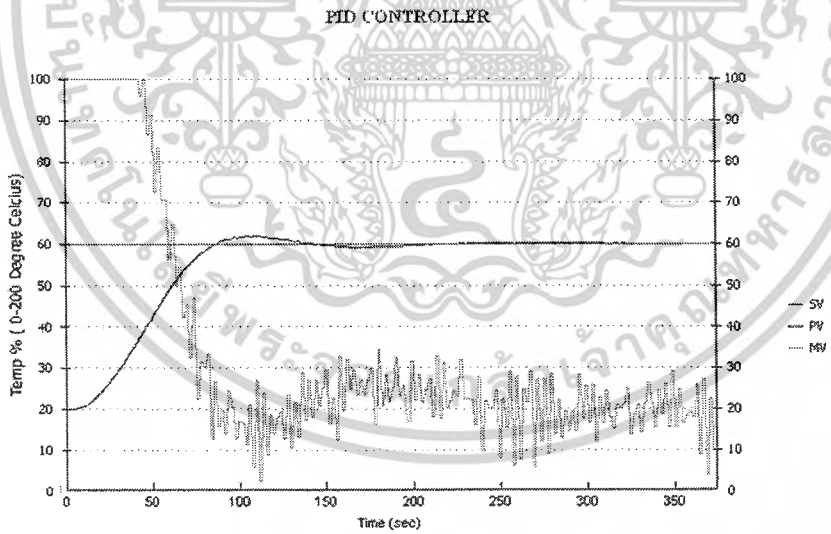
ภาพที่ 5.12 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error)
เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%



ภาพที่ 5.13 แสดงผลตอบสนองของตัวควบคุม PI (Trial & Error) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance

กำหนดค่าพารามิเตอร์สำหรับตัวควบคุมแบบ PID ดังนี้

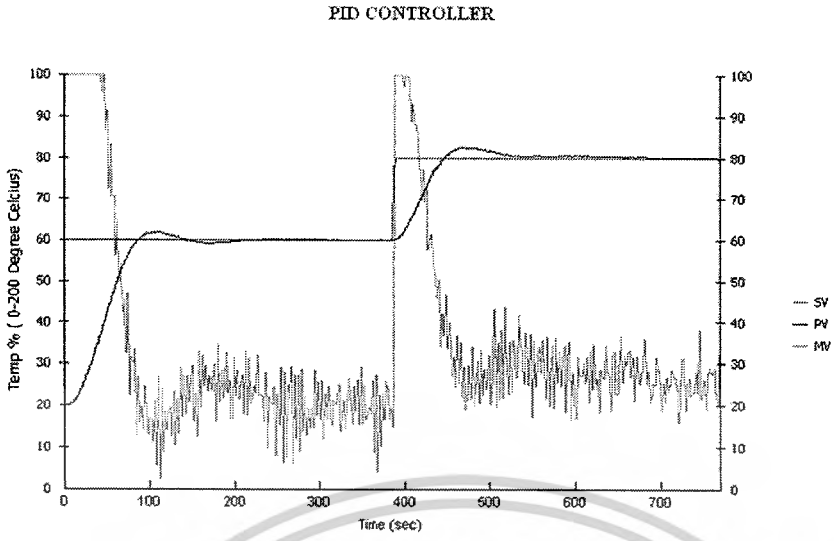
$K_P = 5$, $T_i = 78$ และ $T_d = 3$



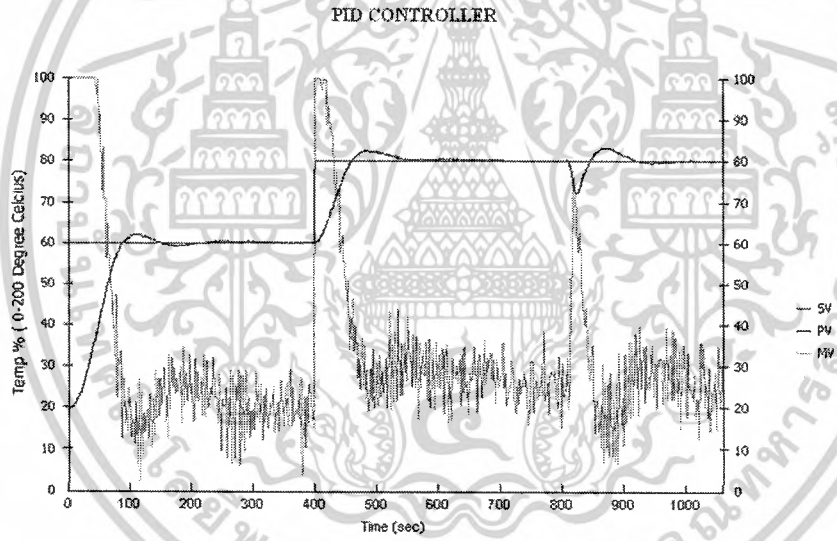
ภาพที่ 5.14 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error) Setpoint = 60%

ผลตอบสนองของกระบวนการเป็นดังนี้

- Percent Maximum Overshoot: $M_p = 5$ %
- Settling Time: $t_s = 150$ sec
- Rise Time: $t_r = 53$ sec



ภาพที่ 5.15 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error) เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80%



ภาพที่ 5.16 แสดงผลตอบสนองของตัวควบคุม PID (Trial & Error)เมื่อเปลี่ยน Setpoint จาก 60% เป็น 80% และเกิด disturbance

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการทดลอง

จากการทดลองผลตอบสนองของตัวควบคุม ซึ่งใช้วิธีการหาค่าพารามิเตอร์ของตัวควบคุม 2 วิธีดังนี้ วิธีของ Ziegler-Nichols และวิธี Trial & Error ทั้ง 2 วิธีให้ผลตอบสนองที่แตกต่างกันดังนั้นจะสรุปแยกกันในแต่ละวิธีดังนี้

วิธีของ Ziegler-Nichols ซึ่งหลังจากที่หาทรานซ์เฟอร์ฟังก์ชั่นของกระบวนการโดยทดสอบระบบเปิดเพื่อคำนวณค่า K_c, T_i และ T_d ได้แล้วนั้น ให้ผลตอบสนองของระบบที่ควบคุมแบบ PI ให้ผลตอบสนองเป็นที่น่าพอใจและเมื่อควบคุมแบบ PID ผลตอบสนองของระบบจะเร็วขึ้นเมื่อทำการเปลี่ยนแปลงค่าเป้าหมายจาก 60% เป็น 80% ตัวควบคุมก็ยังคงทำงานได้เป็นอย่างมีประสิทธิภาพ แม้ไม่มีการเปลี่ยนแปลงค่าพารามิเตอร์ก็ตามเมื่อเกิดสัญญาณรบกวน (disturbance) โดยการเปิดฝาเตาอบค้างไว้เป็นเวลา 10 วินาที ตัวควบคุมก็สามารถควบคุมกระบวนการให้เป็นดังเดิมได้โดยเร็ว อย่างไรก็ตามทั้งการควบคุมแบบ PI และ PID หากต้องการให้ผลตอบสนองดียิ่งขึ้นนั้นจะต้องทำการปรับค่าของพารามิเตอร์ให้ต่างไปจากที่คำนวณได้เล็กน้อย โดยพิจารณาการเปลี่ยนแปลงผลของตอบสนองพร้อมทั้งปรับค่า

ส่วนวิธี Trial & Error นั้น ได้ผลเป็นที่น่าพอใจเพียงแต่จะต้องใช้เวลาในการปรับแต่งนานกว่าวิธีอื่น เนื่องจากต้องพิจารณาพฤติกรรมของพารามิเตอร์แต่ละตัว

สรุปได้ว่าทุก ๆ วิธีในการหาค่าพารามิเตอร์ ค่าที่ได้นั้นยังไม่ใช่ค่าที่ดีที่สุดสำหรับการใช้ควบคุมหากแต่เป็นเพียงค่าประมาณ หากต้องการให้ผลตอบสนองของระบบมีค่าดีขึ้นผู้ควบคุมจะต้องปรับแต่งค่าที่ได้อีกครั้งหนึ่งอย่างละเอียด และการควบคุมกระบวนการใดๆ ตัวควบคุมแบบ PID อาจจะไม่ให้ผลที่ดีที่สุด หากแต่ต้องคำนึงถึงความเหมาะสมของกระบวนการนั้นๆด้วย

6.2 ปัญหาและอุปสรรค์

1. ระบบมีการระบายความร้อนที่น้อยทำให้อุณหภูมิของเตาอบลดลงช้า จึงควรมีการติดตั้งอุปกรณ์ระบายความร้อนเพิ่มขึ้น เนื่องจากการระบายความร้อนของระบบขึ้นอยู่กับปริมาณน้ำที่ใส่ในเตาอบ เมื่ออุณหภูมิเพิ่มขึ้นจึงทำให้น้ำในระบบระเหยระบบจึงระบายความร้อนได้น้อยลงจึงควรมีการรักษาระดับน้ำให้คงที่

2. การปรับค่า span และ zero ของ Thermocouple Transmitter จะต้องทำให้อุณหภูมิคงที่ซึ่งค่าใดค่าหนึ่งก่อนจึงทำการวัดได้

3. ขณะที่ฮีตเตอร์อยู่ในภาวะปกติ เมื่อได้ทำการจ่ายอินพุตเข้าไป ฮีตเตอร์จะต้องใช้เวลาในการทำความร้อนซึ่งไม่สามารถทำงานได้ในทันที

4. ส่วนของฮาร์ดแวร์ จะพบปัญหาความเสถียรของค่าต่างๆในวงจร เช่น แรงดันไฟฟ้าและกระแสไฟฟ้า ทำให้ต้องมีการปรับแต่งอยู่เสมอ และบางครั้งเกิดการผิดพลาดบ้างแต่ยังคงอยู่ในช่วงที่สามารถยอมรับได้

6.3 ข้อเสนอแนะ

การออกแบบตัวควบคุมนั้น จำเป็นต้องคำนึงถึงกระบวนการที่จะนำไปควบคุมเพื่อจะได้กำหนดช่วงเวลา sampling ได้อย่างเหมาะสม อีกทั้งในการหาค่าพารามิเตอร์ K_p,T_i และ T_dก่อนจะนำมาใช้ควบคุมกระบวนการ ผู้ควบคุมจำต้องเข้าใจทฤษฎีในการคำนวณและวิธีการปรับแต่งเสียก่อน

ตัวควบคุมนี้สามารถที่จะนำไปประยุกต์ใช้กับกระบวนการอื่น ๆ ได้ เพียงแต่ต้องทำการแก้ไขโปรแกรมอีกเล็กน้อย เพื่อให้เหมาะสมกับในแต่ละกระบวนการ

บรรณานุกรม

[1] รศ.ประสิทธิ์ จุลเสรีวงศ์. วิศวกรรมการวัดคุม. กรุงเทพฯ : แผนกตำรา คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2549

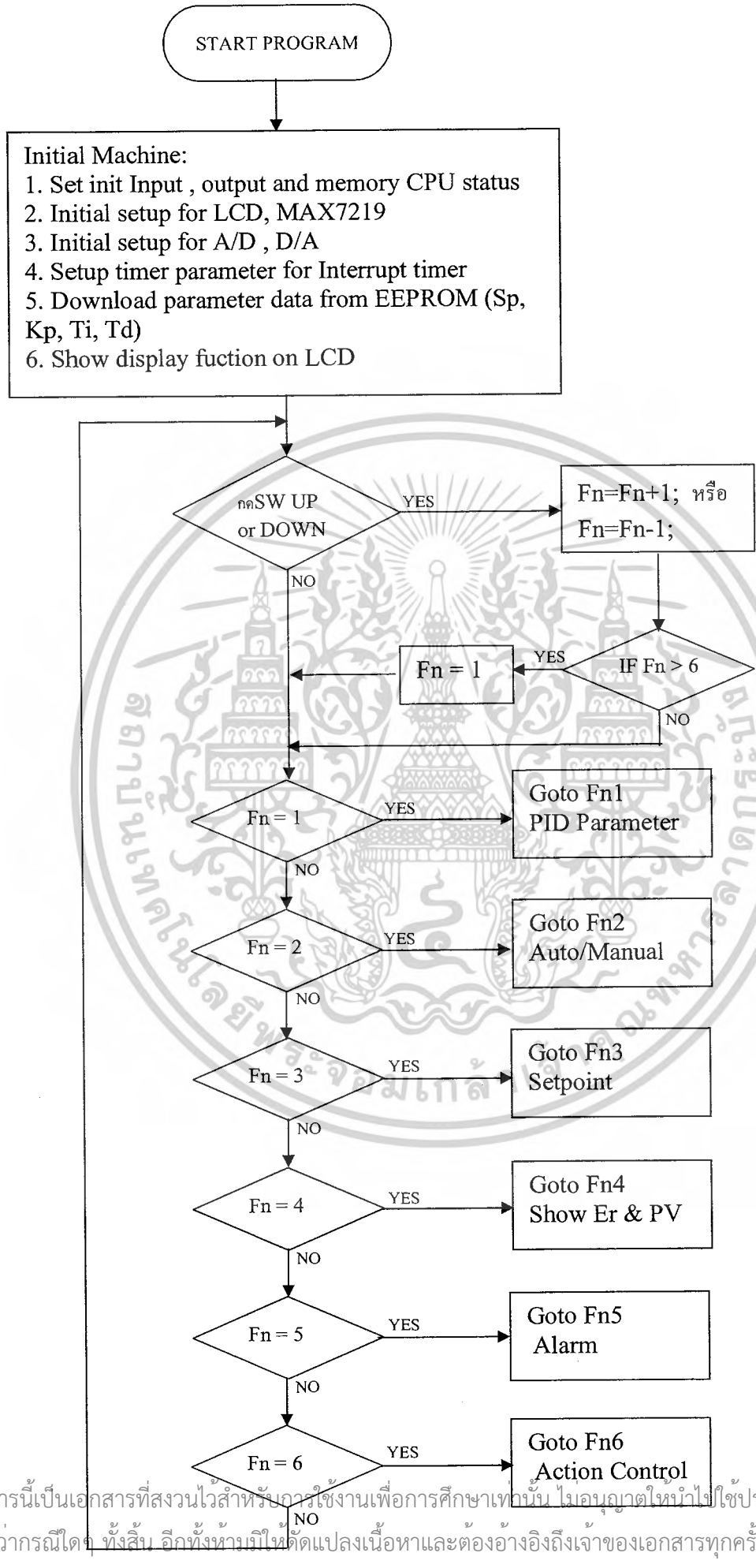
[2] ทวิช ชูเมือง.Industrial Instrumentation Engineering and Design.กรุงเทพฯ:เอช.เอ็น กรุ๊ป,2549

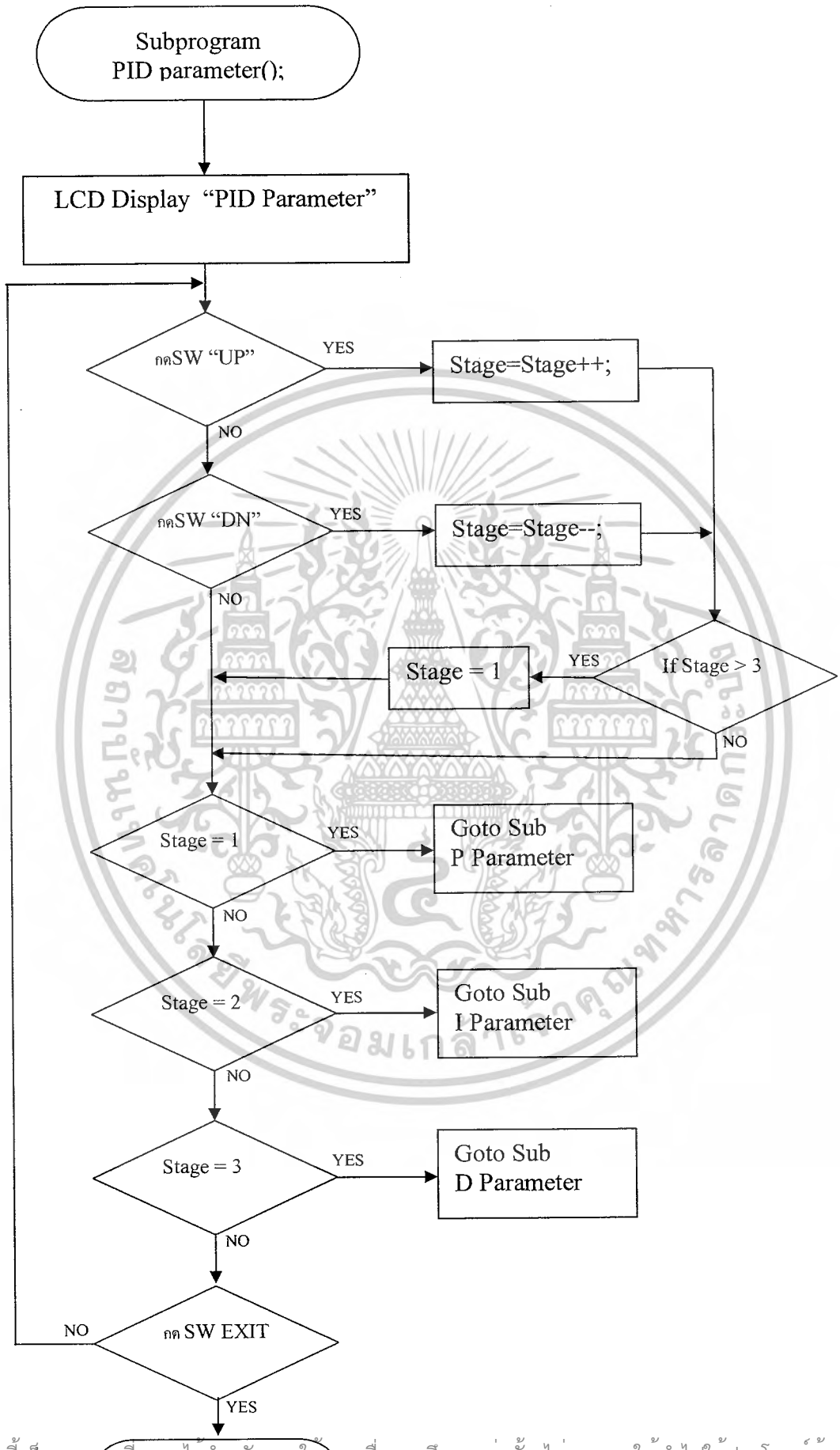
[3] ผศ.วิศรุต ศรีรัตนะ, ผศ.อัมพวัน ใจกล้า, อ.พิทยา ปานนิล . ปฏิบัติการวิศวกรรมการวัดคุม 1. กรุงเทพฯ:แผนกตำรา คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง, 2548

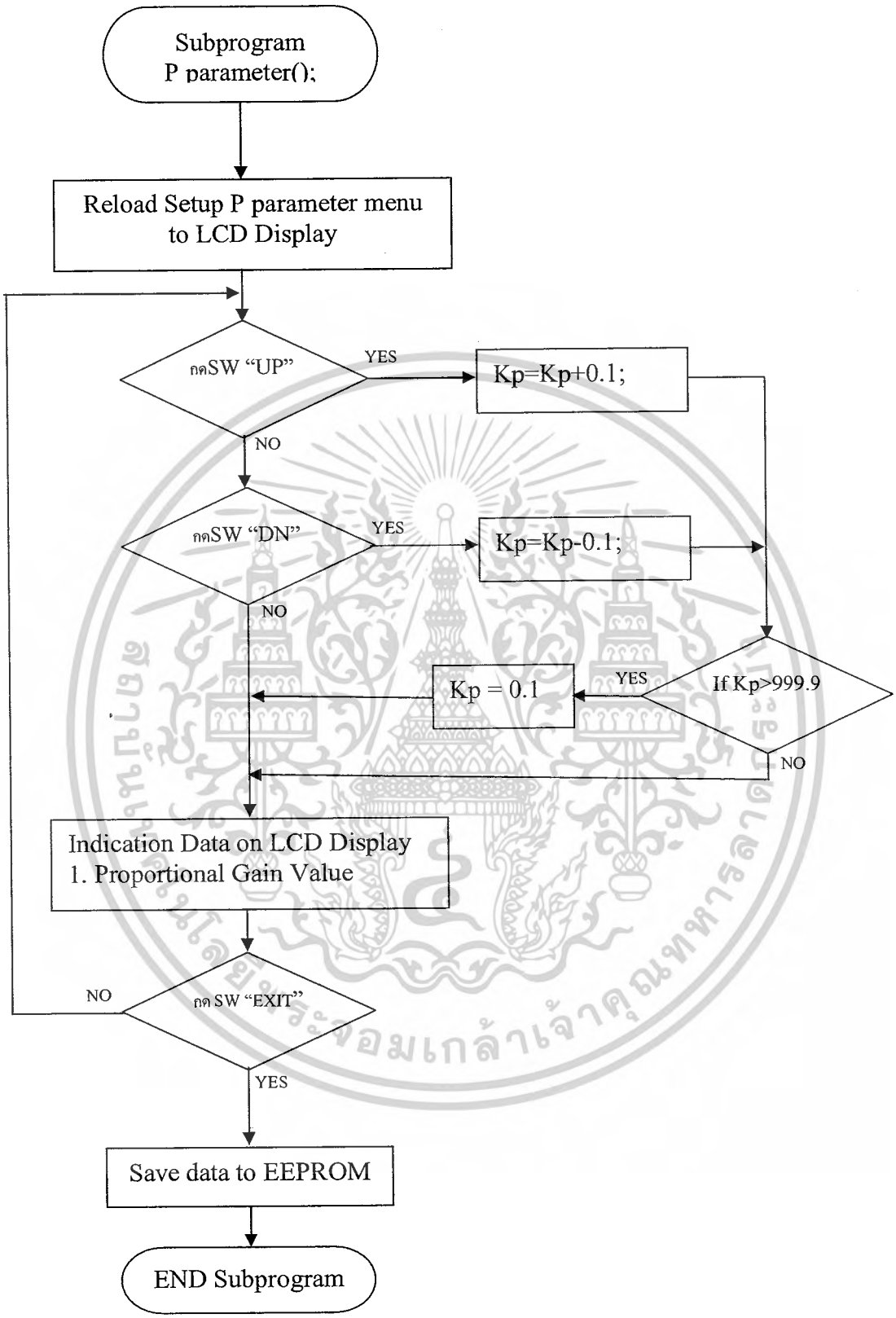
[5] Modern control system analysis and design using MATLAB, Robert H.Bishop

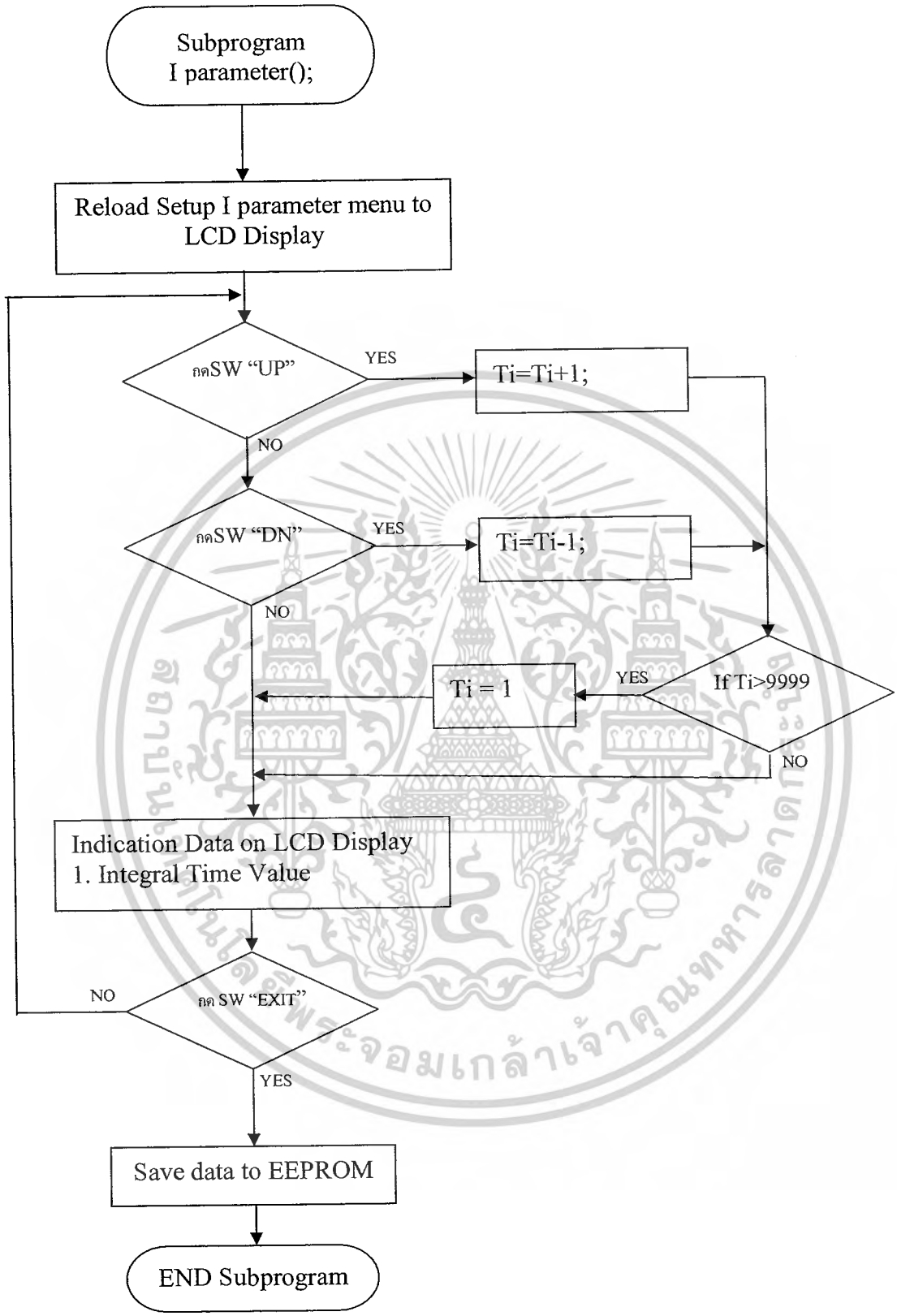
[6] O.J. Smith , " A controller to overcome dead time ," ISA , J. , vol.6 , no.2 , pp28-33 , 1959.

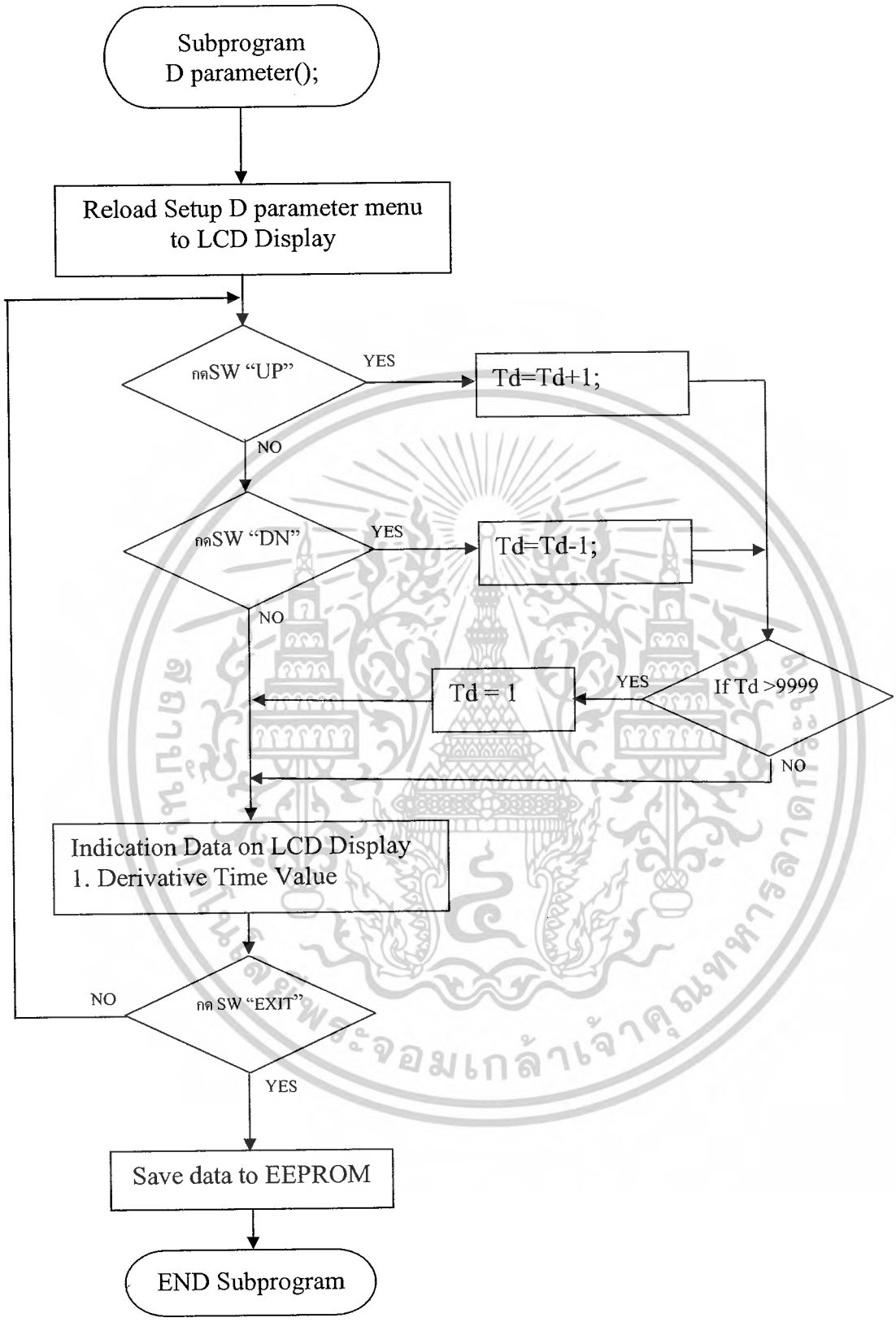
ภาคผนวก ก

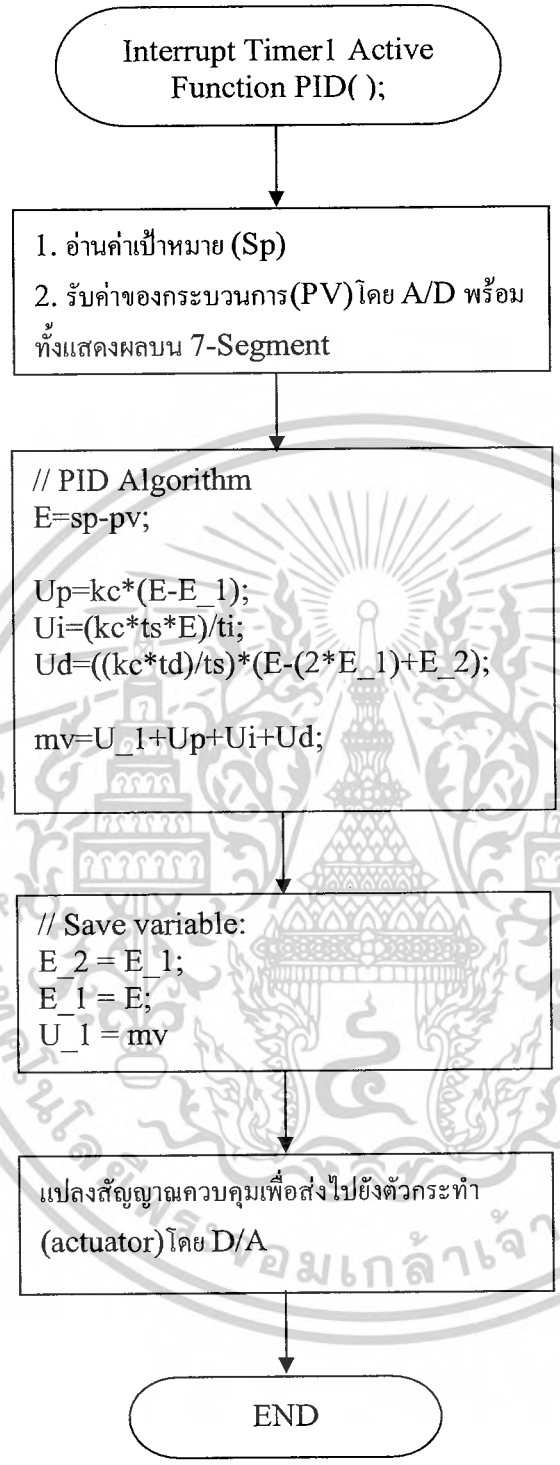


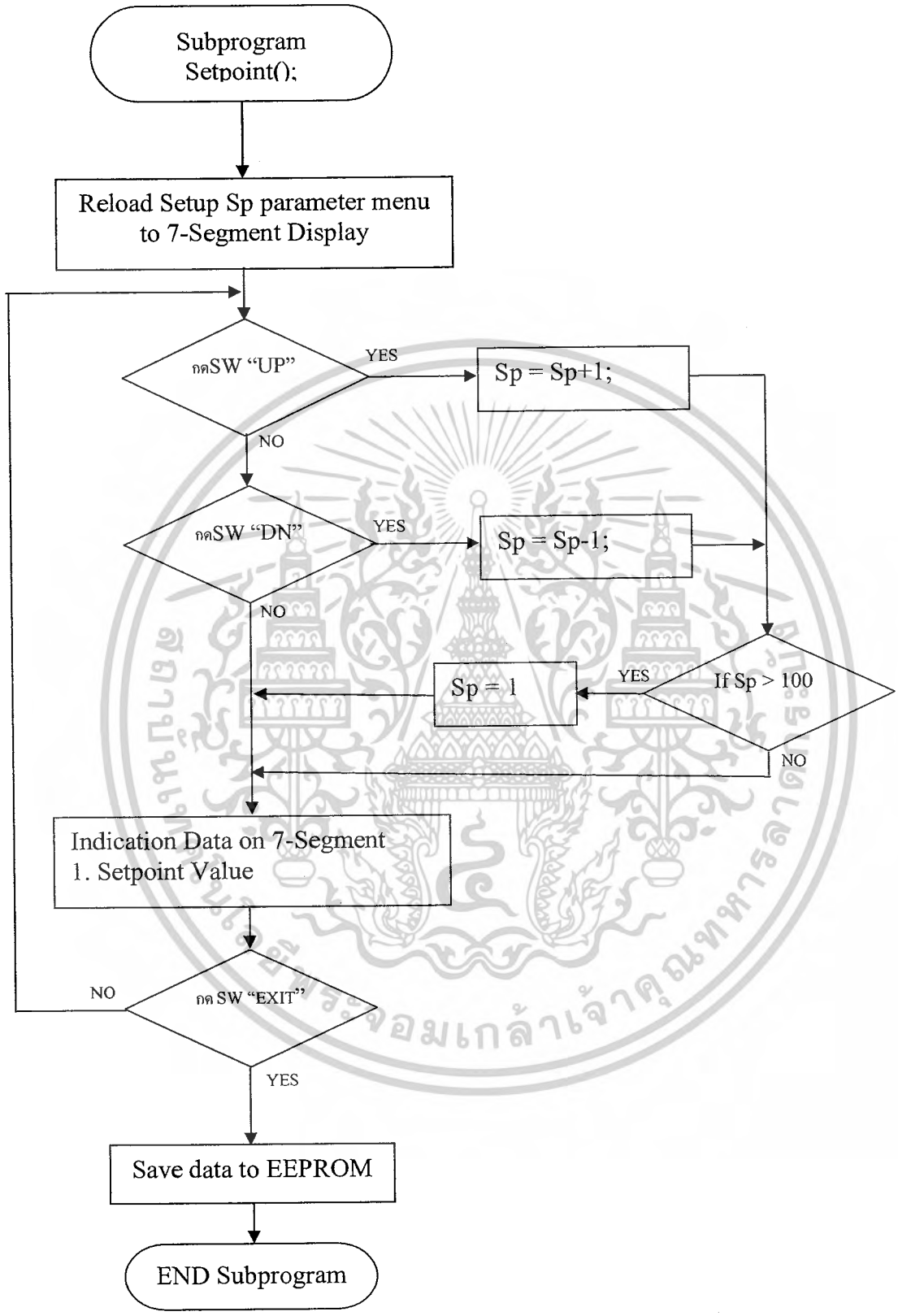


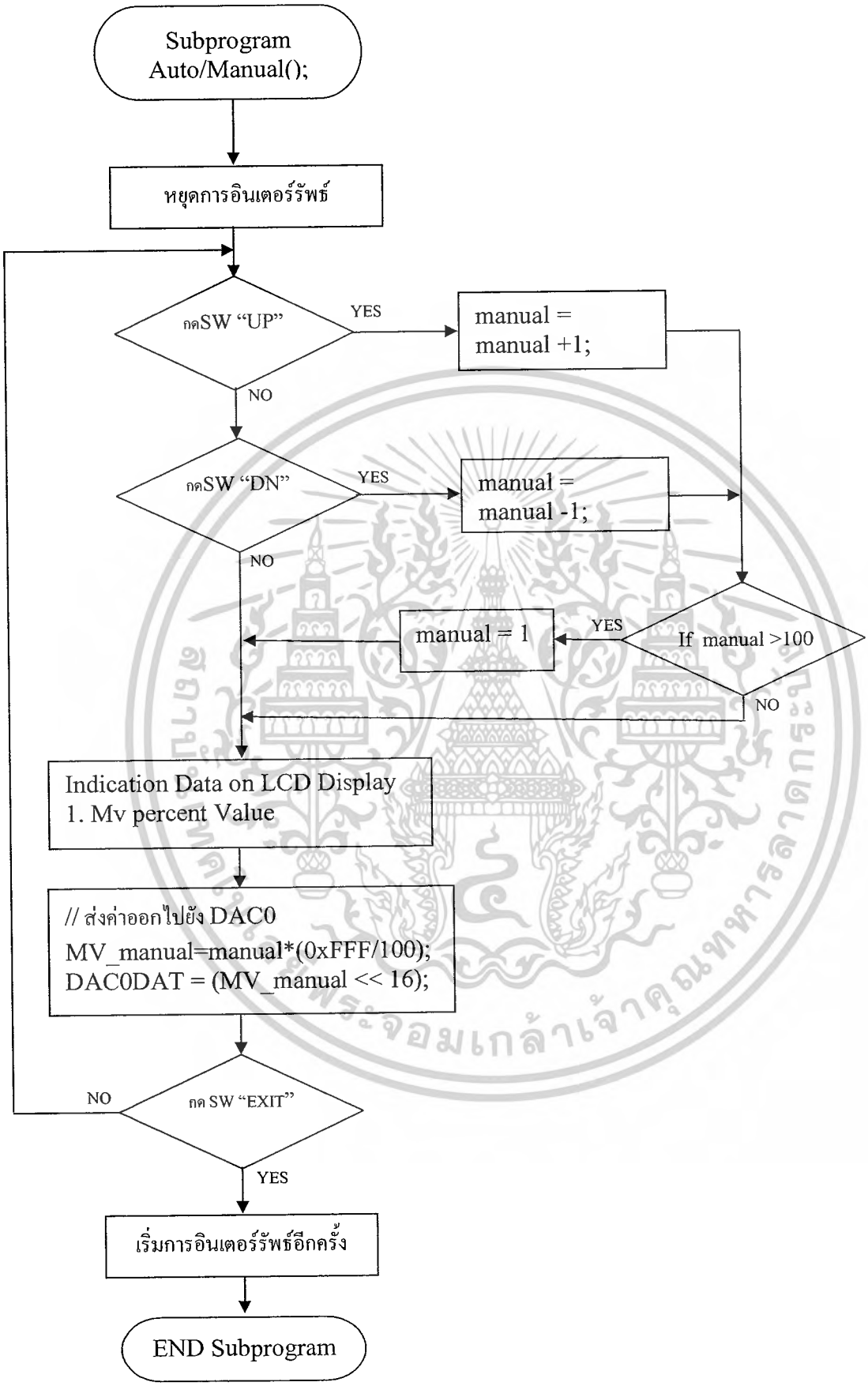


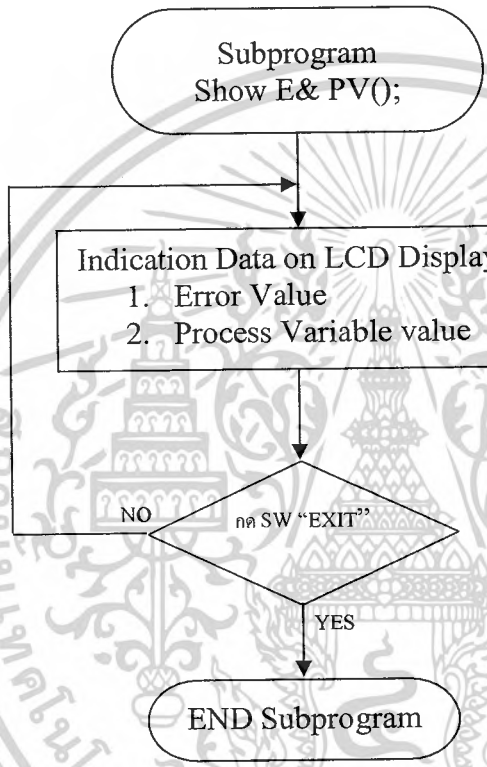












```
#include <ADUc7024.H>                    // ADUc7024 MPU Register
#include <stdio.h>                       // For Used Function printf

/* pototype  section */
///////////////LCD//////////////////
void delay(unsigned long int);           // Delay Function
void lcd_init();                         // Initial LCD
void lcd_out_data4(unsigned char);       // Strobe 4-Bit Data to LCD
void lcd_write_byte(unsigned char);      // Write 1 Byte Data to LCD
void lcd_write_control(unsigned char);   // Write Instruction
void lcd_write_ascii(unsigned char);     // Write LCD Display(ASCII)
void goto_cursor(unsigned char);         // Set Position Cursor LCD
void lcd_print(unsigned char*);          // Print Display to LCD
char busy_lcd(void);                     // Read Busy LCD Status
void enable_lcd(void);                   // Enable Pulse
void test_lcd(void);
void show_value(char);
void initial_adc_dac(void);

///////////////MAX7219//////////////////
void SPI_Write_Data(unsigned char DataByte);   // Write 1 Byte to SPI
void delay(unsigned long int);                 // Delay Time Function
void MAX7219(unsigned char cmd,unsigned char dataX);
const static unsigned char HexTAB[11] = { 0x7E, 0x30, 0x6D, 0x79, 0x33, 0x5B, 0x5F,
0x70,0x7F,0x7B,0x00};
const static unsigned char point[10] =  { 0xFE, 0xB0, 0xED, 0xF9, 0xB3, 0xDB, 0xDF, 0xF0,0xFF,0xFB};
void initial_max(void);

////////////////Interupt  timer////////////////
void IRQ_Handler (void) __irq;

///////////////RTC DS1307////////////////
unsigned char ReadDS1307 (unsigned char Read_Addr);
void WriteDS1307(unsigned char Write_Addr,unsigned char Set_Time);

void Tx_Byte(unsigned char DataOut);     // Write Data to UART
```

```
//////////////by myself//////////////////
void check_sw(void);
void name_function(void);
void pid_para(void);
void auto_man(void);
void sp_pv(void);
void mv_e(void);
void halt(void);
void function_pid(void);
void p_para(void);
void i_para(void);
void d_para(void);
void boot_data(void);
void display_sp(void);
void send_data(void);
void Send2Com(int data2send);


int sw;
float ts=0.25;
int ti,td;
char value_U[3],value_pv[3],value_sp[3],value_E[3];
float kp,kc;
char value_kp[2],value_ti[3],value_td[3],value_sp[3],value_manual[3];
int sp,manual,MV_manual;
int kp_l,kp_h,ti_l,ti_h,td_l,td_h,sp_l,sp_h;
float pv,sumpv,kkk;
float U_percent,mv;
int data1,data2,data3,data4;
int count_sent,count1;
float U_1,E,E_1,E_2,Up,Ud,Ui,Ui_1,pv_1,Upid;
int U,Q;
```

```
// LCD Routines for "ET-ARM7 BASE ADUc7024"
// Character 16x2 4-Bit Mode Interface
// EN = P4.4
// RW = P4.5
// RS = P4.6
// D4 = P4.0
// D5 = P4.1
// D6 = P4.2
// D7 = P4.3
#define LCD_EN    0x00100000        // P4.4(0000 0000 000x 0000 0000 0000 0000 0000)
#define LCD_RW    0x00200000        // P4.5(0000 0000 00x0 0000 0000 0000 0000 0000)
#define LCD_RS    0x00400000        // P4.6(0000 0000 0x00 0000 0000 0000 0000 0000)
#define LCD_D4    0x00010000        // P4.0(0000 0000 0000 000x 0000 0000 0000 0000)
#define LCD_D5    0x00020000        // P4.1(0000 0000 0000 00x0 0000 0000 0000 0000)
#define LCD_D6    0x00040000        // P4.2(0000 0000 0000 0x00 0000 0000 0000 0000)
#define LCD_D7    0x00080000        // P4.3(0000 0000 0000 x000 0000 0000 0000 0000)

#define LCD_DATA  (LCD_D7|LCD_D6|LCD_D5|LCD_D4)
#define LCD_IOALL (LCD_D7|LCD_D6|LCD_D5|LCD_D4|LCD_RS|LCD_RW|LCD_EN)

#define lcd_rs_set()    GP4SET = LCD_RS          // RS = 1 (Select Instruction)
#define lcd_rs_clr()    GP4CLR = LCD_RS          // RS = 0 (Select Data)
#define lcd_rw_set()    GP4SET = LCD_RW          // RW = 1 (Read)
#define lcd_rw_clr()    GP4CLR = LCD_RW          // RW = 0 (Write)
#define lcd_en_set()    GP4SET = LCD_EN          // EN = 1 (Enable)
#define lcd_en_clr()    GP4CLR = LCD_EN          // EN = 0 (Disable)
#define lcd_dir_write() GP4DAT = 0x7F000000      // LCD Data Bus = Write
#define lcd_dir_read()  GP4DAT = 0x70000000      // LCD Data Bus = Read

#define lcd_clear()          lcd_write_control(0x01)   // Clear Display
#define lcd_cursor_home()    lcd_write_control(0x02)   // Set Cursor = 0
#define lcd_display_on()     lcd_write_control(0x0E)   // LCD Display Enable
#define lcd_display_off()    lcd_write_control(0x08)   // LCD Display Disable
#define lcd_cursor_blink()   lcd_write_control(0x0F)   // Set Cursor = Blink
#define lcd_cursor_on()      lcd_write_control(0x0E)   // Enable LCD Cursor
#define lcd_cursor_off()     lcd_write_control(0x0C)   // Disable LCD Cursor
```

```
#define  lcd_cursor_left()          lcd_write_control(0x10)    // Shift Left Cursor
#define  lcd_cursor_right()         lcd_write_control(0x14)    // Shift Right Cursor
#define  lcd_display_sleft()        lcd_write_control(0x18)    // Shift Left Display
#define  lcd_display_sright()       lcd_write_control(0x1C)    // Shift Right Display
////////////////////////////////////////////////////////////////////////////////////////////
// Define LCD PinIO Mask (Bit[23..16])
#define  MAX7219_CLK   0x00100000          // P1.4(0000 0000 000x 0000 0000 0000 0000 0000)
#define  MAX7219_DIN   0x00400000          // P1.6(0000 0000 0x00 0000 0000 0000 0000 0000)
#define  MAX7219_LOAD  0x00200000          // P1.7(0000 0000 x000 0000 0000 0000 0000 0000)

#define  DIN_SET()      GP1SET = MAX7219_DIN      // DIN  = 1
#define  DIN_CLR()      GP1CLR = MAX7219_DIN      // DIN  = 0
#define  CLK_SET()      GP1SET = MAX7219_CLK      // CLK  = 1 (Shift Data)
#define  CLK_CLR()      GP1CLR = MAX7219_CLK      // CLK  = 0
#define  LOAD_SET()     GP1SET = MAX7219_LOAD     // LOAD = 1 (Latch Data)
#define  LOAD_CLR()     GP1CLR = MAX7219_LOAD     // LOAD = 0


void initial_adc_dac(void)
{
        ADCCON  = 0x00000000;          // Reset ADC Config
        ADCCON |= 0x00000020;          // Power-ON ADC Function
        delay(1000);                   // Wait ADC Power-on Ready

  ADCCON |= 0x00001400;                // ADC Clock = fADC/32
  ADCCON |= 0x00000300;                // Acquisition Time = 16 Cycle Clock
  ADCCON &= 0xFFFFFFE7;                // ADC = Single-End Mode
  ADCCON |= 0x00000004;                // Continue Software Convert
  REFCON  = 0x00000001;                // Used Internal 2.5V Reference
  ADCCON |=      0x00000080;           // ADC Start Conversion

        // Initial DAC0
        DAC0CON &= 0xDF;               // DAC0 Used Sysytem Clock
        DAC0CON |= 0x10;               // Enable DAC0
        DAC0CON |= 0x02;               // DAC0 Output Range = +Vref..AGND
        REFCON  = 0x01;                // Used Internal 2.5V Reference
```

```
        // Enable DAC0
        DAC1CON |= 0x02;                    // DAC0 Output Range = +Vref..AGND
        REFCON   = 0x01;                    // Used Internal 2.5V Reference
}


/*****************************/
/* Write Character To UART */
/*****************************/
void Tx_Byte(unsigned char DataOut)         // Write character to Serial Port
{
   while(!(0x40==(COMSTA0 & 0x40)))         // Wait TX Complete
        {
        }
        COMTX = DataOut;                    // Write CR
        while(!(0x40==(COMSTA0 & 0x40)))    // Wait TX Complete
        {
        }
}

/*****************************/
/* Read Character From UART */
/*****************************/
int getchar (void)                          // Read character from Serial Port
{
          while(!(0x01==(COMSTA0 & 0x01)))  // Wait Receive Data Ready
          {
          }
  return (COMRX);
}


/************************/
/* Delay Time Function */
/*    1-4294967296    */
/************************/
```

```
void delay(unsigned long int count1)
{
 while(count1 > 0) {count1--;}                                  // Loop Decrease Counter
}


/*****************************/
/* Set LCD Position Cursor */
/*****************************/
void goto_cursor(unsigned char i)
{
        i |= 0x80;                                              // Set DD-RAM Address Command
        lcd_write_control(i);
}

/*****************************/
/* Write Instruction to LCD */
/*****************************/
void lcd_write_control(unsigned char val)
{
        lcd_rs_clr();                                           // RS = 0 = Instruction Select
        lcd_write_byte(val);                                    // Strobe Command Byte
}

/*****************************/
/* Write Data 1 Byte to LCD */
/*****************************/
void lcd_write_byte(unsigned char val)
{
         lcd_out_data4((val>>4)&0x0F);                          // Strobe 4-Bit High-Nibble to LCD
         enable_lcd();                                          // Enable Pulse
         lcd_out_data4(val&0x0F);                               // Strobe 4-Bit Low-Nibble to LCD
         enable_lcd();                                          // Enable Pulse
        while(busy_lcd());                                      // Wait LCD Execute Complete
 }
```

```
/***********************/
/* Enable Pulse to LCD */
/***********************/
void enable_lcd(void)                          // Enable Pulse
{
        unsigned int i;                        // Delay Count
        lcd_en_set();                          // Enable ON
        for (i=0;i<10;i++);
        lcd_en_clr();                          // Enable OFF
}

/*******************/
/* Wait LCD Ready */
/*******************/
char busy_lcd(void)
{
 unsigned char busy_status;                    // Busy Status Read
 unsigned int i;                               // Delay Count

 lcd_dir_read();                               // LCD Data Bus = Read
 lcd_rs_clr();                                 // Instruction Select
 lcd_rw_set();                                 // Read Direction
 lcd_en_set();                                 // Start Read Busy

 for (i=0;i<100;i++);                          // Delay Before Read
 busy_status = (GP4DAT & 0x08);                // Read LCD Data

 if(busy_status == 0x08)                       // Read & Check Busy Flag
        {
        lcd_en_clr();                          // Disable Read
        lcd_rw_clr();                          // Default = Write Direction
        lcd_dir_write();                       // LCD Data Bus = Write
          return 1;                            // LCD Busy Status
        }
 else
        {
```

```
        lcd_en_clr();                          // Disable Read
        lcd_rw_clr();                          // Default = Write Direction
        lcd_dir_write();                       // LCD Data Bus = Write
          return 0;                            // LCD Ready Status
        }
}

/************************************/
/* Print Display Data(ASCII) to LCD */
/************************************/
void lcd_print(unsigned char* str)
{
 int i;
 for (i=0;i<16 && str[i]!=0;i++)               // 16 Character Print
 {
  lcd_write_ascii(str[i]);                     // Print Byte to LCD
 }
}

/*****************************/
/* Write Data(ASCII) to LCD */
/*****************************/
void lcd_write_ascii(unsigned char c)
{
 lcd_rs_set();                                 // RS = 1 = Data Select
 lcd_write_byte(c);                            // Strobe 1 Byte to LCD
}

/*****************************/
/* Strobe 4-Bit Data to LCD */
/*****************************/
void lcd_out_data4(unsigned char val)
{
 GP4CLR = (LCD_DATA);                          // Reset 4-Bit Pin Data
 GP4SET = (val<<16);                           // 0000:0000:0,RS,RW,EN:DDDD:0000:0000:0000:0000
```

```
}

/*******************************/
/* Initial 4-Bit LCD Interface */
/*******************************/
void lcd_init()
{
 unsigned int i;                                    // Delay Count

 GP4CLR = (LCD_IOALL);                              // Reset (RS,RW,EN,4-Bit Data) Pin
 for (i=0;i<100000;i++);                            // Power-On Delay (15 mS)

 GP4CLR = (LCD_IOALL);                              // Reset (RS,RW,EN,4-Bit Data) Pin
 GP4SET = (LCD_D5|LCD_D4);          // 0000:0000:(0,RS,RW,EN:0011):0000:0000:0000:0000
 enable_lcd();                                      // Enable Pulse
 for (i=0;i<10000;i++);                             // Delay 4.1mS

 GP4CLR = (LCD_IOALL);                              // Reset (RS,RW,EN,4-Bit Data) Pin
 GP4SET = (LCD_D5|LCD_D4);          // 0000:0000:(0,RS,RW,EN:0011):0000:0000:0000:0000
 enable_lcd();                                      // Enable Pulse
 for (i=0;i<1000;i++);                              // delay 100uS

 GP4CLR = (LCD_IOALL);                              // Reset (RS,RW,EN,4-Bit Data) Pin
 GP4SET = (LCD_D5|LCD_D4);          // 0000:0000:(0,RS,RW,EN:0011):0000:0000:0000:0000
 enable_lcd();                                      // Enable Pulse
 while(busy_lcd());                                 // Wait LCD Execute Complete

 GP4CLR = (LCD_IOALL);                              // Reset (RS,RW,EN,4-Bit Data) Pin
 GP4SET = (LCD_D5);                 // 0000:0000:(0,RS,RW,EN:0011):0000:0000:0000:0000
 enable_lcd();                                      // Enable Pulse
 while(busy_lcd());                                 // Wait LCD Execute Complete

 lcd_write_control(0x28);           // Function Set (DL=0 4-Bit,N=1 2 Line,F=0 5X7)
 lcd_write_control(0x0C);           // Display on/off Control (Entry Display,Cursor off,Cursor not Blink)
 lcd_write_control(0x06);           // Entry Mode Set (I/D=1 Increment,S=0 Cursor Shift)
 lcd_write_control(0x01);           // Clear Display  (Clear Display,Set DD RAM Address=0)
```

```
 for (i=0;i<10000;i++);                                    // Wait Command Ready
}


void MAX7219(unsigned char cmd,unsigned char dataX)
 {
  LOAD_CLR();
  SPI_Write_Data(cmd);                                     // Digit[8]
  SPI_Write_Data(dataX);                                   // 7
  LOAD_SET();
}

//Read Byte From RTC
unsigned char ReadDS1307 (unsigned char Read_Addr)
        {
                unsigned char Get_Byte;
                I2C1CCNT = 0x00;

                //1st Start Condition
                I2C1ADR = 0xD0;                            //data write slave recieve
                I2C1MTX = Read_Addr;
                while((I2C1FSTA & 0x30)!=0x00);

                //2nd Start Condition
                I2C1CNT = 0;
                I2C1ADR = 0xD1;                                    //data read slave transmitter

                //enable Generate Stop Condition
                I2C1CCNT = 0x80;

                //Wait recive Data From Device
                while((I2C1MSTA & 0x08)!=0x08);
                Get_Byte = I2C1MRX;
                return Get_Byte;
        }
```

```
//Write Byte to RTC : DS1307
        void WriteDS1307(unsigned char Write_Addr,unsigned char Set_Time)
        {
                I2C1MTX = Write_Addr;
                I2C1MTX = Set_Time;
                I2C1ADR = 0xD0;

                //Wait I2C Send Complete
                while ((I2C1MSTA & 0x04)!=0x04);

        }



void initial_max()
{
// Initial SPI Interface to MAX7219
 // GP1DAT = Direction Control Bit[31..24]
 //              = 0 = Input,1=Output
 //              = dddd dddd xxxx xxxx xxxx xxxx xxxx xxxx
 // GP1DAT = Output Control Bit[23..16]
 //              = xxxx xxxx pppp pppp xxxx xxxx xxxx xxxx
 // GP1DAT = Input Read Bit[7..0]
 // GP1SET  = Bit[23..16]
 // GP1CLR = Bit[23..16]

 GP1CON &= 0xCCCCFFFF;                          // P1[7..4] = GPIO Function
 GP1DAT |= 0xF0000000;                          // P1[7..4] = Output

 // Initial Start MAX7219 Signal
 LOAD_CLR();                                    // Initial Start Signal
 DIN_CLR();
 CLK_CLR();

 // Initial MAX7219 Function Set
        MAX7219(0x0A,0x08);
```

```
        MAX7219(0x0B,0x07);
        MAX7219(0x09,0x00);
        MAX7219(0x0C,0x01);
        MAX7219(0x0F,0x00);

       MAX7219(0x01,HexTAB[0]);
       MAX7219(0x02,HexTAB[0]);
       MAX7219(0x03,HexTAB[0]);
       MAX7219(0x04,HexTAB[0]);
       MAX7219(0x05,HexTAB[0]);
       MAX7219(0x06,HexTAB[0]);
       MAX7219(0x07,HexTAB[0]);
       MAX7219(0x08,HexTAB[0]);
}


/*********************************/
/* Write Data or Command to LCD */
/* D/C = "0" = Write Command    */
/* D/C = "1" = Write Display    */
/*********************************/
void SPI_Write_Data(unsigned char DataByte)
{
  unsigned char Bit = 0;                    // Bit Counter
  int x;                                    // Short Delay Counter

  for (Bit = 0; Bit < 8; Bit++)             // 8 Bit Write
  {
    DIN_CLR();                              // Prepared Default DIN
        if((DataByte & 0x80) >> 7)          // MSB First of Data Bit(7..0)
        {
         DIN_SET();                         // DIN = "1"
        }

        CLK_SET();                          // Strobe Bit Data
        x++;                                // Delay Clock
```

```
        x++;
         CLK_CLR();                                  // Next Clock
         DataByte <<= 1;                             // Next Bit Data
  }
}


/*****************************/
/* Interrupt Service Rountine */
/*****************************/
void IRQ_Handler (void) __irq                        // IRQ Service Routine
{
 if ((IRQSTA & 0x00000008) != 0)                     // if Timer1 IRQ Flag Status
         {
          //send_data();
          function_pid();                            // not scope for led
          T1CLRI = 0;                                // Clear Timer1 Trigger IRQ Flag
         }
        return ;
}


void main(void)
{        ti=1; U=0; mv=0;  Ui=0; Ud=0;
         kp=0.1;  E_2=0;   E_1=0;
         U_1=0;  Ui_1=0;

 // Initial I2C Interface
         GP1CON &= 0xFFFFCCFF;
         GP1CON |= 0x00002200;

 // I2C-MASTER setup
         I2C1CFG = 0x82;

 // I2C-CLOCK = 100 KHz
         I2C1DIV = 0xCFCF;
```

```
        GP1CON |= 0x00000011;                   // Setup P1.1 = TXD & P1.0 = RXD
// Initial UART = 9600BPS
        COMCON0 = 0x80;                         // Setting DLAB
        COMDIV0 = 0x88;                         // Setting DIV0 and DIV1 to DL calculated
        COMDIV1 = 0x00;
        COMCON0 = 0x07;                         // Clearing DLAB

        GP3CON = 0x00000000;
        GP0CON = 0x00000000;
        GP3DAT = 0xFF000000;
        GP0DAT = 0x00000006;
        lcd_dir_write();                        // Initial LCD Write Data
        lcd_init();                             // Initial LCD
        lcd_clear();
        initial_max();
        initial_adc_dac();

        IRQEN |= 0x00000008;                    // Enable Timer1 Trigger IRQ Interrupt

// HCLK = 41.78 MHz
// Time 1 Cycle = 1 / 41.78 MHz
//              = 23.9348 nS
// Time 1 KHz  = 1 / 1000
//              = 1 mS
// 1KHz Signal  = Low 125mS + High 125mS
// Count 1 KHz  = 125mS / 23.9348nS
//              = 10445042 Cycle

T1LD  = 10445042;                               // Timer1 Count 500uS
;
T1CON &= 0xFFFFFFF0;                            // Prescale = HCLK / 1
T1CON &= 0xFFFFFFCF;                            // Format = Binary Counter
T1CON |= 0x00000040;                            // Timer1 Mode = Periodic
T1CON &= 0xFFFFFEFF;                            // Timer1 = Count Down
T1CON &= 0xFFFFF1FF;                            // Timer1 Clock Source = HCLK
```

```
T1CON |= 0x00000080;                                    // Timer1 Enable
boot_data();
display_sp();
// Wait Timer1 Interrupt //
while(1)
                {
                goto_cursor(0x00);                      // Set Cursor Line-1
                lcd_print(" PID CONTROLLER ");
                goto_cursor(0x40);                      // Set Cursor Line-1
                lcd_print("   USING ARM7   ");
                delay(4000000);

                goto_cursor(0x00);                      // Set Cursor Line-1
                lcd_print(" PRESS  UP/DOWN ");
                goto_cursor(0x40);                      // Set Cursor Line-1
                 lcd_print("   TO SELECT FN  ");
                delay(2500000);
                check_sw();
                while(sw==0xE9){delay(20000); name_function();
                 }
} }

void name_function(void)
{
        int state=0;
        while(1)
        {
                check_sw();
                if(sw==0xE9)
                        {
                                while(sw==0xE9)check_sw();
                                state++;
                                if(state>5)state=1;
                        }

                if(sw==0xB9)
```

```
{
        while(sw==0xB9)check_sw();
        state--;
        if(state<0)state=5;
}

switch(state)
{
        case 1:  goto_cursor(0x00);                    // Set Cursor Line-1
                 lcd_print("Function 1     ");
                 goto_cursor(0x40);                    // Set Cursor Line-1
                 lcd_print("PID PARAMETER  ");
                 if(sw==0x79)
                 {
                 while(sw==0x79)check_sw();
                 pid_para();
                 }
                 break;

        case 2:  goto_cursor(0x00);                    // Set Cursor Line-1
                 lcd_print("Function 2     ");
                 goto_cursor(0x40);                    // Set Cursor Line-1
                 lcd_print("AUTO / MANUAL  ");
                 if(sw==0x79)
                 {
                 while(sw==0x70)check_sw();
                 auto_man();
                 }
                 break;

        case 3:  goto_cursor(0x00);                    // Set Cursor Line-1
                 lcd_print("Function 3     ");
                 goto_cursor(0x40);                    // Set Cursor Line-1
                 lcd_print("SETPOINT       ");
                 if(sw==0x79)
                 {
```

```
                while(sw==0x79)check_sw();
                sp_pv();
                }
                break;

        case 4:  goto_cursor(0x00);                    // Set Cursor Line-1
                lcd_print("Function 4     ");
                goto_cursor(0x40);                     // Set Cursor Line-1
                lcd_print("SHOW  MV & ERROR");
                if(sw==0x79)
                {
                while(sw==0x79)check_sw();
                mv_e();
                }
                break;;

        case 5:  goto_cursor(0x00);                    // Set Cursor Line-1
                lcd_print("Function 5     ");
                goto_cursor(0x40);                     // Set Cursor Line-1
                lcd_print("HALT           ");
                if(sw==0x79)
                {
                while(sw==0x79)check_sw();
                halt();
                }
                break;
        }
        if(sw==0xD9){while(sw==0xD9)check_sw();break;}      //exit function
}}


void pid_para(void)
        {
        int state,a;
        lcd_clear();
        while(sw==0xD0)check_sw();
```

```
while(1)
{       check_sw();
        if(sw==0xE9)
                        {
                                while(sw==0xE9)check_sw();
                                state++;
                                if(state>3)state=1;
                        }

        if(sw==0xB9)
                        {
                                while(sw==0xB9)check_sw();
                                state--;
                                if(state<0)state=3;
                        }
        switch(state)
            {
                        case 1:   goto_cursor(0x00);                    // Set Cursor Line-1
                                  lcd_print("PID PARAMETER   ");
                                  goto_cursor(0x40);                    // Set Cursor Line-1
                                  lcd_print("P PARAMETER     ");
                                  if(sw==0x79)
                                  {
                                  while(sw==0x79)check_sw();
                                  p_para();
                        IRQCLR |= 0x00000008;     // Disable Timer1 Trigger IRQ Interrupt
                        a=kp*10; kp_h=a/100; kp_l=a%100;
                        WriteDS1307(0x0c,kp_h);     delay(100000);
                        WriteDS1307(0x0b,kp_l);     delay(100000);
                        IRQEN |= 0x00000008;       // Enable Timer1 Trigger IRQ Interrupt
                                  }
                                  break;
                        case 2:   goto_cursor(0x00);                    // Set Cursor Line-1
                                  lcd_print("PID PARAMETER   ");
                                  goto_cursor(0x40);                    // Set Cursor Line-1
                                  lcd_print("I PARAMETER     ");
```

```
                    if(sw==0x79)
                    {
                    while(sw==0x79)check_sw();
                    i_para();
                    IRQCLR |= 0x00000008;
                    ti_h=ti/100; ti_l=ti%100;
                    WriteDS1307(0x08,ti_l);      delay(100000);
                    WriteDS1307(0x09,ti_h);      delay(100000);
                    IRQEN |= 0x00000008;
                    }
                    break;
            case 3: goto_cursor(0x00);                  // Set Cursor Line-1
                    lcd_print("PID PARAMETER  ");
                    goto_cursor(0x40);                  // Set Cursor Line-1
                    lcd_print("D PARAMETER    ");
                    if(sw==0x79)
                    {
                    while(sw==0x79)check_sw();
                    d_para();
                    IRQCLR |= 0x00000008;
                    td_h=td/100; td_l=td%100;
                    WriteDS1307(0x0e,td_l);      delay(100000);
                    WriteDS1307(0x0f,td_h);      delay(100000);
                    IRQEN |= 0x00000008;
                    }
                    break;
        }
        if(sw==0xD9){while(sw==0xD9)check_sw();break;}
    }
}


void p_para(void)
    {
    lcd_clear();
    while(1)
```

```
        {
        check_sw();
        if(sw==0xE9) kp=kp+0.1;     if(sw==0x69) kp=kp+10;    //up value
        if(sw==0xB9) kp=kp-0.1;     if(sw==0x39) kp=kp-10;    //down value
        delay(100000);
        if(kp<0.1) kp=999.9;
        if(kp>999.9) kp=0.1;

        goto_cursor(0x00);
        lcd_print("P parameter    ");
        goto_cursor(0x40);
        sprintf(value_kp,"%1.1f",kp);
        lcd_print("Kp = ");
        lcd_print(value_kp);            if(kp<1000) lcd_print("          ");

        if(sw==0xD9){while(sw==0xD9)check_sw();break;}

        }}


void i_para(void)
{
        lcd_clear();
        while(1)
        {
        check_sw();

        if(sw==0xE9) ti++;              if(sw==0x69) ti=ti+10;      //up value
        if(sw==0xB9) ti--;              if(sw==0x39) ti=ti-10;      //down value
        delay(100000);
        if(ti>9999) ti=0;
        if(ti<0) ti=9999;

        goto_cursor(0x00);
        lcd_print("I parameter    ");
        goto_cursor(0x40);
```

```
        sprintf(value_ti,"%d",ti);
        lcd_print("Ti = ");
        lcd_print(value_ti);  if(ti<1000) lcd_print("    ");

        if(sw==0xD9){while(sw==0xD9)check_sw();break;}

        }
}

void d_para(void)
{
        lcd_clear();
        while(1)
        {
        check_sw();
        if(sw==0xE9) td++;              if(sw==0x69) td=td+10;     //up value
        if(sw==0xB9) td--;              if(sw==0x39) td=td-10;     //down value
        delay(100000);
        if(td>9999) td=0;
        if(td<0) td=9999;

        goto_cursor(0x00);
        lcd_print("D parameter    ");
        goto_cursor(0x40);
        sprintf(value_td,"%d",td);
        lcd_print("Td = ");
        lcd_print(value_td);            if(td<100) lcd_print("   ");

        if(sw==0xD9){while(sw==0xD9)check_sw();break;}      //exit function
        }
}

void auto_man(void)
{
         int val,A,B,C,B1,D,G,F;
        float pv_volt,pv_percent;
```

```
IRQCLR |= 0x00000008;

while(1)
{
ADCCP =0;                              // Select Channel to Conversion
delay(1000);                           // Wait Select Channel Ready
while (!ADCSTA){};                     // Wait ADC Conversion Complete (Bit0="1")
val = (ADCDAT >> 16)& 0x00000FFF;      // Shift ADC Result to Integer
pv_volt = val * (2.50 / 4095.0);       // Volt = ADC Result x [2.5V / 4095]
pv_percent=(pv_volt*100)/2.5  ;
pv=pv_percent;

A=pv_percent/100;
B1=pv_percent/10;
B=B1;       if(B1>9) B=B1%10;
C=pv_percent*10; D=C/10; G=D%10;
F=C%10;
MAX7219(0x05,HexTAB[F]);
MAX7219(0x06,point[G]);
MAX7219(0x07,HexTAB[B]);
MAX7219(0x08,HexTAB[A]);
check_sw();

if(sw==0xE9)
        {       manual++; delay(100000);  if(manual>100) manual=100;  }
if(sw==0xB9)
        {       manual--; delay(100000);  if(manual<0) manual=0;        }

goto_cursor(0x00);
lcd_print("MANUAL ON      ");
goto_cursor(0x40);
sprintf(value_manual,"%d",manual);
lcd_print("MV = ");
lcd_print(value_manual); lcd_print(" %    ");

MV_manual=manual*(0xFFF/100);
```

```
        mv=manual;

        DAC0DAT = (MV_manual << 16);
        delay(1000000);
        send_data();                                        //send data to ComTx
        if(sw==0xD9){delay(1000000);break;}
        }
        IRQEN |= 0x00000008;
        }

void sp_pv(void)
        {
        int A,B,C,B1,D,E,F;
        lcd_clear();
        while(1)
        {
        check_sw();
        if(sw==0xE9)
                {       sp++; delay(300000);  if(sp>100) sp=0;          }
        if(sw==0xB9)
                {       sp--; delay(300000);  if(sp<0) sp=100;  }

        goto_cursor(0x00);
        lcd_print("SETPOINT var    ");
        goto_cursor(0x40);
        sprintf(value_sp,"%d",sp);
        lcd_print("SV = ");
        lcd_print(value_sp);            if(sp<100) lcd_print("     ");

        A=sp/100;
        B1=sp/10;
        B=B1;       if(B1>9) B=B1%10;
        C=sp*10; D=C/10; E=D%10;
        F=C%10;
        MAX7219(0x01,HexTAB[F]);
        MAX7219(0x02,point[E]);
```

```
        MAX7219(0x03,HexTAB[B]);
        MAX7219(0x04,HexTAB[A]);
        if(sw==0xD9){while(sw==0xD9)check_sw();break;}
        }
        IRQCLR |= 0x00000008;
        sp_h=sp/100;  sp_l=sp%100;
        WriteDS1307(0x12,sp_l);       delay(100000);
        WriteDS1307(0x13,sp_h);       delay(100000);
        IRQEN |= 0x00000008;
        }

void display_sp(void)
{
        int A,B,C,B1,D,E,F;
                                A=sp/100;
                                B1=sp/10;
                                B=B1;       if(B1>9) B=B1%10;
                                C=sp*10; D=C/10; E=D%10;
                                F=C%10;
                                MAX7219(0x01,HexTAB[F]);
                                MAX7219(0x02,point[E]);
                                MAX7219(0x03,HexTAB[B]);
                                MAX7219(0x04,HexTAB[A]);
}

void mv_e(void)
        {
         lcd_clear();

         while(1)
         {
                check_sw();
                goto_cursor(0x40);
                sprintf(value_U,"%1.1f",mv);
                lcd_print("MV = ");
                lcd_print(value_U);
```

```
            lcd_print(" %");  lcd_print("    ");
            goto_cursor(0x00);
            sprintf(value_E,"%1.3f",E);
            lcd_print("ERROR = ");
            lcd_print(value_E);            if(E<10) lcd_print("    ");

            if(sw==0xD9){while(sw==0xD9)check_sw();break;}
            }
        }


void check_sw(void)
{
        sw = GP0DAT & 0xF9;
}



void function_pid(void)
{       int val,A,B,C,B1,D,G,F;
        float pv_volt,pv_percent;
        kc=kp;
        count1=count1+1;
        send_data();
        ADCCP =0;                          // Select Channel to Conversion
        delay(1000);                       // Wait Select Channel Ready
        while (!ADCSTA){};                 // Wait ADC Conversion Complete (Bit0="1")
        val = (ADCDAT >> 16)& 0x00000FFF;  // Shift ADC Result to Integer
        pv_volt = val * (2.50 / 4095.0);   // Volt = ADC Result x [2.5V / 4095]

        pv_percent=(pv_volt*100)/2.5  ;
        pv=pv_percent;

        sumpv=sumpv+pv;
        if(count1==5){kkk=sumpv/5; count1=0; sumpv=0; }
```

```
                A=kkk/100;
                B1=kkk/10;

                B=B1;       if(B1>9) B=B1%10;
                C=kkk*10; D=C/10; G=D%10;
                F=C%10;
                MAX7219(0x05,HexTAB[F]);
                MAX7219(0x06,point[G]);
                MAX7219(0x07,HexTAB[B]);
                MAX7219(0x08,HexTAB[A]);
////////////////////////////////////////////
/////////PID Algorithm/////////////
////////////////////////////////////////////
        E=sp-pv;                                        //Calculate error

        Up=kc*(E-E_1);                                  //calculate P term
        Ui=(kc*ts*E)/ti;                                //calculate I term
        Ud=((kc*td)/ts)*(E-(2*E_1)+E_2);                //calculate D term

        mv=U_1+Up+Ui+Ud;                                //calculate PID output

        U=(mv*4095)/100;
        if(U>4095)      { Ui=Ui_1;      mv=100; }       //condition
        if(U<0)         { Ui=Ui_1;      mv=0;   }
        U=(mv*4095)/100;

        E_2=E_1;                                        //save variables
        E_1=E;
        U_1=mv;
        Ui_1=Ui;
////////////////////////////////////////////
                DAC0DAT = (U << 16);            // Update DAC0 MV value Output(0..2.5V)for(4-20
mA)     DAC1DAT = (0xCCC << 16);        // Update DAC0 MV value Output(0..2.5V)for(4-20 mA)

        manual=mv;
}
```

```
//send value to COMTx
void send_data(void)
{

        data1=0; data2=10*sp; data3=10*pv; data4=10*mv;
        count_sent=count_sent+1;
                if(count_sent==1)       Tx_Byte(0x41);
                if(count_sent==2)       Send2Com(data1);
                if(count_sent==3)       Tx_Byte(0x42);
                if(count_sent==4)       Send2Com(data2);
                if(count_sent==5)       Tx_Byte(0x43);
                if(count_sent==6)       Send2Com(data3);
                if(count_sent==7)       Tx_Byte(0x44);
                if(count_sent==8)       {Send2Com(data4); count_sent=0;}
}

void Send2Com(int data2send)
{
        int base_100,base_10,base_1,temp,point,a;
                base_100= data2send/1000;
                temp = data2send%1000;
                base_10 = temp /100;
                a = data2send/10;
                base_1  = a%10;
                point   = data2send%10;

        Tx_Byte(base_100+0x30);
        Tx_Byte(base_10+0x30);
        Tx_Byte(base_1+0x30);
        Tx_Byte(0x2E);
        Tx_Byte(point+0x30);
}
```

```
void boot_data(void)
{
        kp_l = ReadDS1307(0x0b);
        kp_h = ReadDS1307(0x0c);
        kp = (kp_h*10)+(kp_l*0.1);

        ti_l = ReadDS1307(0x08);
        ti_h = ReadDS1307(0x09);
        ti = (ti_h*100)+ti_l;

        td_l = ReadDS1307(0x0e);
        td_h = ReadDS1307(0x0f);
        td = (td_h*100)+td_l;

        sp_l = ReadDS1307(0x12);
        sp_h = ReadDS1307(0x13);
        sp = (sp_h*100)+sp_l;

}
```

ภาคผนวก ข

โครงสร้างส่วนประกอบของกระบวนการ
และการออกแบบกระบวนการ

กล่าวนำ

ในที่นี้จะเป็นการออกแบบระบบในการควบคุมกระบวนการ และนำอุปกรณ์ต่างๆที่นำมาใช้ในกระบวนการมาเชื่อมต่อกัน เช่น Thermocouple type K, Thermocouple Transmitter TT95, และเตาอบเพื่อให้ได้กระบวนการตามต้องการ

โครงสร้างส่วนประกอบของกระบวนการ

ส่วนประกอบและอุปกรณ์เชื่อมต่อของกระบวนการ

เตาอบ

เตาอบเป็นส่วนของอุปกรณ์ที่ต้องการควบคุมอุณหภูมิภายใน ดังแสดงในภาพที่ 1 และลักษณะโครงสร้างเตาอบไฟฟ้า ดังแสดงในภาพที่ 2

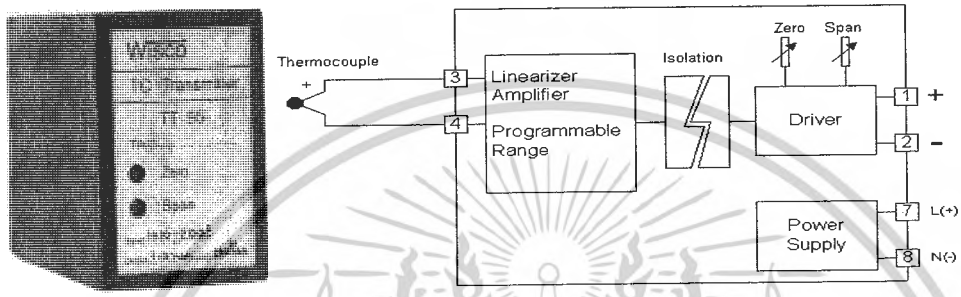
ภาพที่ 1 แสดงภาพเตาอบไฟฟ้า

ข้อมูลทางเทคนิค

เตาเอาไฟฟ้า Turbora รุ่น	TGO-07
กำลังไฟฟ้าสูงสุด	650 W
ความถี่คลื่นไฟฟ้า	50 Hz
ฮีสเตอร์บนรวม	325 W
ความจุเครื่อง	7 L
ฮีสเตอร์ล่าง	325 W
น้ำหนักโดยประมาณ	2.9 Kg.
แรงเคลื่อนไฟฟ้า	220 V

Thermocouple Transmitter TT95

เทอร์โมคัปเปิ้ลจะตรวจจับค่าความร้อนมาจากเตาอบออกมาในรูปของแรงดันไฟฟ้า(mV) เนื่องจากแรงดันที่ได้ไม่เหมาะสมกับการนำไปใช้งาน จึงต้องนำThermocouple Transmitter 95 (TT95) ซึ่งเป็นตัวแปลงสัญญาณที่รับสัญญาณจากเทอร์โมคัปเปิ้ลเพื่อแปลงให้เป็นสัญญาณมาตรฐาน 1-5V จากนั้นจึงนำสัญญาณที่ได้ส่งต่อไปยังตัวควบคุม ลักษณะวงจรการทำงานภายใน TT95 ดังแสดงในภาพที่ 7.2



ภาพที่ 2 แสดงลักษณะของวงจรภายใน TT95

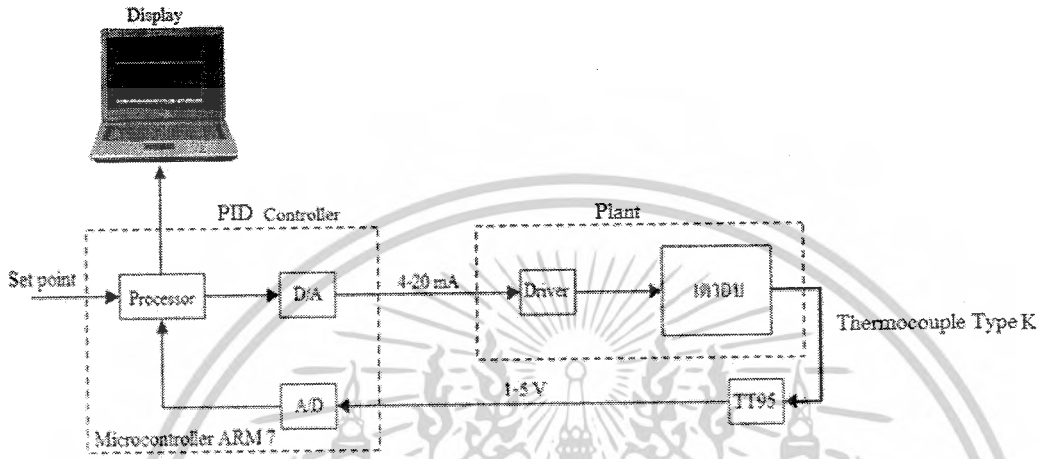
ข้อมูลทางเทคนิค

Thermocouple Input	ย่านการใช้งานของ Type K 0 ถึง 200 °C
Cold Junction Compensation	0 ถึง 50 °C
Linearity	< ± 0.2% ของ span
Output	1 - 5 volt
Power Supply	220 VAC
Ambient Temperature	0 ถึง 50 °C
Isolation Voltage	500 VAC
Connection	Plug -11 pins socket
Mounting	Wall or DIN rail
Dimension	W50×H70×D130 mm

การออกแบบกระบวนการ

Plant

การเชื่อมต่ออุปกรณ์ต่างๆและโครงสร้างไดอะแกรมของกระบวนการ ที่ได้กล่าวไว้ข้างต้น เมื่อนำมาออกแบบจะได้กระบวนการสำหรับควบคุมอุณหภูมิ ดังแสดงในภาพที่ 7.3



ภาพที่ 3 แสดงลักษณะการออกแบบกระบวนการ

การทำงานของกระบวนการนั้น แบ่งออกเป็น 2 ส่วน คือ ส่วนที่เป็นส่วนควบคุม กับส่วนที่เป็นกระบวนการเตาอบ การทำงานของกระบวนการเริ่มจากการกำหนดค่าเป้าหมาย (set point) ซึ่งกำหนดจากตัวควบคุมได้โดยตรง จากนั้นไมโครคอนโทรลเลอร์จะประมวลผลเพื่อสร้างสัญญาณควบคุมจากอัลกอริทึมที่ได้ออกแบบไว้ สัญญาณที่ออกจากตัวควบคุมจะเป็นสัญญาณมาตรฐาน 4-20mA ส่งผ่านไปยังFinal Control Element (heater) เพื่อจ่ายพลังงานความร้อนให้กับกระบวนการเตาอบ โดยการทำงานของตัวทำความร้อนจะถูกควบคุมโดย Solid State Relay ซึ่งสร้างสัญญาณพัลส์ควบคุมตามสัญญาณที่ได้รับ(4-20 mA) อุณหภูมิของกระบวนการเตาอบจึงเกิดการเปลี่ยนแปลง เทอร์โมคัปเปิ้ลจะตรวจจับค่าความร้อนมาจากเตาอบออกมาในรูปของแรงดันไฟฟ้า (mV) เนื่องจากแรงดันที่ได้ไม่เหมาะสมกับการนำไปใช้งาน จึงต้องนำThermocouple Transmitter 95 (TT95) ซึ่งเป็นตัวแปลงสัญญาณที่รับสัญาณจากเทอร์โมคัปเปิ้ลเพื่อแปลงให้เป็นสัญญาณมาตรฐาน 1-5V จากนั้นจึงนำสัญญาณที่ได้ส่งต่อไปยังตัวควบคุม สัญญาณจะถูกส่งเข้าไปยังส่วนของA/D ในตัวไมโครคอนโทรลเลอร์ARM7เพื่อแปลงสัญญาณที่ได้รับซึ่งเป็นสัญญาณอนาลอกให้อยู่ในรูปของสัญญาณดิจิตอล และนำไปประมวลผลต่อไป

ภาคผนวก ค

อุปกรณ์วัดอุณหภูมิ

กล่าวนำ

อุณหภูมิเป็นตัวแปรพื้นฐานทางกระบวนการผลิตอีกตัวแปรหนึ่งที่ต้องมีการควบคุมและแสดงค่าบนหน่วยแสดงของระบบควบคุมเพื่อใช้ในการควบคุมกระบวนการผลิต ดังนั้นในการควบคุมกระบวนการให้เป็นไปตามประสงค์จึงต้องมีการจัดเตรียมเครื่องมือวัดอุณหภูมิ ร่วมกับเซนเซอร์อุณหภูมิในกระบวนการผลิตให้เป็นสัญญาณทางไฟฟ้าที่เหมาะสมกับระบบควบคุม

ซึ่งเซนเซอร์วัดอุณหภูมิมีหลายชนิดให้เลือกใช้ เช่น Thermocouple,RTD ,Filled system เป็นต้นเพื่อให้การวัดอุณหภูมิมีความแม่นยำต้องทำการติดตั้งเซนเซอร์วัดอุณหภูมิให้เหมาะสม

1. RTD (Resistance Temperature Detector)

RTD เป็นความต้านทานที่มีความไวในการเปลี่ยนแปลงค่าความต้านทาน เมื่ออุณหภูมิรอบตัวเปลี่ยนไป โดยสัมประสิทธิ์การเปลี่ยนแปลงความต้านทานต่ออุณหภูมิมีค่าเป็นบวกนั่นคือความต้านทานของวัสดุจะเพิ่มขึ้นเมื่ออุณหภูมิเพิ่มขึ้น

ความต้านทานไฟฟ้าในเส้นลวดโลหะจะเปลี่ยนค่าไปตามสมการ(3.1) ดังนี้

$$R_t = R_0(1 + \alpha T)$$

เมื่อ R_t คือ ค่าความต้านทานของลวดโลหะ ที่อุณหภูมิ t $^{\circ}$C

R_0 คือ ค่าความต้านทานของลวดโลหะที่อุณหภูมิ 0 $^{\circ}$C

α คือ สัมประสิทธิ์ของการเปลี่ยนแปลงค่าความต้านทานไฟฟ้า ต่ออุณหภูมิ 1 $^{\circ}$C($\Omega/\Omega/^{\circ}$C) (Temperature Coefficient of Resistance)

ซึ่งค่า α มีค่าเปลี่ยนไปตามชนิดของโลหะ เช่น แพลทินัม 0.00392 $\Omega/\Omega/^{\circ}$C จากย่านอุณหภูมิ 0 $^{\circ}$C ถึง100 $^{\circ}$C , นิกเกิล 0.0063 $\Omega/\Omega/^{\circ}$C ทองแดง 000.00425 $\Omega/\Omega/^{\circ}$C

ในทางปฏิบัติค่า α ของการเปลี่ยนแปลงอุณหภูมิแต่ละช่วงจะไม่แปรผันเป็นเส้นตรง (Nonlinearity) ในห้องปฏิบัติการมาตรฐานที่ต้องการค่าแน่นอน สามารถทำได้โดยการใช้สมการ (3.2)ต่อไปนี้

$$R_t = R_0(1 + \alpha T + \beta T^2 + \gamma T^4)$$

ค่า α , β และ γ ได้จากการทดลอง (Empirical Quantity) ซึ่งทางบริษัทผู้ผลิตเป็นผู้กำหนดมา เช่น แพลทินัม

α = 3.985

β = -5.856

γ = 4.330

สูตรนี้สามารถใช้ได้ทั้งย่านการใช้งานที่ต้องการ Accuracy สูงแต่โดยทั่วไป การคำนวณจะใช้สูตร $R_t = R_0(1 + \alpha T)$

1.1 ชนิดของอาร์ทีดี (Type of RTD)

1.แพลทินัม เป็นแบบที่นิยมใช้มากที่สุด เขียนบอกไว้เป็น PT ได้แก่ PT-10, PT-100, PT-1000 ความสามารถในการทำซ้ำสูง แต่ความไวต่ำ ราคาแพงมากเมื่อเทียบกับนิกเกิลซึ่งมี ความสามารถในการทำซ้ำน้อย แต่มีความไวมากกว่า และราคาถูกกว่า

2.ทองคำและเงิน ธาตุทั้งสองมีค่าความต้านทานจำเพาะต่ำ

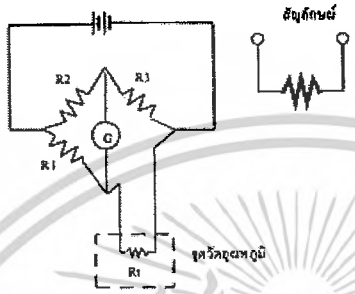
3.ทังสเตนมีค่าความต้านทานจำเพาะสัมพัทธ์สูง มักใช้กับการวัดอุณหภูมิที่มีค่าสูง เพราะ หากใช้ที่อุณหภูมิปกติจะมีความเปราะและยากต่อการใช้งาน

4.นิกเกิล ใช้กับย่านวัดอุณหภูมิสูงๆ มีความเป็นเชิงเส้นต่ำ ทำให้เกิดค่าดริฟต์ (drift) กับเวลา นอกจากนี้ยังมีวัสดุชนิดอื่นๆ ที่ใช้ทำอาร์ทีดี ได้แก่ เหล็ก เป็นต้น

1.2 วงจรการต่อใช้งานของอาร์ทีดี

วงจรต่อใช้งานพื้นฐานของอาร์ทีดี คือ "วีธสโตน บริดจ์" (Wheatstone Bridge) ให้ . "X" คือตัว อาร์ทีดี ซึ่งติดตั้งอยู่ในจุดที่ต้องการวัดอุณหภูมิ รีซีสเตอร์ประกอบอีก 3 ตัว คือ A, B และ C อยู่ในทรานสมิตเตอร์ในอุณหภูมิบรรยากาศ รีซีสเตอร์ A,B และ C ที่ใช้เป็นแบบที่มีความถูกต้องสูง , ค่า Drift (ความต้านทานเปลี่ยนค่าไปเองเมื่อใช้ไปนานๆ) ต่ำมาก

วงจรบริดจ์นี้จะอยู่ในสภาวะสมดุลเมื่ออาร์ทีดีอยู่ในอุณหภูมิ 0 °C ซึ่งจะทำให้อัตราส่วนกัลป์วานอมิเตอร์จะชี้ที่ 0 °C วงจรนี้ให้ได้เมื่อตัวทรานสมิตเตอร์อยู่ใกล้กับตัวอาร์ทีดีมากๆเท่านั้น เพราะ ถ้าสายยาวค่าผิดพลาดจะเกิดขึ้นเนื่องจากความต้านทานของสาย ค่าผิดพลาดนี้ขึ้นอยู่กับความยาวของสายตัวนำอาร์ทีดีและอุณหภูมิของสายตัวนำนี้ ถ้ามีค่ามากขึ้นค่าผิดพลาดก็จะสูงขึ้น วงจรแบบสองสายจึง เหมาะสำหรับการวัดที่ไม่ต้องการความถูกต้องสูงนัก

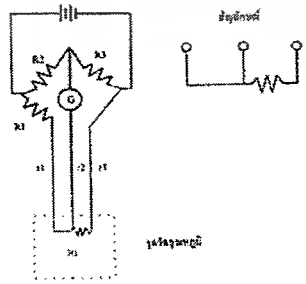


ภาพที่ 1 วงจรอาร์ทีดี แบบ2สาย

วงจรอาร์ทีดี แบบ 3 สาย เป็นแบบมาตรฐานที่นิยมใช้กันมากที่สุดในวงการอุตสาหกรรมทั่วไป สายทั้งสาม a,b และ c จากอาร์ทีดีที่ต่อเข้าวงจรบริดจ์จะต้องมีขนาด ความยาวเท่ากันและอยู่ในบรรยากาศที่มีอุณหภูมิเดียวกันตลอด เพื่อให้ค่าความต้านทานของทั้งสามสายเปลี่ยนแปลงไปในขนาดและทิศทางเดียวกัน เป็นการชดเชยความผิดพลาดอันเกิดจากการลากสายตัวนำในสภาวะงานที่หลีกเลี่ยงไม่ได้จากวงจรเมื่อบริดจ์อยู่ในสภาวะสมดุล

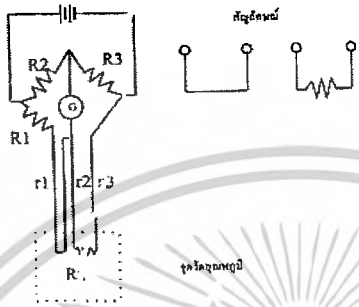
ให้ B=C ปกติจะออกแบบให้เท่ากัน

เนื่องจาก a=b=c, X+b จึงเท่ากับA+a ค่าอุณหภูมิของการวัดจึงขึ้นอยู่กับค่าความต้านทานของอาร์ทีดี “X” เพียงตัวเดียว วงจรวัดแบบ 3 สายนี้เป็นแบบที่ให้ความถูกต้องสูง



ภาพที่ 2 วงจรอาร์ทีดีแบบ3สาย

วงจรการวัดแบบ 4 สาย เป็นแบบที่เลื่อนจุดต่อของบริดจ์(Bridge Point) ไปอยู่ภายนอกสายที่ต่อจากอาร์ทีดี ทั้ง 4 เส้นจะต้องมีขนาด ความยาวเท่ากันและอยู่ในบรรยากาศที่มีอุณหภูมิเดียวกันตลอดเหมือนวงจรการวัดแบบ 3 สาย วิธีนี้ให้ความถูกต้องสูงกว่า



ภาพที่ 3 วงจรอาร์ทีดีแบบ4สาย

วงจรการวัดแบบ 4 สายแบบที่2 ใช้กรณีที่ต้องการความถูกต้องสูงสุด ต้องการทราบค่าอุณหภูมิเป็นจุดๆ ไม่ต้องการวัดค่าแบบต่อเนื่อง เช่น ในห้องปฏิบัติการ ลักษณะการต่อวงจรเป็นแบบ 3 สาย มีสวิตช์สำหรับโยกสลับสายเพื่อหาค่าเฉลี่ย ในการวัดครั้งหนึ่งๆ ต้องทำการอ่านค่า 2 ครั้ง ตามตำแหน่งสวิตช์ ค่าความต้านทานของอาร์ทีดีเป็นค่าเฉลี่ยของค่าที่อ่านได้ทั้งสองครั้ง เพื่อลดความคลาดเคลื่อนอันเกิดจากวิธีการวัดแบบ 3 สาย เนื่องจากค่าความต้านทานของสายอาจไม่เท่ากัน เช่นตำแหน่งบนอ่านได้ 250.170 $^{\circ}$C และตำแหน่งล่างอ่านได้ 250.160 $^{\circ}$C เป็นต้น บวกกับสิ่งที่ต้องระมัดระวังในการวัดด้วยวิธีนี้ คือ จุดต่อหรือหน้าคอนแทก ทุกจุดจะต้องมั่นใจว่าแน่นสนิท สะอาดปราศจากออกไซด์ ซึ่งจะทำให้ค่าที่อ่านได้ไม่แน่นอน และกัลป์วานอมิเตอร์ที่ใช้จะต้องเป็นแบบอิมพีแดนซ์สูง

2 เทอร์โมคัปเปิล (Thermocouple)

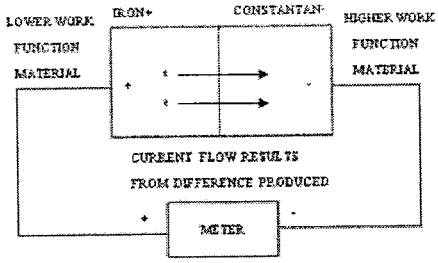
หนึ่งในหลายๆวิธีของการวัดอุณหภูมิในงานทางด้านวิทยาศาสตร์ และงานทางด้านอุตสาหกรรมก็คือ การใช้ผลของเทอร์โมคัปเปิล เทอร์โมคัปเปิลจัดได้ว่าเป็นทรานสดิวเซอร์ที่สามารถสร้างหรือผลิตแรงดันไฟฟ้าได้ด้วยตัวมันเอง ดังนั้นจึงสามารถใช้ได้โดยตรงกับเครื่องมือวัดหรือบันทึกอุณหภูมิ รวมทั้งเครื่องควบคุมต่างๆ เทอร์โมคัปเปิลจะทำให้เกิดแรงดันไฟฟ้าเมื่ออุณหภูมิเปลี่ยนแปลงไป

เทอร์โมคัปเปิลประกอบด้วยเส้นลวดโลหะต่างชนิดกันสองเส้นต่อเข้าด้วยกันที่ปลายข้างหนึ่ง ส่วนปลายอีกข้างหนึ่งจะถูกต่อไปใช้งาน ปลายของเส้นลวดที่ต่อเข้าด้วยกันนี้เรียกว่า Hot junction ส่วนปลายด้านหนึ่งเรียก Cold junction ซึ่งปลายด้านหนึ่งที่ต่อใช้งาน เมื่อจุดต่อ Hot junctionได้รับความร้อนและมีอุณหภูมิสูงขึ้นก็จะทำให้เกิดแรงดันไฟฟ้าที่สามารถวัดค่าได้ที่จุด Cold junction

ภาพที่ 4 การเชื่อมต่อของโลหะต่างชนิด

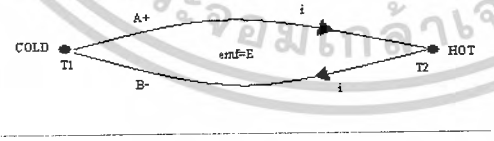
โลหะที่ใช้ประกอบกันเป็นเทอร์โมคับเปิลนั้นส่วนมากจะได้แก่ เหล็ก-คอนสแตนแตน โครเมล-อลูเมล พลาตินัม/โรเดียม-พลาตินัม และทองแดง-คอนสแตนแตน โดยชื่อของโลหะตัวแรกจะเป็นขั้วบวกทางไฟฟ้าส่วนโลหะชนิดหลังจะเป็นขั้วลบ พฤติกรรมที่ปรากฏของเทอร์โมคับเปิลนั้นสามารถอธิบายได้ด้วยผลการทดลองของ ซีเบ็ค (Seebeck Effect) เพลเทียร์ (Peltier Effect) และทอมสัน (Thomson Effect)

Thomas Seebeck นักวิทยาศาสตร์ชาวเยอรมัน เป็นผู้ค้นพบอุปกรณ์ที่สามารถให้แรงดันไฟฟ้าที่เกิดจากความร้อน (thermo – electric) ขึ้นเป็นครั้งแรกโดยไม่คาดคิดมาก่อนในเวลานั้น เขาได้นำลวดโลหะต่างชนิดกันมาต่อปลายเข้าด้วยกัน แสดงดังภาพที่3.4



ภาพที่ 5 การเกิดแรงดันในลวดโลหะ

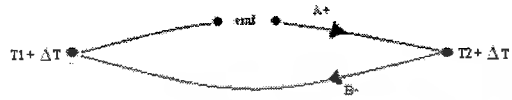
Seebeck ได้อธิบายผลทีเกิดขึ้นจากการทดลองว่า ถ้าให้ความร้อนที่ปลายข้างหนึ่งของเส้นลวดโลหะต่างชนิดกันที่ติดกันอยู่ กระแสที่ไหลจะเปลี่ยนแปลงไปตามค่าของอุณหภูมิที่แตกต่างกันระหว่าง hot junction กับ cold junction ซึ่งการไหลของกระแสเหล่านี้เกิดจากความปั่นป่วนของอิเล็กตรอนตรงจุดที่สัมผัสหรือได้รับความร้อน ซึ่งวัตถุที่ใช้นั้นจะต้องเป็นวัตถุที่มีแถบพลังงานไม่เท่ากัน ในที่นี้จะขอเรียกใหม่ว่า ฟังก์ชั่นการทำงาน วัตถุที่มีฟังก์ชั่นการทำงานที่ต่ำกว่า (lower work function material) จะให้อิเล็กตรอนได้เร็วกว่าวัตถุที่มีฟังก์ชันการทำงานที่สูงกว่า (higher work function material) ดังนั้นเมื่ออุณหภูมิเพิ่มขึ้น อิเล็กตรอนก็จะถูกปล่อยออกจากวัตถุที่มีฟังก์ชั่นการทำงานที่ต่ำกว่า ซึ่งจะเป็นผลทำให้เกิดเป็นขั้วบวกขึ้น ที่ระดับอุณหภูมิคงที่จำนวนอิเล็กตรอนที่ปลดปล่อยออกมาก็จะมีจำนวนที่คงที่และกระแสจะไหลสม่ำเสมอ เมื่ออุณหภูมิสูงขึ้นที่ถูกปลดปล่อยออกมาก็จะมากขึ้นกระแสก็จะไหลมากขึ้น จากปรากฏการณ์ดังกล่าวสามารถแสดงได้ด้วยภาพที่ 3.5



ภาพที่ 6 การไหลของกระแสตามทฤษฎีของ Seebeck

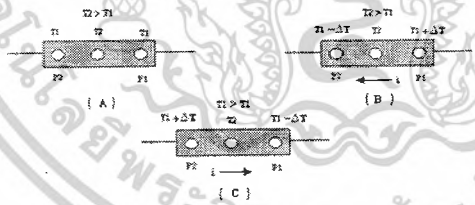
กระแสไฟฟ้าทีเกิดขึ้นจะคงไหลอยู่ตลอดเวลาตราบเท่าที่จุดต่อทั้งสองยังมีอุณหภูมิที่แตกต่างกันอยู่ แรงดันไฟฟ้าซึ่งเกิดจากการไหลของกระแสเหล่านี้ คือ แรงดันไฟฟ้าที่เกิดจากความร้อน (Seebeck themal emf.)

Jean C.A Peltier ได้ค้นพบว่าเมื่อกระแสผ่านจุดเชื่อมต่อของเส้นลวดที่ต่างชนิดกัน ที่ปลายข้างหนึ่งจะมีอุณหภูมิมี่เพิ่มขึ้น ในขณะที่ปลายอีกข้างหนึ่งอุณหภูมิจะลดลง กล่าวคือ ถ้ากระแสไหลผ่านจุดต่อในทิศทางหนึ่งความร้อนจะซึมซับเอาไว้ และความร้อนจะถูกปลดปล่อยออกมาในขณะที่กระแสไหลไปในอีกทิศทางหนึ่ง



ภาพที่ 7 การไหลของกระแสตามทฤษฎีของ Peltier

W.T.Thomson นักวิทยาศาสตร์ชาวอังกฤษพบว่า เมื่ออุณหภูมิแตกต่างกันภายในเส้นลวดเส้นเดียวกัน ก็สามารถที่จะทำให้เกิดแรงดันไฟฟ้าขึ้นได้ เขาพบว่าเมื่อกระแสไหลผ่านเส้นลวดทองแดงอุณหภูมิของแต่ละจุดไม่เท่ากัน ที่ จุด P ใดๆความร้อนจะลดลงถ้ากระแสที่ไหลผ่านนั้นไหลในทิศทางเดียวกันกับการไหลของความร้อน (กระแสจาก hot junction ไป cold junction) แต่ถ้ากระแสไหลในทิศทางตรงกันข้ามแล้วที่ จุด P จะมีความร้อนเพิ่มขึ้น



ภาพที่ 8 การไหลของกระแสตามทฤษฎีของ Thomson

ในรูป (A) เป็นแท่งโลหะที่ตรงจุดกึ่งกลางถูกทำให้มีอุณหภูมิ และจุด และ นั้นจะมีอุณหภูมิเป็นเท่ากัน โดยอุณหภูมิ มีค่ามากกว่าอุณหภูมิ เมื่อป้อนกระแสจากภายนอกให้แท่งโลหะในทิศทางตามรูป (B) อุณหภูมิที่จุด และ จะเปลี่ยนเป็น $+\Delta T$ และ $-\Delta T$ตามลำดับ แต่เมื่อกระแสไหลในทิศทางตรงกันข้ามอุณหภูมิที่จุด และ จะกลายเป็น $-\Delta T$ และ $+\Delta T$ตามลำดับเช่นกันดังแสดงในภาพ (c)

ผลจากการทดลองของ Thomsonนั้นต่างจากผลการทดลองของ Peltierตรงที่ว่าผลอันนี้เกิดขึ้นในตัวนำชนิดเดียวกันแทนที่จะเป็นจุดเชื่อมต่อของตัวนำต่างชนิด ดังนั้นแรงดันไฟฟ้าของ Seebeckก็คือผลรวมของปรากฏการณ์ที่ได้จากแรงดันไฟฟ้าPeltier ที่จุดเชื่อมต่อกับแรงดันไฟฟ้า Thomsonบนเส้นลวดที่ต่างกันทั้งสองเส้นถ้าให้ E เป็นแรงดันไฟฟ้าของ Thermocouple หรือแรงดันไฟฟ้าของ Seebeckแล้วเราจะได้ว่า

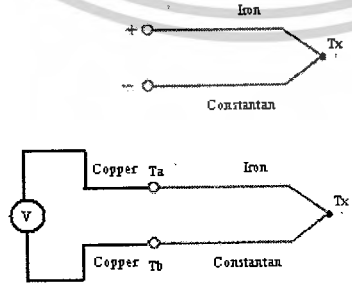
$$E = c(T_1 - T_2) + k(T_1^2 - T_2^2)$$

โดยที่ c,k = ค่าคงที่ของเทอร์โมคับเปิลแต่ละชนิด

2.1 การใช้งานของเทอร์โมคัปเปิล

RTD และเทอร์มิสเตอร์ จะวัดอุณหภูมิสัมบูรณ์แต่เทอร์โมคับเปิลจะวัดเพียงอุณหภูมิสัมพัทธ์ก็คือ การวัดอุณหภูมิใดๆแล้วต้องมาเปรียบเทียบกับอุณหภูมิอ้างอิงหรืออุณหภูมิที่ทราบค่า ทั้งนี้ก็เนื่องมาจากการทำงานของเทอร์โมคับเปิลที่อาศัยความแตกต่างของอุณหภูมิสองจุด

หากเราใช้เทอร์โมคับเปิลชนิด J ซึ่งประกอบด้วยเส้นลวดหนึ่งที่ทำมาจากเหล็ก (iron) และอีกเส้นหนึ่งทำมาจากคอนสแตนแตน (constantant) เมื่อเราต่อเทอร์โมคับเปิลชุดดังกล่าวนี้เข้ากับโวลต์มิเตอร์หรืออุปกรณ์ควบคุม สิ่งที่เกิดขึ้นเมื่อเราต่อสายทองแดงก็คือ เป็นการสร้างเทอร์โมคับเปิลขึ้นมาอีกสองเส้น ดังแสดงในรูป ซึ่งแต่ละเส้นย่อมมีผลต่อแรงดันของวงจร เท่ากับว่าตอนนี้เทอร์โมคับเปิลอยู่สามเส้นพร้อมอุณหภูมิที่ไม่รู้ค่าอีกสามจุด



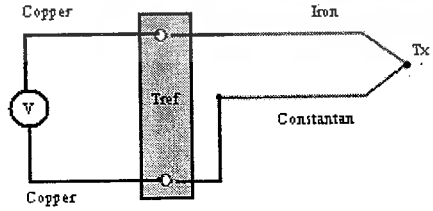
ภาพที่ 3.9 การต่อเทอร์โมคับเปิลอ้างอิงกับอุณหภูมิ

วิธีการดั้งเดิมในการแก้ปัญหาก็คือ การเพิ่มเทอร์โมคัปเปิลที่ตรงกันข้ามและจุดต่ออ้างอิงที่ทราบอุณหภูมิดังแสดงในภาพที่ 3.9 ในตัวอย่างนี้เทอร์โมคัปเปิลที่ตรงกันข้ามคือจุดต่อ ทองแดง – เหล็ก (copper-iron) อีกอันหนึ่งเพื่อให้คู่กับจุดต่อ ทองแดง- เหล็ก ที่เกิดขึ้นเมื่อต่อลวดทองแดงเข้ากับลวดเหล็กของเทอร์โมคัปเปิล ที่แท้จริงแล้วจุดต่อสองจุดนี้จะลบล้างซึ่งกันและกันอย่างมีประสิทธิภาพ ถ้ามันถูกแยกให้อยู่ในกล่อง ที่แยกอยู่ต่างหาก (isothermal block)

เพราะฉะนั้นในขณะนี้จะมีจุดต่อเพียงสองจุด จุดต่อแรกมาจากเทอร์โมคัปเปิล (Tx) และจุดต่ออ้างอิง (Tref) เป็นจุดที่สร้างขึ้นมา และหากเราทราบค่าอุณหภูมิที่จุดอ้างอิงนี้ก็สามารถที่จะคำนวณหาค่าอุณหภูมิ (Tx) ที่จุดวัดได้

เนื่องจากจุดอ้างอิงอุณหภูมิที่ใช้งานได้มีน้อยมากและมักจะมีราคาแพง จุดเยือกแข็งและจุดเดือดของน้ำที่อุณหภูมิ 0°C และ 100° C ดูเป็นทางออกที่ง่ายที่สุดที่ธรรมชาติหยิบยื่นมาให้ วิธีการทั่วไปที่ใช้ในการกำหนดอุณหภูมิ Tref หรืออุณหภูมิอ้างอิงก็คือ การวางจุดต่อในอ่างน้ำแข็ง วิธีการใช้อ่างน้ำแข็งแม้จะให้ผลการวัดที่เที่ยงตรงก็จริงแต่ไม่ใช่วิธีที่สะดวกที่สุด

เพื่อให้กระบวนการทุกอย่างง่ายขึ้น คือ ตัดอ่างน้ำแข็งออกไป แล้ววัดอุณหภูมิอ้างอิงด้วยอุปกรณ์วัดอุณหภูมิสัมบูรณ์ เช่น RTD หรือเทอร์มิสเตอร์ และชดเชยผลการวัดด้วยวิธีการทางคณิตศาสตร์ ความจำเป็นในการบังคับให้อุณหภูมิอ้างอิงเป็นศูนย์ก็หมดไป ขั้นตอนถัดไปก็คือ การกำจัดเทอร์โมคัปเปิลตัวที่สอง โดยการขยายกล่องแยกอุณหภูมิ (isothermal block) ให้รวมอุณหภูมิอ้างอิงเข้าไปด้วยดังแสดงในรูป ทำให้สามารถตั้งอุณหภูมิของกล่องเป็นอุณหภูมิอ้างอิง Tref (เนื่องจากเทอร์โมคัปเปิลทั้งสองที่อยู่ในกล่องยังคงหักล้างซึ่งกันและกัน) ณ.จุดนี้การหาค่าอุณหภูมิอ้างอิง Tref จึงเป็นเพียงเรื่องของการวัดอุณหภูมิของกล่องแยกอุณหภูมิด้วย RTD หรืออุปกรณีวัดค่าอุณหภูมิสัมบูรณ์อื่นๆเท่านั้น



ภาพที่ 10 การต่อเทอร์โมคัปเปิลอ้างอิงกับTref

Tref หรืออุณหภูมิอ้างอิงเป็นหนึ่งในสองปริมาณที่ต้องใช้เพื่อคำนวณหาค่า Tx ส่วนอีกค่าหนึ่งที่ต้องทราบคือ แรงดัน (E) ซึ่งสามารถวัดได้ด้วยโวลต์มิเตอร์จากสมการ E = c (Tx-Tref)+k(เมื่อ c และ k คือค่าคงที่ของเทอร์โมคัปเปิลแต่ละชนิด (หรืออาจใช้ตามตารางมาตรฐานของเทอร์โมคัปเปิลแต่ละชนิดก็ได้)แต่ถึงอย่างไรก็ตามแต่ c และ k รวมทั้งแรงดันเอาต์พุตที่ได้มักจะเป็นตัวเลขน้อยๆทำให้เป็นการยากที่จะวัดค่าสัมบูรณ์ และการเปลี่ยนแปลงสัมพัทธ์ได้อย่างถูกต้อง

เนื่องจากเทอร์โมคัปเปิล จะผลิตแรงดันไฟฟ้าที่เป็นสัดส่วนโดยตรงกับค่าความต้านทานของอุณหภูมิระหว่าง hot junction และ cold junction ดังนั้นจึงต้องระมัดระวังในการเลือกขนาดของวัตถุและหลอดป้องกันหรือฝักโลหะ (sheath) อย่างถูกต้อง การส่งถ่ายความร้อนอาจจะช้าเกินไปถ้า hot junction ใหญ่เกินไป หรือฝักโลหะที่มีขนาดไม่ถูกต้องจะทำให้ผลการตอบสนองทางไฟฟ้าเข้าไปด้วย hot junction จึงควรมีขนาดเล็กที่สุดเท่าที่เป็นได้ และหลอดป้องกันหรือฝักโลหะก็ควรจะมีการลดทอนและช่วงการหน่วงน้อยที่สุดด้วย