APPENDIX A

CALIBRATION OF TEMPERATURE CONTROL CHAMBER

This Table is shown the resistance of thermistor and the temperature of solution in sample cell.

Resistance (Ω)	Temperature (°C)
110	48.3
115	47.3
120	45.9
125	44.7
130	43.5
135	42.4
140	41.3
145	40.3
150	39.2
155	38.2
160	37.3
165	36.2
170	35.2
180	33.4
185	32.5
190	31.7
195	30.8
200	30.0

Using the computer program to find the best fitted of this result, it was found to be the Equation

$$Y = a + b(\ln X)^2$$

And a = 116.50 and b = -3.08

Where Y is temperature and X is resistance. The plot of resistance versus temperature is shown in the Figure

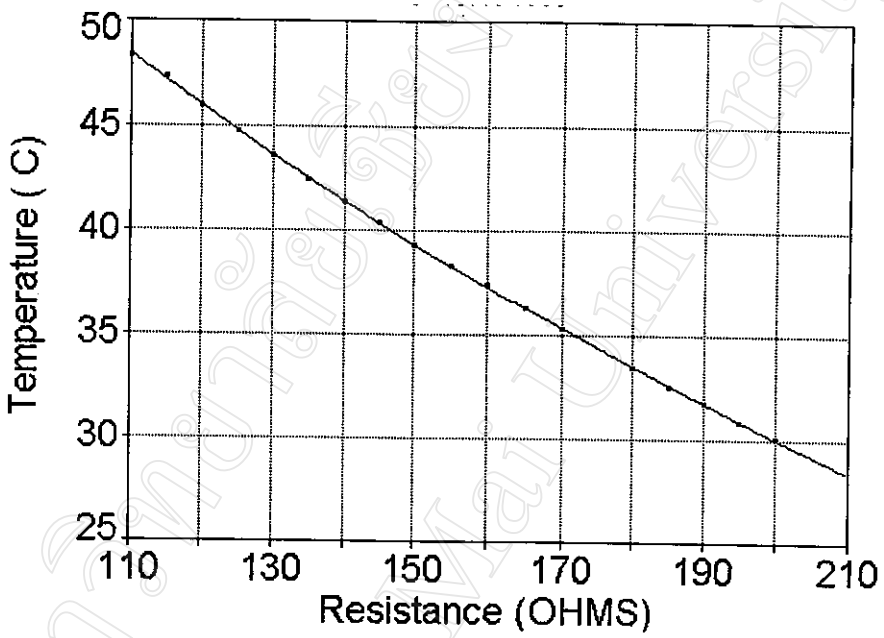

Figure shows the plot of temperature versus resistance

APPENDIX B

COMPUTER PROGRAM

The program P_MCS.CPP is uses for control the multichannel scaler and record the correlation function and photon count of the scattering signal.

```
/* ================================
   Program control and calculate
   correlation data from MCS
   Turbo-MCS EG&G ORTEC
   Chaiyapong Ruangsuwan
   December 27, 97
   ================================ */
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#pragma inline

#define  MAINFORE   YELLOW
#define  MAINBACK   LIGHTBLUE
#define  MSSGFORE   LIGHTRED
#define  UP         72
#define  DOWN       80
#define  ENTER      13
#define  ESC        27
#define  TIME       250
#define  CHOICE     5
#define  PRINTNAME  "prn"


#define K32    32768
#define K16    16384
```

```
#define ss    "            "

/* Define constants used for communication */

#define OUTFLG   0xD0000000L    /* Segment:Offset  D000:0000 */
#define TEST     0xD0000002L    /* Segment:Offset  D000:0002 */
#define OUTLENLO 0xD000001CL    /* Segment:Offset  D000:001C */
#define OUTLENHI 0xD000001EL    /* Segment:Offset  D000:001E */
#define OUTBUF   0xD0000020L    /* Segment:Offset  D000:0020 */
#define INFLG    0xD00003E0L    /* Segment:Offset  D000:03E0 */
#define INLENLO  0xD00003FCL    /* Segment:Offset  D000:03FC */
#define INLENHI  0xD00003FEL    /* Segment:Offset  D000:03FE */
#define INBUF    0xD0000400L    /* Segment:Offset  D000:0400 */

#define TRUE     '\377'
#define FALSE    '\0'

void Initialize_MCS (void);
void near MCS_C (void);
void near MCS_STAT (void);
class Output
 {
   public:
    void textbox( int left, int top, int right,
            int bottom, int style);
 } Output;
extern int iMbxIO(char *, char *, char *);
extern  char *cGetResponse(void);

char     far  *cpMcbOutFlag,     /* Set TRUE when command is ready */
         far  *cpMcbTest,        /* Written and read to see if mailbox exists */
         far  *cpMcbOutLengthLow,   /* Low byte of length of command string */
         far  *cpMcbOutLengthHigh,  /* High byte of length of command string */
         far  *cpMcbOutBuffer,      /* Buffer in 914 for commands */
         far  *cpMcbInFlag,      /* Set TRUE by 914 when response is ready */
         far  *cpMcbInLengthLow,    /* Low byte of length of response string */
         far  *cpMcbInLengthHigh,   /* High byte of length of response string */
         far  *cpMcbInBuffer;       /* Buffer in 914 for responses */



unsigned char mcd [K16];
double       density [100];
double       c_data [100];
unsigned int  lb [101];
unsigned int  hb [101];
int          rrun;
```

```
long int     li, lj;
float        a, b, x, y, st;
unsigned char pp, sp;
unsigned int  l1, l2, l3, l4, dlay;
struct       time l;
unsigned int  time_set;
char         *dwell, r, *z, t;
char         rsp, fname [14], *Ss;
char        ff, temp [10];
char         ch, *s, unit;
static char   * corr [10] =
                { "stop", "clear_all", "disable_summation",
                  "disable_dwell_external", "disable_trigger",
                  "enable_discriminator", "set_discriminator_trailing",
                  "set_pass 0", "set_pass_length 16384",
                  "set_pass_preset 1" };

FILE         *output_file, *step_file;
float            f_stop, channel, Ohm_read;
main ()
{
  char  szResponse[512],         /* response record, if one exists */
        szPercentResponse[512];          /* percent response record */

  unsigned char  byMcbNumber;          /* MCB number to talk to */
  char far *ReadMCS;
  int     iCommError;             /* communication timeout error flag */
  int  loop;
  register int  i, runtime, j;
  int angle;

/* Initialize pointers */

  cpMcbOutFlag        = (char far *)OUTFLG;
  cpMcbTest    = (char far *)TEST;
  cpMcbOutLengthLow = (char far *)OUTLENLO;
  cpMcbOutLengthHigh = (char far *)OUTLENHI;
  cpMcbOutBuffer   = (char far *)OUTBUF;
  cpMcbInFlag          = (char far *)INFLG;
  cpMcbInLengthLow  = (char far *)INLENLO;
  cpMcbInLengthHigh  = (char far *)INLENHI;
  cpMcbInBuffer    = (char far *)INBUF;

  clrscr ();
  Output.textbox ( 15, 2, 60, 7, 2);
  textcolor (YELLOW);
  gotoxy ( 19, 3); cprintf ( "PowerMCS as correlator Version 1.4000");
```

```
  gotoxy ( 31, 4); cprintf ( "By Mr. Frank");
  gotoxy ( 16, 5); cprintf ( "Laser Research Laboratory, Dept. of Physics");
  gotoxy ( 27, 6); cprintf ( "Chiangmai University");
  textcolor (WHITE);
/* Get MCB to communicate with
 this program set MCB number is 1*/
  byMcbNumber = 1;
  outp (0x292, (byMcbNumber + 7)); /* Data memory is MCB -1, mailbox is MCB +
7 */
  *cpMcbTest = '\252';      /* Write test pattern 010101010  */
  if (*cpMcbTest != '\252')  /* If we can't read it, mailbox isn't there */
    {
    cout << "\n\nThis program can not run without Turbo-MCS\n";
    cout << "ERROR: Dual Port memory does not exist\n";
    cout << "Please check the connection between PC and MCS\n";
    cout << "and run TurboMCS again\n";
    exit(1);
    }
  Initialize_MCS ();
  gotoxy ( 1, 9);
  cout << "\nPlease enter Sample time (microsec).....";
  scanf ( "%d", &dlay);
  itoa ( dlay, temp, 10);
        strncat (temp, "e-6", 25);
  printf ( "Number of 16K loop : ");
  scanf ( "%d", &rrun);
  printf ( "Scattering angle : ");
  scanf ( "%d", &angle);
  cout << "f-stop : ";
  cin >> f_stop;
  cout << "Temperature (Ohm) : ";
  cin >> Ohm_read;
  printf ( "Save data as : ");
  scanf ( "%s", &fname);
  if ((output_file = fopen ( fname, "wt")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    return 1;
  }
  clrscr ();
  cout << "Receiving data from MCS ... Please wait";
  Ss = "set_dwell ";
  strncat ( Ss, temp, 20);
  strcpy ( temp, Ss);
  iMbxIO( temp, szResponse, szPercentResponse);
  iMbxIO( "clear", szResponse, szPercentResponse);
  iMbxIO( "set_pass_preset 1", szResponse, szPercentResponse);
```

```
  iMbxIO( "start", szResponse, szPercentResponse);
// Initial value of variables
  for ( i = 0; i < 100; i++)
  {
     c_data [i] = 0;
     density [i] = 0;
  }
  delay ( dlay * 25);
 j = 0;
  channel = (float) dlay * 1.0e-6;
  runtime = rrun;
  do
  {
    ReadMCS = (char far *)OUTFLG;
    i = 0;
    outp ( 0x292, 0);
     do
     {
          mcd [i] = *ReadMCS;
          ReadMCS += 4;
          i ++;
     } while (i!=K16);
    outp ( 0x292, 8);
    iMbxIO( "clear", szResponse, szPercentResponse);
    iMbxIO( "set_pass_preset 1", szResponse, szPercentResponse);
    iMbxIO( "start", szResponse, szPercentResponse);
    delay (dlay * 25);
    MCS_STAT ();
    for ( i = 0; i < 100; i++)
         density [i] += ( lb [i] + 65536 * hb [i]);
    MCS_C ();
    for ( i = 0; i < 100; i++)
         c_data [99 - i] += ( lb [i] + 65536 * hb [i]);
    clrscr ();
         cout << "Please wait, in processing\n";
         cout << "Run number "<< j + 1 << " of " << runtime << "\n";
  if ( j == 999)
     step_file = fopen ( "take1000.txt", "wt");
   else if ( j == 1999)
     step_file = fopen ( "take2000.txt", "wt");
   else if ( j == 2499)
     step_file = fopen ( "take2500.txt", "wt");
   else if ( j == 2999)
     step_file = fopen ( "take3000.txt", "wt");
   else if ( j == 3499)
     step_file = fopen ( "take3500.txt", "wt");
   else if ( j == 3999)
```

```
    step_file = fopen ( "take4000.txt", "wt");
   if ( step_file != NULL)
   {
    for ( i = 0; i < 100; i++)
        fprintf ( step_file, "%d\t%8.6f\t%lg\t%lg\n",
                i, i * channel, c_data [i], density [i]);
        fclose ( step_file);
   }
  j++;
  } while ( j!=runtime);

  fprintf ( output_file, "FILE : %s\n", fname);
  fprintf ( output_file, "%s\n", temp);
  fprintf ( output_file, "number of run := %d\n", runtime);
  fprintf ( output_file, "Scattering angle : %d\n", angle);
  fprintf ( output_file, "Temperatur setting : %6.2f\n", Ohm_read);
  fprintf ( output_file, "f-stop := %6.2f\n", f_stop);
  fprintf ( output_file, "Channel\t\Time\tCorrelation\tDensity\n");
  for ( i = 0; i < 100; i++)
    fprintf ( output_file, "%d\t%8.6f\t%lg\t%lg\n",
                i, i * channel, c_data [i], density [i]);
  fclose ( output_file);
  cout << "Data is already in file : " << fname << "\n";
  sound ( 500);
  delay ( 2000);
  nosound ();
  while (kbhit()) getch (); /* Clear KeyBoard Buffer */
  return 0;
}

// function MCS_C
// create photon correlation funtion
// by using full correlation method
// channel : 100
// input : mcd [0...16383]
// output : lh [0...99] and hb [0...99]

void near MCS_C ()
{
  asm {
        xor  ax,ax
        mov  cx,100
        xor  si,si }
  loop_0:
  asm {
        mov  lb[si],ax
        mov  hb[si],ax
```

```
          inc  si
          inc  si
          loop loop_0  }

    asm   xor  si,si
    loop_1:
          asm {
           push si
           mov  bl,mcd[si]
           inc  si
           mov  cx,100  }
    loop_2:
          asm {
           mov  al,mcd[si]
           mul  bl
           push ax
           inc  si
           loop loop_2  }       /*   ; save 100 A1*Bl on stack*/

          asm {
           xor  si,si
           mov  cx,100 }
    loop_3:
          asm {
           pop  ax
           add  lb[si],ax
           adc  hb[si],0
           inc  si
           inc  si
           loop loop_3  }    /*       ; add 100-point Correlation Functions*/

      asm  {
           pop  si
           inc  si
           cmp  si,03f9ah    /*   ; Check if 16282 points DONE*/
           jb   loop_1 }

}

// procedure MCS_STAT
// find the distribution of photon counting

void near MCS_STAT ()
{
   asm {
          xor  ax,ax
          mov  cx,100
```

```
        xor  si,si }
  loop_0:
   asm {
        mov  lb[si],ax
        mov  hb[si],ax
        inc  si
        inc  si
        loop loop_0 }

   asm  xor  si,si
  loop_1:
   asm {
        mov  bl,mcd[si]
        xor  ax,ax
        mov  al,bl
        mov  di,ax
        add  di,di
        add  lb[di],1     /*  ; Distribution Function*/
        adc  hb[di],0
        inc  si
        cmp  si,03f9ah     /*   ; Check if 16282 points DONE*/
        jb   loop_1  }
}

void Initialize_MCS ()
{
  int i;
  char szResponse [512], szPercentResponse [512];

  for ( i = 0; i < 10; i++ )
    iMbxIO( corr [i], szResponse, szPercentResponse);
}

/*****************************************************************
***/
/*       iMbxIO -- sends a command to the mailbox and gets the response     */
/*****************************************************************
***/

int iMbxIO(char *szCommand, char *szResponse, char *szPercentResponse)
{

  int     iCounter,          /* general loop counter */
          iErrorFlag;          /* Timeout error flag */

  long   lStartTime,       /* used to calculate timeout */
          lPresentTime;
```

```
  iErrorFlag = -1;            /* init as error until we are successful */
  *cpMcbOutLengthLow = '\0';       /* send a zero length message to sync mailbox */
  *cpMcbOutLengthHigh = '\0';
  *cpMcbOutFlag = TRUE;

  *cpMcbInFlag    = FALSE;
  time(&lStartTime);    /* wait for MCB to clear output flag */
  time(&lPresentTime);
  while ( (*cpMcbOutFlag == TRUE) && (lPresentTime - lStartTime < 5))
    {
    *cpMcbInFlag = FALSE;
    time(&lPresentTime);
    }
  if (lPresentTime - lStartTime >= 5)
    {
    printf("MCB not responding\n");
    return(iErrorFlag);
    }

/* Put command in output buffer */

  for (iCounter = 0; iCounter < (int)strlen(szCommand); iCounter++)
    *(cpMcbOutBuffer + (2 * iCounter)) = *(szCommand + iCounter);

/* Write out length of command */

  *cpMcbOutLengthLow = (char)(strlen(szCommand) % 256);
  *cpMcbOutLengthHigh = (char)(strlen(szCommand) / 256);

/* Set the out flag to say the command is ready */

  *cpMcbOutFlag = TRUE;

/* Get first response record */

  strcpy(szPercentResponse,cGetResponse());
  if (strcmpi(szPercentResponse,"err") == 0)
    return(iErrorFlag);

/* See if it was a percent response */

  if (strncmp(szPercentResponse,"%",1) == 0)
    {
    strnset(szResponse,'\0',1);
    iErrorFlag = 0;                  /* good return */
    return(iErrorFlag);
    }
```

```
/* It wasn't so copy it to response and go get the percent response */
  strcpy(szResponse,szPercentResponse);
  strcpy(szPercentResponse,cGetResponse());
  if (strcmpi(szPercentResponse,"err") == 0)
    return(iErrorFlag);

  iErrorFlag = 0;                    /* good return */
  return(iErrorFlag);
}



/*************************************************************************
***/
/*                cGetResponse -- gets response from the MCB                */
/*************************************************************************
***/

char * cGetResponse()


{

  char  iResponseBuffer[512];

  int     iCounter,
          iNumOfChars;

  long  lStartTime,
          lPresentTime;

/* Wait for MCB response */

  time(&lStartTime);
  time(&lPresentTime);
  while ((*cpMcbInFlag == FALSE) && (lPresentTime - lStartTime < 5))
    time(&lPresentTime);
  if (lPresentTime - lStartTime >= 5)
    {
    printf("MCB not responding\n");
    strcpy(iResponseBuffer,"err");
    return(iResponseBuffer);
    }

/* Get number of characters in response and read */
```

```
  iNumOfChars = (int)*cpMcbInLengthLow + 256 * (int)*cpMcbInLengthHigh;
  memset(iResponseBuffer,'\0',512);
  for (iCounter = 0; iCounter < iNumOfChars; iCounter++)
    iResponseBuffer[iCounter] = *(cpMcbInBuffer + (2 * iCounter));

/* reset input buffer flag and return response address */

  *cpMcbInFlag = FALSE;
  return(iResponseBuffer);
}

/* The style argument determinees the type of dox draw
        0 = no box
        1 = single-score
        2 = double-score                */

void Output :: textbox( int left, int top, int right,
                        int bottom, int style)
{
 register  r, c;
 static bord [][6] = {          /* border character */
        { 196, 179, 218, 191, 217, 192 },
        { 205, 186, 201, 187, 188, 200 }};
 if ( style == 0) return;
 --style;
 /* Draw horizontals */
 for( c = left + 1; c < right; c++){
  gotoxy( c, top);
  cprintf("%c",bord[style][0]);
  gotoxy( c, bottom);
  cprintf("%c",bord[style][0]);
 }
 /*  Draw vertical  */
 for(r = top + 1; r < bottom; r++){
  gotoxy( left, r);
  cprintf("%c",bord[style][1]);
  gotoxy( right, r);
  cprintf("%c",bord[style][1]);
 }
 /* set corners */
 gotoxy(left,top);   cprintf("%c",bord[style][2]);
 gotoxy(right,top);  cprintf("%c",bord[style][3]);
 gotoxy(right,bottom);cprintf("%c",bord[style][4]);
 gotoxy(left,bottom); cprintf("%c",bord[style][5]);
}/*---------------------------*/
```

APPENDIX C

VISCOSITY OF CYCLOHEXANE

The viscosity η of cyclohexane was measured by the Cannon-Fenske Routine Viscometer of the Cannon Instrument CO. State College, PA USA. The results shown in Tables following.

Temperature (°C)	Viscosity (cp)
30.0	1.117
32.0	1.026
34.0	0.943
36.0	0.866
38.0	0.797
40.0	0.736
42.0	0.683
44.0	0.637
46.0	0.599
48.0	0.568
50.0	0.545

The viscosity was fitted to the polynomial $Y = a + bX + cX^2 + dX^3$ with computer software. The constants are

$a = 3.3775$, $b = -0.1030$, $c = 0.0009$, $d = 2.7195 \times 10^{-7}$

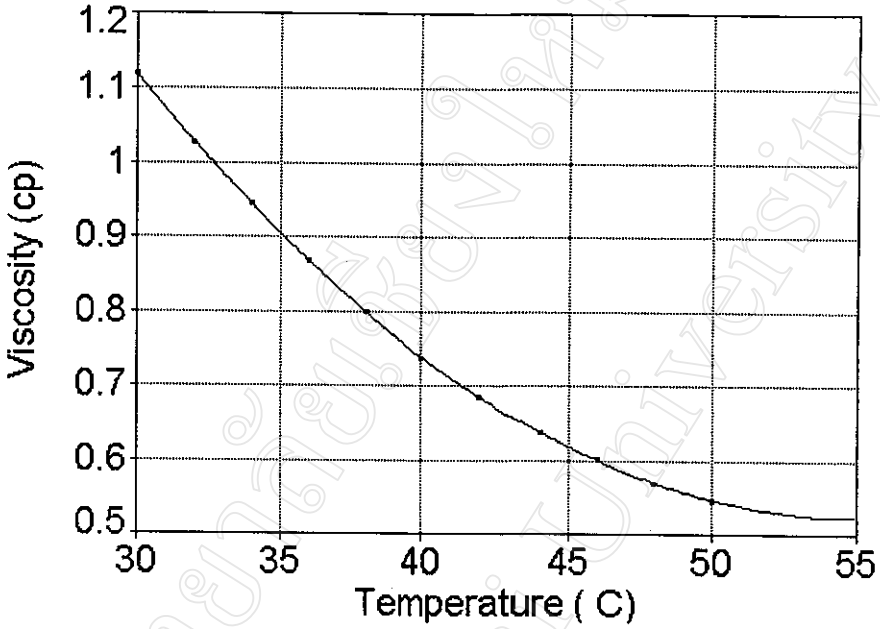

Figure shows plot of viscosity of cyclohexane versus temperature.

Curriculum Vitae

Name	Mr.Chaiyapong Ruangsuwan
Date of Birth	2 March 1970
Education	B.Sc. (physics), Khon Kaen university, Khon Kaen.
Experience	1992-1993: Failure analysis engineer, Seagate Technology (Thailand).
	Since 1993: Lecturer, department of physics Khon Kaen university, Khon Kaen.