รายการอ้างอิง

ภาษาไทย

สุเธียร เกียรติสุนทร, *พื้นฐานวิศวกรรมระบบควบคุมในกระบวนการอุตสาหกรรม*, สมาคมส่งเสริมเทคโนโลยี(ไทย-ญี่ปุ่น), 2537

ภาษาอังกฤษ

Czogala, E., T. Rawlik, "Modeling of a Fuzzy Controller with Application to the Control of Biological Process," *Fuzzy Set and Systems*, Vol. 31, No. 1, pp. 13-22, 1989.

Couillard, D. and S. Zhu, "Control Strategy for Activated Sludge Process Under Shock Loading," *Water Research*, Vol. 20, No. 5, pp 649-655, 1992.

Klir, G.J. and T.A. Fougler, *Fuzzy Set, Uncertainty and Information*, Prentice-Hall International, 1988.

Harris, C.J., C.G. Moore and M. Brown, Intelligent Control, *Aspects of Fuzzy Logic and Neural Nets*., World Scientific publishing, Singapore, 1993.

———., "Intelligent Identification and Control for Autonomous Guided Vehicles Using Adaptive Fuzzy Based Algorithms," *Eng. Applic. of AI*, Vol. 2, pp. 267-285, 1989.

Hang, F.C., W.K. Ho and T.H. Lee, "Intelligent Control," *Journal of The Institution of Engineers Singapore*, Vol. 33, No. 3, 33-34, 1993.

Hanakuma, Y., Y. Irizuki, M. Adachi and E. Nakanishi, "Design of a Self Tuning Fuzzy Control System and The Application to a Distillation Column," *International Chemical Engineering*, Vol. 34, No.1, pp. 91-96, 1994.

Holmblad, L.P. and J. J. Ostergaard, "Control of a Cement Kiln by Fuzzy Logic," *Fuzzy Information and Decision Process*, Gupta, M. M., and Sanchez, E., eds., North-Holland, 1982.

Kickert, W.J.M. and H.R. Van Nauta Lemke, "Application of a Fuzzy Control in a Warm Water Plant," *Automatica*, Vol. 12, pp. 301-308, 1984.

Koffman, S.J., R.R. Fullmer and R.C. Brown, "Fuzzy Logic Control of a Fluidized Bed Combustor," *Proc. 1989 Am.Control Conf.*, Vol. 3, pp. 2756-2758, 1989.

Kuipers, B. and K. Astroms, "The Composition and Validation of Heterogenrous Control Laws," *Automatica*, Vol. 30, pp. 223-249, 1994.

Likens, D.A. and S.B.Hasnain, "Self-Organizing Fuzzy Logic Control and Application to Muscle Relaxant Aneasthesia," *IEE Proceedings-D*, Vol. 138, pp. 274-284, 1991.

Ling, C. and T. Edgar, "A New Fuzzy Gain Scheduling Algorithm for Process Control," *Proc. of ACC'92*, Chicago, pp. 2284-2290, 1992.

Mamdani, E.H. and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. Man-Machine Studies*, Vol. 7, pp. 1-13, 1975.

Jamshidi, M., N. Vadiee and T. J. Ross, *Fuzzy Logic and Control, software and Hardware Applications*, Prentice-Hall International, New Jersey, 1993.

Postlewaite, B.E., "Fuzzy State Estimator for Fed-Batch Fermentation," *Chemical Engineering Research & Design*, Vol. 67, No. 3, pp. 267-272, 1989.

Qin, S.J. and G. Borders, "A Multi-region Fuzzy Logic Controller for Controlling Process with Nonlinear Gains," *Proceeding of the 1993 International Symposium on Intelligent Control*, pp. 445-450, 1993.

Seborg, D.E., T.F. Edgar and D. A. Mellichamp, *Process Dynamics and Control*, John Wiley and Sons, New York, 1989.

Shah, I., and K. Rajamani, "Fuzzy Logic Controller: Application to Liquid Level System," *Minerals and Metallurgical Processing*, pp. 186-192, 1988.

_____., "Self-Organizing Controller for Process pH Control," *Control 90 Miner Metal Process Minerals & Metalurgical Processing*, pp. 45-52, 1990.

Stoll, K.E., P.A. Ralston and S. Ramanganesan, "Simplify Fuzzy Logic Control Implementation," *Hydrocarbon Processing*, pp. 49-55, 1993.

Tong, R.M., "An Assessment of a Fuzzy Control Algorithm for a Nonlinear Multivariable," *Tech. Report EES-MMS-DSFR-76 Queen Mary Collage*, London, 1976.

_____., M.B. Beck and A. Latten, "Fuzzy Control of Activated Sludge Waste Water Treatment Process," *Automatica*, Vol. 16, pp. 695-702, 1980.

Umbers, J.G. and P.J. King, "An Analysis of Human Decision in Cement Kiln Control and Implication for Automation," *Int. J. Man-Machine Studies*, Vol. 12, No. 1, pp. 11-13, 1980.

Van, Amerongen J., H. R. Van Nauta Lemke and J. C. Van Der Veen, "An Autopilot for ships Designed with fuzzy Sets in Digital Computer Applies to Process Control," Van Naute Lemke (Ed.), North Holland, 1977.

Viot. G., "Fuzzy Logic in C; Creating a Fuzzy-Based Inference Engine," *Dr. Dobb's Journal* pp. 40-49, 1993.

Walter, H.B., J.M. Robert and S.S. Sam, "Design of a Self-Tuning Rule Based Controller for a Gasoline Refinery Catalytic Reformer," *IEEE Trans. on Automatic Control*, Vol.35, pp.156-164, 1990.

Yan, J., M. Ryan and J. Power, *Using Fuzzy Logic*, Prentice-Hall International, 1994.

Yamashita, Y., S.Mitsumoto and M. Suzuki, "Start-Up of a Catalytic Reactor by Fuzzy Controller," *Jurnal of Chemical Engineering of Japan*, Vol. 21, No. 3, pp. 277-282,1988.

Zadeh, L.A., "Interpolative Reasoning in Fuzzy Logic and Neural Network Theory," *Proc. of 1st IEEE Int. Conf. on Fuzzy Syst.*, pp. 8-12, 1992.

Zhang, B.S.and J.M.Edmunds, "Self-Organizing Fuzzy Logic Controller," *IEE Proceedings-D*, Vol. 139, No. 5, pp. 460-464, 1992.

Zheng, Li., "A Practical Guide to Tune of Proportional Integral (PI) Like Fuzzy Controller," *Proc. of 1st IEEE ICFS (March 8-12)*, pp. 636-640. 1992.

ภาคผนวก ก.
การอินเตอร์เฟซสัญญาณด้วยการ์ด PCL-812PG

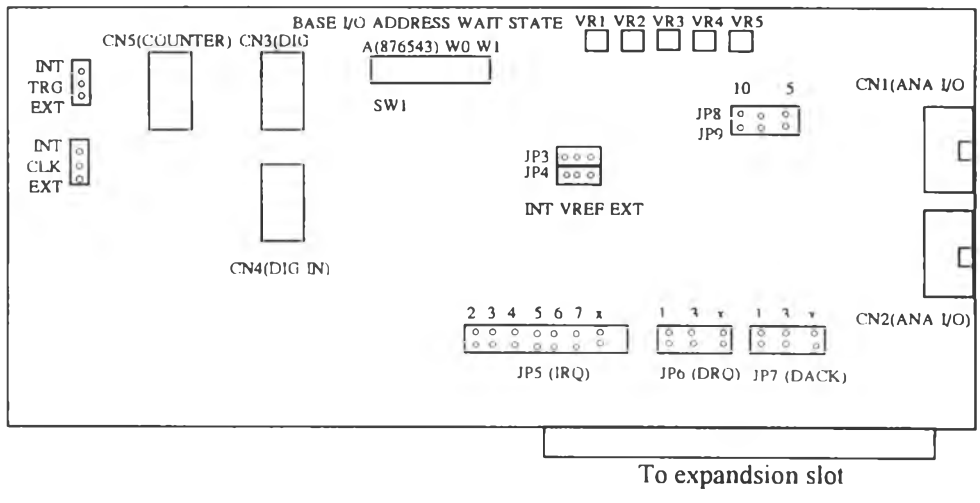
การทำงานของโปรแกรมตัวควบคุมมีความจำเป็นที่ต้องมีการเก็บข้อมูลจากกระบวนการโดยรับสัญญาณไฟฟ้าที่ได้จากตัววัดระดับในรูปของสัญญาณอะนาล็อกและแปลงสัญญาณนี้ให้อยู่ในรูปของสัญญาณดิจิตัล เพื่อใช้เป็นข้อมูลอินพุทของโปรแกรมตัวควบคุม หลังจากทำการประมวลผลแล้วสัญญาณเอาท์พุทจากโปรแกรมตัวควบคุมจะถูกส่งออกมาในรูปดิจิตัล สัญญาณนี้จะยังนำไปใช้โดยตรงไม่ได้ เนื่องจากอุปกรณ์ที่เกี่ยวข้องทั้งหมดสามารถรับสัญญาณในรูปแบบอะนาล็อกได้เพียงอย่างเดียว จึงต้องมีการเปลี่ยนสัญญาณดิจิตัลนี้ให้อยู่ในรูปสัญญาณอะนาล็อกเสียก่อนจึงจะนำไปใช้งานได้ กระบวนการเก็บ-ส่งข้อมูลและแปลงสัญญาณทั้งหมดได้ถูกรวมไว้ในโมดูลของการ์ดเก็บข้อมูลในรุ่น PCL-812PG ชุดการ์ดนี้จะประกอบด้วยส่วนสำคัญสองส่วนคือ

1. ตัวการ์ด PCL-812PG
2. ซอฟท์แวร์ไดรเวอร์และไฟล์ไลบรารี

ขั้นตอนการติดต่อใช้งานของการ์ดที่จำเป็นสำหรับงานวิจัยมีดังนี้

ก.1 ลักษณะของการ์ด PCL-812PG ตำแน่งสวิทซ์และจุดเชื่อมต่อ

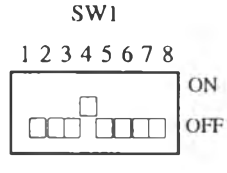
ลักษณะของการ์ดจะเหมือนกับการ์ดอุปกรณ์เชื่อมต่อทั่วไป การติดตั้งทำได้ง่ายโดยเสียบการ์ดนี้เข้ากับช่องที่ไม่ได้ใช้งานบนบอร์ดคอมพิวเตอร์ (expandsion slot) ในขั้นตอนต่อไปคือการจัดตั้งสวิทซ์ต่างๆ ให้สอดคล้องกับโปรแกรมใช้งาน ตำแหน่งของสวิทซ์ต่างๆ แสดงในรูปที่ ก.1



รูปที่ ก.1 ตำแหน่งสวิทซ์และจัมพ์เปอร์ต่างๆ ของ PCL-812PG

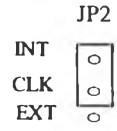
ก่อนการใช้งานให้ตั้งตำแหน่งสวิทซ์และจัมพ์เปอร์ดังนี้

SW1 (Base I/O address): 0x220

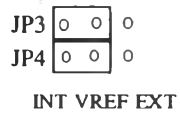


JP1 (Trigger sorce selection): internal

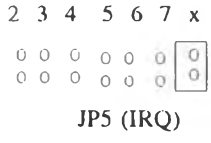
JP2 (Input clock selection): internal



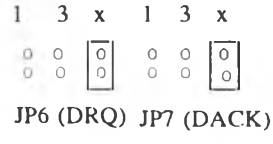
JP3, JP4 (D/A reference source selection): 10 V.



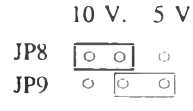
JP5 (IRQ level selection): No interupt



JP6, JP7 (DMA chanel selection): No DMA



JP8 (D/A internal reference selection): 10 V.



JP9 (A/D maximum input voltage selection): +/- 5V.

จุดเชื่อมต่อสัญญาณบน PCL-812PG ทั้งหมดมี 5 ชุด คือ

conector 1 (CN1) - Analog input (Single-ended chanels)

conector 2 (CN2) - Analog output

conector 3 (CN3) - Digital output

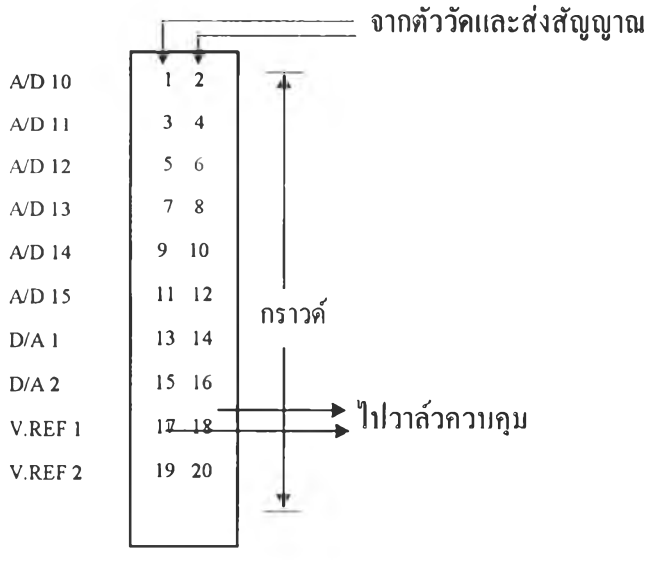
conector 4 (CN4) - Digital input

conector 5 (CN5) - counter

ในกรณีที่อุปกรณ์วัดและควบคุมทั้งหมดเป็นแบบอะนาล็อก ดังนั้นจุดเชื่อมต่อสัญญาณที่จำเป็นต้องใช้คือ CN1 หรือ CN2 เท่านั้น จำนวนอินพุทที่การ์ดสามารถรับได้ทั้งหมดคือ 16 ช่องสัญญาณ และส่งออกเป็นอะนาล็อกได้ 2 ช่องสัญญาณ ในงานวิจัยนี้ต้องการวัดเพียงระดับของเหลวภายในถังและส่งสัญญาณควบคุมไปยังวาล์วควบคุมเพียงอย่างละ 1 ช่องสัญญาณเท่านั้น ดังนั้นจึงเลือกจุดเชื่อมต่อสัญญาณที่ CN2 โดยแสดงตำแหน่งการเชื่อมต่อคือ

A/D at chanel : 10

D/A at chanel :1



CN 2

ก.2 ซอฟท์แวร์ไดรเวอร์และไฟล์ไลบรารี

การใช้งานการ์ดนี้จะทำได้โดยการสั่งงานจากไลบรารีผ่านโปรแกรมไดรเวอร์ไปสู่การ์ด ดังนั้นจึงจำเป็นต้องมีไฟล์ไลบรารีและไฟล์ไดรเวอร์ประกอบการใช้งานเสมอ สำหรับซอฟท์แวร์ที่ประกอบกับชุดการ์ดจะประกอบไปด้วยไฟล์ต่างๆ ที่จำเป็นดังนี้

ไฟล์ไดรเวอร์ ประกอบด้วย

PCL-812.EXE: ทำหน้าที่เป็นไดรเวอร์ติดต่อระหว่างโปรแกรมใช้งานกับการ์ดในลักษณะโปรแกรมเรสซิเดนท์ (Resident Program) ซึ่งจะแขวนอยู่ในหน่วยความจำตลอดเวลาการใช้งาน

FREE812.EXE : ทำหน้าที่ยกเลิกการใช้งานโปรแกรมไดรเวอร์และการ์ด

ไฟล์ไลบรารี ประกอบด้วยไฟล์ที่อยู่ในไดเรคทอรีต่างๆ คือ

ไดเรคทอรี BAS เป็นไดเรคทอรีที่เก็บไฟล์ไลบรารีภาษาเบสิค ประกอบด้วยไฟล์

812qb.lib 812qb.obj 812qb.qlb และ 812bas.bin

ไดเรคทอรี C เป็นไดเรคทอรีที่เก็บไฟล์ไลบรารีภาษาซี ประกอบด้วยไฟล์

812cs.lib 812cm.lib 812cc.lib และ 812cl.lib

ไดเรคทอรี PAS เป็นไดเรคทอรีที่เก็บไฟล์ไลบรารีภาษาปาสคาล ประกอบด้วยไฟล์

812mps.obj 812tpf.obj และ 812tpn.obj

ไฟล์ไดรเวอร์และไฟล์ไลบรารีต่างๆ เป็นส่วนสำคัญที่จะทำให้สามารถอินเตอร์เฟซสัญญาณได้ ซึ่งลักษณะและวิธีการใช้งานจะได้แสดงต่อไป

ก. 3 ขั้นตอนการอินเตอร์เฟซสัญญาณ

การทำงานของ PCL-812PG เป็นการสั่งงานผ่านรูทีนของไลบรารีที่ให้มากับซอฟท์แวร์ของการ์ด ซึ่งสนับสนุนในหลายภาษาคือ ซี เบสิค ปาสคาล ในงานวิจัยนี้จะเลือกใช้ไลบรารีภาษาซี ในขั้นแรกจำเป็นต้องทำการติดตั้งโปรแกรมไดรเวอร์ซึ่งเป็นโปรแกรมเรซิเดนท์ ซึ่งทำ

หน้าที่เชื่อมต่อระหว่างโปรแกรมใช้งานฟังก์ชันต่างๆ ของไดรเวอร์กับการ์ด ทำได้โดยเรียกชื่อไฟล์ PCL-812.EXE โปรแกรมนี้จะรันอยู่ในหน่วยความจำของเครื่องตลอดเวลาเพื่อรอรับคำสั่งต่างๆ จากนั้นให้รวมไลบรารีของไดรเวอร์ 812CL.LIB เข้าไว้ในโปรเจคไฟล์ หลังจากนั้นจะเป็นการเรียกใช้งานฟังก์ชันต่างๆ ที่จะให้การ์ดทำงาน ขั้นตอนการเรียกใช้งานการ์ดที่สำคัญคือการเก็บข้อมูล และการส่งข้อมูลออกซึ่งมีรายละเอียดดังนี้

ก การเก็บข้อมูล

ในการติดต่อกับ PCL-812PG ทุกครั้งจะทำโดยการสั่งงานผ่านรูทีน pcl812(func,param) โดย

func : หมายถึงหน้าที่ของการ์ดที่จะต้องทำ เช่น อ่านข้อมูลเข้า หรือส่งออก

param : หมายถึงพารามิเตอร์ที่จำเป็นในการใช้งานตามหน้าที่นั้นๆ เช่น ตั้ง Base I/O address, DMA level, IRQ level ซึ่งค่าเหล่านี้จะต้องสอดคล้องกับสวิทช์และจัมพ์เปอร์ต่างๆ ที่ตั้งไว้บนการ์ด

การอ่านข้อมูลเข้าทำตามขั้นตอนดังนี

1. เริ่มต้นระบบด้วยการเตรียมหน่วยความจำที่จำเป็นโดยเรียก รูทีน pcl812(func, param)

func = 3 (Hardware initialization) การเริ่มต้นระบบนี้ทำเพียงครั้งเดียวเท่านั้น

2. อ่านข้อมูลเข้าด้วยการตั้งค่าพารามิเตอร์ดังนี้

param[1] = 0x220 (Base I/O address)

param[14] = 50 (A/D conversion number)

param[15] = 0xA (A/D conversion start at chanel 10)

param[16] = 0xA (A/D conversion stop at chanel 10)

param[17] = 0 (Overall gain code 0: +/- 5 V.)

ตั้งค่า func = 4 (A/D initialization)

เรียกรูทีน pcl812() เป็นการเริ่มต้นการแปลงข้อมูล A/D

จากนั้นตั้ง func = 5 (perform A/D conversion)

เรียกรูทีน pcl812() เพื่ออ่านและแปลงข้อมูล ข้อมูลที่อ่านได้จะถูกเก็บไว้ในตัวแปร data ในรูปข้อมูล 12 บิตจากนั้นทำการเปลี่ยนข้อมูลนี้ให้เป็นเลขฐานสิบได้โดย

DataBuf = data&0xFFF

DataBuf = ((5-(-5))+DataBuf/4096)+(-5)

โดย DataBuf : ข้อมูลในรูปเลขฐานสิบ

data : ข้อมูลในรูป 12 บิต

4096 : ค่าฟูลสเกลข้อมูล 12 บิต

((5-(-5)) : A/D input range

ทำการหาค่าเฉลี่ยข้อมูลทั้งหมดโดยหาผลรวมข้อมูลแล้วหารด้วยจำนวนครั้งของการเก็บข้อมูล (ในที่นี้คือ 50 ครั้ง)

ข. การส่งข้อมูลออก

กรณีต้องการส่งข้อมูลออกตั้งค่าพารามิเตอร์ที่จำเป็นดังนี้

param[1] = 0x220 (Base I/O address)

param[24] = 1 (D/A conversion number)

param[25] = 0 (D/A conversion start at chanel 1)

param[26] = 0 (D/A conversion stop at chanel 1)

แปลงข้อมูลที่จะส่งออกให้อยู่ในรูปข้อมูล 12 บิตแล้วกำหนดให้อยู่ในตัวแปร data

ด้วยสมการ

data = volt out*4096/full scale volt

ตั้งค่า func = 12 (D/A initialization)

เรียกรูทีน pcl812() เป็นการเริ่มต้นการแปลงข้อมูล D/A

จากนั้นตั้ง func = 13 (N times of D/A conversion)

เรียกรูทีน pcl812() ข้อมูลแบบอะนาล็อกจะถูกส่งออกจากการ์ดที่ CN1

ตัวอย่างการเรียกใช้งานฟังก์ชัน

หลังจากทำการตั้งค่าพารามิเตอร์ param ตามเงื่อนไขการใช้งานที่กล่าวไว้แล้ว การเรียกใช้งานฟังก์ชันทำได้ดังนี้

1. เริ่มต้นระบบด้วยฟังก์ชัน 3

pcl812(3,param); (Func 3 : Hardware initialization)

2. อ่านข้อมูลเข้าด้วยฟังก์ชัน 4, 5

```
pcl812(4,param);        (Func 4 : A/D initialization)

pcl812(5,param);        (Func 5 : Perform A/D conversion)
```

3. ส่งข้อมูลออกด้วยฟังก์ชัน 12, 13

```
pcl812(12,param);       (Func 12 : D/A initialization)

pcl812(13,param);       (Func 13 : D/A output)
```

การเริ่มต้นระบบในขั้นตอนที่ 1 จะทำเพียงครั้งเดียวเสมอ คือในช่วงเริ่มต้นทำงานเครื่อง จากนั้นจึงทำในขั้นตอนที่ 2 รับข้อมูลเข้าเพื่อนำไปใช้งาน จากนั้นเป็นขั้นตอนที่ 3 ซึ่งเป็นการส่งข้อมูลออกจากการ์ด ในกรณีที่การอ่านข้อมูลเข้าและส่งออกเป็นลักษณะลูพ เช่น โปรแกรมตัวควบคุมนี้ สามารถทำได้โดยการวนลูพกลับไปที่ขั้นตอนที่ 2 และ 3 ต่อไปได้เรื่อยๆ

ภาคผนวก ข.

ซอร์สโค้ด

ข.1 ไฟล์โปรแกรมอินเตอร์เฟซการ์ด พีซีแอล-812พีจี

```
// PCL-812PG driver section   (pcldrv.c)
void InitPCL812PG(void)
// Initialization of PCL-812PG card
 { /* PCL-812PG card setting
        Base I/O address : 0x220
        DMA A,B Level : X
        IRQ Level   : X
        N times Conversion : 50
        Start Channel : 10
        Stop Channel  : 10
        Gain Code   : 0 (Gain x1)  */
   dat = data;
   param[0] = 0;          /* Board number                */
   param[1] = 0x220;      /* Base I/O address            */
   param[2] = 0x0;        /* DMA level                   */
   param[3] = 0x0;        /* DMA level                   */
   param[4] = 0x0;        /* IRQ level  : IRQ2           */
   param[5] = 50;         /* Pacer rate = 2M / (50 * 100) = 400 Hz  */
   param[6] = 100;
```

```
    param[7] = 0;       /* Trigger mode, 0 : pacer trigger          */
    param[8] = 0;       /* Non-cyclic                               */
    param[10] = FP_OFF(dat); /* Offset of A/D data buffer A         */
    param[11] = FP_SEG(dat); /* Segment of A/D data buffer A        */
    param[12] = 0;      /* Data buffer B address, if not used,      */
    param[13] = 0;      /* must set to 0.                           */
    param[14] = 50;     /* A/D conversion number                    */
    param[15] = 0xA;    /* A/D conversion start channel             */
    param[16] = 0xA;    /* A/D conversion stop channel              */
    param[17] = 0;      /* Overall gain code, 0 : +/- 5V            */
    /* param[18] = FP_OFF(gain_array);
          param[19] = FP_SEG(gain_array); */
    /* param[45] : Error code
          param[46] : Return value 0
          param[47] : Return value 1 */
    pcl812(3, param);   /* Func 3 : Hardware initialization         */
    if (param[45] != 0) {
          printf("\nDRIVER INITIALIZATION FAILED !");
          printf("\nPress Any Key!");
          getch();
          exit(1);
          }
  }
float GetInputAD(void)
// Getting Analog Signal Input Form the Process
 { int i;
```

```
float volt_in; // Reading signal in the form of 0-10 Volt.
float read_in = 0.00;
float DataBuf;
dat = data;
  param[0] = 0;          /* Board number                    */
  param[1] = 0x220;      /* Base I/O address                */
  param[2] = 0x0;        /* DMA level                       */
  param[3] = 0x0;        /* DMA level                       */
  param[4] = 0x0;        /* IRQ level : IRQ2                */
  param[5] = 50;         /* Pacer rate = 2M / (50 * 100) = 400 Hz   */
  param[6] = 100;
  param[7] = 0;          /* Trigger mode, 0 : pacer trigger */
  param[8] = 0;          /* Non-cyclic                      */
  param[10] = FP_OFF(dat); /* Offset of A/D data buffer A   */
  param[11] = FP_SEG(dat); /* Segment of A/D data buffer A  */
  param[12] = 0;         /* Data buffer B address, if not used,  */
  param[13] = 0;         /* must set to 0.                  */
  param[14] = 50;        /* A/D conversion number           */
  param[15] = 0xA;       /* A/D conversion start channel    */
  param[16] = 0xA;       /* A/D conversion stop channel     */
  param[17] = 0;         /* Overall gain code, 0 : +/- 5V   */
  /* param[18] = FP_OFF(gain_array);
        param[19] = FP_SEG(gain_array); */
  /* param[45] : Error code
        param[46] : Return value 0
        param[47] : Return value 1 */
```

```
pcl812(4, param);       /* Func 4 : A/D initialization          */
if (param[45] != 0) {
        closegraph();exit(0);
        printf("\nA/D INITIALIZATION FAILED !");
        printf("\nPress Any Key!");
        getch();
        exit(1);
        }
pcl812(5, param);       /* Func 5 : Perform A/D conversion */
                        /* with software data transfer          */
if (param[45] != 0) {
        closegraph();
        printf("\nA/D PACER TRIGGER WITH DATA TRANSFER FAILED !");
        printf("\nPress Any Key!");
        getch();
        exit(1);
        }
for (i = 0; i < param[14]; i++)
      {
        DataBuf = data[i] & 0xFFF;
        DataBuf =( (5 - (-5)) * DataBuf / 4096) + (-5);
        /*
          (10 - (-10)) : A/D input range (-10V to 10V)
          4096     : Full scale 12 bit A/D data
          DataBuf  : A/D input data
          (-10)    : A/D input range "-10" V  */
```

```
            read_in = read_in + DataBuf;
        }
        volt_in = 2*(read_in/param[14]);
        return volt_in;
}
void SendOutDA(float volt_out)
{   dat = data;
    param[0] = 0;          /* Board number                    */
    param[1] = 0x220;      /* Base I/O address                */
    param[20] = FP_OFF(dat); /* Offset of D/A output data buffer A   */
    param[21] = FP_SEG(dat); /* Segment of D/A output data buffer A  */
    param[22] = 0;         /* Output data buffer B address, if not used  */
    param[23] = 0;         /* must set to 0.                  */
    param[24] = 1;         /* D/A conversion number           */
    param[25] = 0;         /* D/A conversion start channel    */
    param[26] = 0;         /* D/A conversion stop channel     */
    /* param[45] : Error code
        param[46] : Return value 0
        param[47] : Return value 1 */
    pcl812(12, param);     /* Func 12 : D/A  initialization   */
    if (param[45] != 0) {
        closegraph();printf("\007");
        printf(" D/A INITIALIZATION FAILED !");
        printf("\n Press any key!");
        getch();
        exit(1);
```

```
            }
  data[0] = volt_out*4095/10;
   pcl812(13, param) ;      /* Func 13 : "N" times of D/A output    */
   if (param[45] != 0) {
            closegraph();printf("\007");
            printf(" D/A OUTPUT FAILED !");
            printf("\n Press any key!");
            getch();
            exit(1);
            }
 }
```

ข.2 ไฟล์ส่วนหัวของโปรแกรมตัวควบคุม

```
////////////////////////////////////////////////////////////
// Global Head Section  (ghder.h)
////////////////////////////////////////////////////////////
#include <dos.h> #include <stdio.h> #include <conio.h>
#include <stdlib.h> #include <graphics.h> #include <string.h>
#include <math.h> #include <time.h>#include <stdarg.h> #include <alloc.h>
extern pcl812(int, unsigned int *);
#define MANUAL 0 // Manual mode
#define AUTO 1       // Auto mode
#define ON 1         // Bumpless transfer
#define OFF 0
#define se_limit 100    // Defined scaling factor full range is 100%
#define sce_limit 100
```

```
#define sdu_limit 100
//.....
float SptValue,
    ManValue;  // 0 - 100%
float DT = 6;
float se = 80,
    sce = 17,
    sdu = 38;       // Fuzzy variable scaling factors
int  mode;          // Current mode
int  Bumpless ,  // Bumpless transfer status
    MA;             // Transfer status : Manual to Auto
int  RESET_Status = 0 ; int CHGALG; // Change algorithm status
float pv_volt[2];     /*  PV, SP , MV variables    */
float sv_volt[2];
float mv_volt[2];     /* ------------------------- */
float delta_co = 0,
      delta_sv  = 0;
/* ---------- PCL 812PG ---------- */
unsigned int param[60];     /* If two boards installed, need to declare
                               the second parameter array      */
unsigned int data[60];     /* Conversion data buffer          */
unsigned int far * dat;
/* ------------------------------------------------------- */
#include "fuzzy.h"
```

ข.3 ไฟล์ส่วนหัวของอัลกอริธึมตัวควบคุมฟัซซีลอจิก

```
// Head file For Fuzzy Controller Defining Rule-Base Code & Membership Functions (fuzzy.h)
#define NB 0  #define NM 1  #define NS 2
#define ZE 3  #define PS 4  #define PM 5
#define PB 6  #define XX 7                    // Define fuzzy variables code
// Define fuzzy rule-bases  {IF e , ce, AV, THEN Δu}
int rule_base_code[75][4] = { {NB,NB,NB,NB}, {NB,NS,NB,XX}, {NB,ZE,NB,XX},{NB,PS,NB,XX},
                              {NB,PB,NB,NB}, {NS,NB,NB,NM}, {NS,NS,NB,NS}, {NS,ZE,NB,NS},
                              {NS,PS,NB,PS}, {NS,PB,NB,PM}, {ZE,NB,NB,NS}, {ZE,NS,NB,NS}
                              {ZE,ZE,NB,ZE}, {ZE,PS,NB,PS}, {ZE,PB,NB,PS}, {PS,NB,NB,PS},
                              {PS,NS,NB,PM}, {PS,ZE,NB,PM}, {PS,PS,NB,PB}, {PS,PB,NB,PB},
                              {PB,NB,NB,PB}, {PB,NS,NB,XX}, {PB,ZE,NB,XX}, {PB,PS,NB,XX},
                              {PB,PB,NB,PB}, {NB,NB,ZE,NB}, {NB,NS,ZE,XX}, {NB,ZE,ZE,XX},
                              {NB,PS,ZE,XX}, {NB,PB,ZE,NB}, {NS,NB,ZE,NB}, {NS,NS,ZE,NS},
                              {NS,ZE,ZE,NS}, {NS,PS,ZE,NS}, {NS,PB,ZE,PS}, {ZE,NB,ZE,NM},
                              {ZE,NS,ZE,NS}, {ZE,ZE,ZE,ZE}, {ZE,PS,ZE,PS}, {ZE,PB,ZE,PM},
                              {PS,NB,ZE,NM}, {PS,NS,ZE,PS}, {PS,ZE,ZE,PS}, {PS,PS,ZE,PM},
                              {PS,PB,ZE,PB}, {PB,NB,ZE,PB}, {PB,NS,ZE,XX}, {PB,ZE,ZE,XX},
                              {PB,PS,ZE,XX}, {PB,PB,ZE,PB}, {NB,NB,PB,NB}, {NB,NS,PB,XX},
                              {NB,ZE,PB,XX}, {NB,PS,PB,XX}, {NB,PB,PB,NB}, {NS,NB,PB,NM},
                              {NS,NS,PB,NS}, {NS,ZE,PB,NS}, {NS,PS,PB,PS},{NS,PB,PB,PM},
                              {ZE,NB,PB,NS}, {ZE,NS,PB,NS},{ZE,ZE,PB,ZE}, {ZE,PS,PB,PS},
                              {ZE,PB,PB,PS}, {PS,NB,PB,NB}, {PS,NS,PB,PS}, {PS,ZE,PB,PS},
                              {PS,PS,PB,PS}, {PS,PB,PB,PM}, {PB,NB,PB,PB}, {PB,NS,PB,XX},
                              {PB,ZE,PB,XX}, {PB,PS,PB,XX},{PB,PB,PB,PB}, };

int num_mf_if=3; int num_mf_then=1;  // Define number of Antecedent & Consequent
```

```
int num_rules = 75;              : // Maximum rules are 75
float input[3];                   // 3 input variables
float sum_of_products = 0;        // Used in system output calculation
float sum_of_area   = 0;
// Defining shapes of membership functions
float mf_antic_point[2][5][4] = {  { /* point1 point2 point3 point4 */
                                  { -1.100,-1.000,-0.700,-0.200 }, // 0 NB
                                  { -0.750,-0.200,-0.150,-0.000 }, // 2 NS
 /* Error */                     { -0.100,-0.000, 0.000, 0.100 }, // 3 ZE
                                  { 0.000, 0.150, 0.200, 0.750 }, // 4 PS
                                  { 0.200, 0.500, 1.000, 1.100 }, // 6 PB
                                 },
                                 { /* point1 point2 point3 point4 */
                                  { -1.100,-1.000,-0.700,-0.200 },
                                  { -0.700,-0.300,-0.250, 0.000 },
 /* Change of error */           { -0.200,-0.000, 0.000, 0.200 },
                                  { 0.000, 0.250, 0.300, 0.750 },
                                  { 0.200, 0.700, 1.000, 1.100 },
                                 }
                                };
 float mf_aux_point[3][4] =              { /* point1 point2 point3 point4 */
                                  { -1.100,-1.000,-0.850,-0.300 }, // NB
                                  { -0.900,-0.800, 0.800, 0.900 }, // ZE
  /* Level */                    { 0.750, 0.850, 1.000, 1.100 },  // PB
                                 };
 float mf_conseq_point[7][4] = { /* point1 point2 point3 point4 */
```

```
                                { -1.100,-1.000,-1.000,-0.500 },
                                { -0.750,-0.500,-0.500,-0.250 },
                                { -0.400,-0.080,-0.080, 0.000 },
                                { -0.250,-0.000, 0.000, 0.250 },
   /* Δu(k) */                  { 0.000, 0.080, 0.080, 0.350 },
                                { 0.250, 0.400, 0.400, 0.750 },
                                { 0.500, 1.000, 1.000, 1.100 },
                                };
```

ข.4 ไฟล์อัลกอริธึมตัวควบคุมฟัซซี

```
/* .....
    3R FUZZY LOGIC Controller  (fuzzy.c)
  ..... */
float MIN(float val1,float val2)
// Min. operator
 {  float result;
    if(val1>val2)
   result = val2;
    else result = val1;
  return result;
 }
float ComputeDegreeOfMembership(float point1,float point2,float point3,float point4,float fuzzy_sys_input)
// Evaluation of degree of membership for Input variables
{           float delta1,delta2,
                  val1,val2;
            float slope1,slope2,
```

```
            mf_data_value;
        float Degree_Of_Fuzz;
         delta1 = fuzzy_sys_input - point1;
         delta2 = point4 - fuzzy_sys_input;
        if (point1 != point2)
         slope1 = (1-0)/(point2-point1);
          else slope1 = 1000;
         if (point3 != point4)
         slope2 = (0-1)/(point4-point3);
          else slope2 = -1000;
          val1 = fabs(slope1*delta1);
          val2 = fabs(slope2*delta2);
        if (delta1 <= 0 || delta2 <= 0)   // if input outside membership function
         mf_data_value = 0;              // then degree mf is 0
        else
         mf_data_value = MIN(val1,val2);
        Degree_Of_Fuzz = MIN(1.0,mf_data_value); // detect mf value based on 1.0
        return Degree_Of_Fuzz;
        }
   // Use Center-of-Gravity defuzzification
float ComputeAreaOfTrapesoid(float point1,float point2,float point3,float point4,float strength)
// Computation of trapesoidal area
   {        float base,top;
        float computed_area,
            slope1,slope2,
            point2_new,point3_new;
```

```
        // Correlation Minimum Inference Method is used
         if (point1 != point2)
          slope1 = (1-0)/(point2 - point1);
          else slope1 = 1000.00;  // Assign  slope1 approch  to  +infinity
            point2_new = (strength - 0)/slope1 + point1;
         if (point3 != point4)
          slope2 = (0-1)/(point4 - point3);
                else slope2 = -1000.00;  // Assign  slope1 approch  to  -infinity
          point3_new = -(0 - strength)/slope2 + point4;
        // Correlation minimum inference method....
          if(slope1 != 0 && slope2 != 0)
          {
           base = fabs(point4-point1);
           top  = fabs(point3_new - point2_new);
           computed_area = strength*(base+top)/2;
          }
          else computed_area = 0;
          return computed_area;
    }
float Antecedent_Operator(float fuzzy_value1,float fuzzy_value2)
// Fuzzy Logic Operator: (AND)
  {  float AND;
   AND = MIN(fuzzy_value1,fuzzy_value2);
   return AND;
  }
float Fuzzification(int Rule_No,int Fired_CrispInput_No)
```

```
// Fuzzification Unit.
{     float point1,point2,
            point3,point4,
            fuzzy_system_input;
    float Fuzzy_value;
    int mf_code;
if(Fired_CrispInput_No <= (num_mf_if-2)) {
 mf_code = rule_base_code[Rule_No][Fired_CrispInput_No];
    if(mf_code==2) mf_code = 1;
    else
    if(mf_code==3) mf_code = 2;
    else
    if(mf_code==4) mf_code = 3;
    else
    if(mf_code==6) mf_code = 4;
 point1 = mf_antic_point[Fired_CrispInput_No][mf_code][0]; /* Rule decoder */
 point2 = mf_antic_point[Fired_CrispInput_No][mf_code][1]; /* Rule decoder */
 point3 = mf_antic_point[Fired_CrispInput_No][mf_code][2]; /* Rule decoder */
 point4 = mf_antic_point[Fired_CrispInput_No][mf_code][3]; /* Rule decoder */
                                                    }
  else if(Fired_CrispInput_No == (num_mf_if-1)) {
 mf_code = rule_base_code[Rule_No][Fired_CrispInput_No];
    if(mf_code==0) mf_code = 0;
    else
    if(mf_code==3) mf_code = 1;
    else
```

```
    if(mf_code==6) mf_code = 2;
   point1 = mf_aux_point[mf_code][0]; /* Rule decoder */
   point2 = mf_aux_point[mf_code][1]; /* Rule decoder */
   point3 = mf_aux_point[mf_code][2]; /* Rule decoder */
   point4 = mf_aux_point[mf_code][3]; /* Rule decoder */
                                                    }
  fuzzy_system_input = input[Fired_CrispInput_No];          // Scan input vars.
  Fuzzy_value = ComputeDegreeOfMembership(point1,point2,point3,point4,fuzzy_system_input);
  // result is mf_data value
  return Fuzzy_value;
  }
void Rules_aggregation(int Rule_No,float strength)
// Aggregation of all fuzzy rules.
  {    float point1,point2,
              point3,point4;
    float defuzz_area,centroid,
              centroid1,centroid2,centroid3;
    int  conseq_code;
    conseq_code = 3; /* there is 1 element of THEN side */
  if(rule_base_code[Rule_No][conseq_code] != 7)
  {
  point1 = mf_conseq_point[rule_base_code[Rule_No][conseq_code]][0]; /* rule (THEN side) decoder */
  point2 = mf_conseq_point[rule_base_code[Rule_No][conseq_code]][1]; /* rule (THEN side) decoder */
  point3 = mf_conseq_point[rule_base_code[Rule_No][conseq_code]][2]; /* rule (THEN side) decoder */
  point4 = mf_conseq_point[rule_base_code[Rule_No][conseq_code]][3]; /* rule (THEN side) decoder */
    defuzz_area = ComputeAreaOfTrapesoid(point1,point2,point3,point4,strength);
```

```
    centroid1 = 2*(point2-point1)/3 + point1;
    centroid2 = (point3-point2)/2 + point2;
    centroid3 = point4 - 2*(point4-point3)/3 ;
    centroid = ( 0.5*1*(point2-point1)*centroid1 +
                    1*(point3-point2)*centroid2 +
                    0.5*(1)*(point4-point3)*centroid3 ) /
                    (0.5*((point4-point1)+(point3-point2))*1);
    sum_of_products = sum_of_products + defuzz_area*centroid;
   }
   if (rule_base_code[Rule_No][conseq_code] == 7) defuzz_area = 0;
    sum_of_area = sum_of_area + defuzz_area;
    }
float Defuzzification()
// Defuzzification Unit.
   {    float CenterOfGravity;
    CenterOfGravity = sum_of_products/sum_of_area; /* Center of Gravity */
    return CenterOfGravity;
   }
void ControllerInputNormalization(float pvvolt,float setpoint_volt)
// Normalization of all fuzzy inputs.
   {    float system_input[3],
            level;
     static float prev_err;
     level = pvvolt;                          // true reading level (0-10 volt.)
     if (RESET_Status == 1) prev_err = 10;
     system_input[0] = (setpoint_volt - level); // error -10 - 10 volt.
```

```
    system_input[1] = 2.1*(system_input[0] - prev_err);  // change of error
    system_input[2] = /* pvvolt ;*/setpoint_volt; // 0-10 volt.
          if(system_input[0] > se/10)          // ..... .....
           system_input[0] = se/10;            // Error scaling &
          if(system_input[0] < -se/10)         // Error Normalization
            system_input[0] = -se/10;
           input[0] = (system_input[0])/(se/10); //    -1 to 1
                                               //   scaling CE
          if(system_input[1] >= sce/10)        // for CE max. = Sce
            system_input[1]= sce/10;
          if(system_input[1] <= -sce/10)
            system_input[1]= -sce/10;          // CE Normalization
           input[1]=(system_input[1])/(sce/10); //    -1 to 1
           input[2] = (system_input[2]-5)/5;   // AV Normalization
                                               //   -1 to 1
     prev_err = system_input[0];
   }
float ControllerOutput(float defuzzified_value)
// Controller output computing unit.
   {     float delta_ctrl_signal,
             delta_ctrl_output,
             current_ctrl_signal;
         static float prev_ctrl_signal;
         if(RESET_Status == 1) prev_ctrl_signal = 0;
         if(Bumpless == ON && MA == ON)
          {
```

```
        MA = OFF;
        prev_ctrl_signal = mv_volt[1];
        }
         // convert to delta control output
        delta_ctrl_output = defuzzified_value;   // control output is delta u(k)
         // convert the output value to control signal 0 - 10 volt.
        delta_ctrl_signal = delta_ctrl_output*(sdu/10);
         // scaleout delta-output signal
        if (delta_ctrl_signal > sdu/10) delta_ctrl_signal = sdu/10;
        if (delta_ctrl_signal < -sdu/10) delta_ctrl_signal = -sdu/10;
        current_ctrl_signal = prev_ctrl_signal + delta_ctrl_signal; // update new output signal
        if (current_ctrl_signal < 0) current_ctrl_signal = 0;        // Limit output signal
        if (current_ctrl_signal > 10) current_ctrl_signal = 10;      // into the range 4 - 20 mA.
        prev_ctrl_signal = current_ctrl_signal;                      // swap control signal
        return current_ctrl_signal;
   }
float FuzzyController(float pvvolt,float setpoint_volt,int curr_mode)
// Fuzzy Logic Inferencing Algorithm.
 {      float current_co,
            defuzzified_value;
        int  Fired_Rule_No,
            Fired_CrispInput_No;
        float Fuzzified_value;
        float Strength;      // Output strength for each infered rule
    static float previous_co;
    if (Bumpless == ON && MA == ON) previous_co = mv_volt[1];
```

```
switch (curr_mode)
{
case AUTO:
 sv_volt[1] = setpoint_volt + delta_sv; delta_sv = 0;
 if (sv_volt[1] > 10) sv_volt[1] = 10;
 if (sv_volt[1] < 0) sv_volt[1] = 0;
 setpoint_volt = sv_volt[1];
        ControllerInputNormalization(pvvolt,setpoint_volt);
        // Controller processing
        { sum_of_products = 0;
          sum_of_area = 0; }
 for (Fired_Rule_No = 0 ;Fired_Rule_No <= num_rules-1;++Fired_Rule_No)
 {
  Strength = 1.0;
  for (Fired_CrispInput_No = 0,Fired_CrispInput_No <= num_mf_if-1;++Fired_CrispInput_No)
  {
        Fuzzified_value = Fuzzification( Fired_Rule_No,Fired_CrispInput_No);
        Strength = Antecedent_Operator(Fuzzified_value,Strength);
        if (Strength == 0)  // reduce computation time
          break;
  }
        Rules_aggregation(Fired_Rule_No,Strength);
 }
   if (sum_of_area == 0) defuzzified_value = 0;
   else
   defuzzified_value = Defuzzification();
```

```
    current_co = ControllerOutput(defuzzified_value);
    if(current_co > 10) current_co = 10;
    if(current_co < 0) current_co = 0;
    previous_co = current_co;
  break;        // case A
 case MANUAL:
    current_co = previous_co + delta_co;
    if(FAST_SET) { current_co = mv_volt[1]; FAST_SET = 0;}
    if(current_co > 10) current_co = 10;
    if(current_co < 0) current_co = 0;
    previous_co = current_co;
    delta_co = 0;
  break;      // case M
 }              // switch A/M
 return current_co;
 }
```

ข.5 ไฟล์ส่วนหลักของโปรแกรม

```
/* ******************************************************
        Running section  (run.c)
  ****************************************************** */
 #include "ghder.h"
 #include "pcldrv.c"
 #include "fuzzy.c"
void RUN()
 {
```

```
        time_t time0,time1,
                time2; // Time count variables
        float co,pv_dummy;int i,time_passed;
    mouse_show_cursor();
// Controller processing section
    MA = OFF; RESET_Status = 1; SendOutDA(0.00);
    time_passed = 0;
  do // forever loop
  {     time0 = time('\0');
        time1 = time0 + DT;
        pv_dummy = 0;
    for (i=0;i<5;++i) {
     pv_dummy = pv_dummy + GetInputAD(); // Get input data from process using PCL-812PG A/D
card
        delay(200);   }
        pv_volt[1] = pv_dummy/5;
        if (pv_volt[1] > 10) pv_volt[1] = 10;
        if (pv_volt[1] < 0) pv_volt[1] = 0;
    // Limit PV value into the bounded range
        co = FuzzyController(pv_volt[1],sv_volt[1],mode); // fuzzy logic controller
        mv_volt[1] = co;
        SendOutDA(mv_volt[1]); // Send out control signal to PCL-812PG D/A card
        RESET_Status = 0;
  if (time_passed < 10000) time_passed += DT;
  time2 = time('\0');
   while (time2 < time1)
```

```
  {
    time2 = time('\0');                    //  delay excution time until cycle time pass trough DT sec.
  };
 } while (RESET_Status==0);      //  control loop
 }
// ———————————— MAIN PROGRAM ————————————————
 void main()
 {
        InitPCL812PG();     //  PCL812-PG card initialization
        RUN();              //  Run program
 }
```

ข.6 ไฟล์โปรแกรมตัวควบคุมแบบพีไอดีและอัลกอริธึมของพีไอดีแบบกำหนดเกนต่างๆ กัน

```
/*  PID Controller
   6 Algorithm
   position & velocity diff on err & measurment
   anti-reset windup option & bumpless transfer options are included
*/
float PID(float fKc, float fTi, float fTd,float current_pvvolt,float setpoint_volt,int algor,int curr_mode)
{
  static float BIAS;     //  bias value (volt)
  float  error,current_error,
         integral,     //  PID parameters
         diff,
         delta_int,
         current_co; // current controller output
```

```
 static float previous_error,
                    previous2_error,
                    previous_pv,
                    previous2_pv,
                    previous_integral,
                    previous_co;
switch (curr_mode)
{
 case AUTO:
  sv_volt[1] = setpoint_volt + delta_sv, delta_sv = 0;
  if (sv_volt[1] > 10) sv_volt[1] = 10;
  if (sv_volt[1] < 0) sv_volt[1] = 0;
  current_error = ( sv_volt[1] - current_pvvolt );
  if (RESET_Status == 1)
   {
   BIAS = 0;
   previous_pv = 0;
   previous2_pv = 0;
   previous_error = 0;
   previous2_error = 0;
   previous_integral = 0;
   previous_co = 0;
   }
  if ( (Bumpless == ON && MA == ON) || CHGALG == ON)
   {
    MA = OFF; CHGALG = OFF;
```

```
previous_pv = pv_volt[1];
BIAS = mv_volt[1];
previous2_pv = pv_volt[1];
previous_integral = 0;
previous_co = mv_volt[1];
}
switch (algor) {
case 1:
        // position diff on error with Anti-Reset windup
        error   = current_error;
        delta_int = current_error*DT;
if(ANTI)                {
if(previous_co >= 10 || previous_co <= 0)
         delta_int = 0;    }
        integral = previous_integral + delta_int;
        diff    = (current_error - previous_error)/DT;
current_co = BIAS + fKc*(error +integral/fTi +fTd*diff);
        if(current_co > 10)  current_co = 10;
        if(current_co < 0)   current_co = 0;
 break;
case 2:
        // velocity diff on error
        error   = current_error - previous_error;
        integral = current_error*DT;
        diff    = (current_error - 2*previous_error + previous2_error)/DT;
current_co = fKc*(error +integral/fTi +fTd*diff) + previous_co;
```

```
        if(current_co > 10)  current_co = 10;
        if(current_co < 0)   current_co = 0;
 break;
case 3:
        // position diff on measurement with Anti-Reset windup
        error   = current_error;
        delta_int = current_error*DT;
        if(ANTI)          {
         if(previous_co >= 10 || previous_co <= 0)
          delta_int = 0      }
        integral = previous_integral + delta_int;
        diff     = (current_pvvolt - previous_pv)/DT;
current_co = BIAS + fKc*(error +integral/fTi +fTd*diff);
        if(current_co > 10)  current_co = 10;
        if(current_co < 0)   current_co = 0;
 break;
case 4:
        //  velocity diff on measurment
        error   =  previous_pv - current_pvvolt;
        integral =  current_error*DT;
        diff    = (2*previous_pv - current_pvvolt -  previous2_pv)/DT;
current_co = fKc*(error +integral/fTi +fTd*diff) + previous_co;
        if(current_co > 10)  current_co = 10;
        if(current_co < 0)   current_co = 0;
 break;
          }
```

```
previous_pv = current_pvvolt;
previous2_pv = previous_pv;
previous_error = current_error;
previous2_error = previous_error;
previous_integral = integral;
if (current_co > 10) current_co = 10;
if (current_co < 0) current_co = 0;
previous_co = current_co;
break;   // case A
case MANUAL:
 if (RESET_Status == 1) previous_co = mv_volt[1];
 current_co = previous_co + delta_co;
 if (FAST_SET) { current_co = mv_volt[1]; FAST_SET = 0; }
 if (current_co > 10) current_co = 10;
 if (current_co < 0) current_co = 0;
 previous_co = current_co;
 delta_co = 0;
 break;   // case M
 }          // switch A/M
 return current_co;
}
void Gain_Schd_Mode(KC_PRG)
 { if ( KC_PRG )
  { if (pv_volt[1] <= 10 && pv_volt[1] > 8) Kc = Kc2 + (pv_volt[1] - 8)*(Kc1-Kc2)/(10-8);
    if (pv_volt[1] <= 8  && pv_volt[1] >= 5) Kc = Kc3 + (pv_volt[1] - 6)*(Kc2-Kc3)/(8-6);
    if (pv_volt[1] <= 5  && pv_volt[1] > 3) Kc = Kc4 + (pv_volt[1] - 3)*(Kc3-Kc4)/(6-3);
```

```
    if(pv_volt[1]<=3  && pv_volt[1]> 2) Kc=Kc5+(pv_volt[1]-2)*(Kc4-Kc5)/(3-2);
    if(pv_volt[1]<=2  && pv_volt[1]>=0) Kc=Kc5;
   };
        co=PID(Kc,Ti,tD,pv_volt[1],sv_volt[1],ALGOR,mode); /* PID Controller */
        mv_volt[1]=co;
}
```

ประวัติผู้เขียน

นายนฤพนธ์ มัญมณี เกิดวันที่ 25 มกราคม พ.ศ. 2513 ที่จังหวัดสุพรรณบุรี สำเร็จการศึกษาระดับปริญาตรีวิทยาศาสตร์บัณฑิต สาขาเคมีอุตสาหกรรม คณะวิทยาศาสตร์ประยุกต์ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ในปีการศึกษา 2534 และศึกษาต่อในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมเคมี เมื่อ พ. ศ. 2535