```
645     int v_offset_ch0=0;
646     int v_offset_ch1=0;
647
648     for(i=0;(i<=count) && (m_bEnable == FALSE);i++)
649   {
650         if(i==0)
651       {
652             ao(0,0);
653             ao(1,0);
654             Sleep(1);
655
656             ad_set_gain(volt_card_gain);
657             for(j=0;j<Average;j++)
658             {
659               ad_set_channel(V_0_CH);
660                 ad_acquire(&ad);
661                 v_offset_ch0+=ad;
662
663                 ad_set_channel(V_1_CH);
664                 ad_acquire(&ad);
665                 v_offset_ch1+=ad;
666             }
667
668             v_offset_ch0 /= Average;
669         v_offset_ch1 /= Average;
670
671             v_offset_ch0 -= 2048;
672         v_offset_ch1 -= 2048;
673       }
674       else
675       {
676             ao(0,(i-1)*step_len);
677             ao(1,(i-1)*step_len);
678       }
679         Sleep(1);
680
681         a_sum_ch0=0;
682         v_sum_ch0=0;
683         a_sum_ch1=0;
684         v_sum_ch1=0;
685
686         ad_set_gain(amp_card_gain);
687         for(j=0;j<Average;j++)
688         {
689             ad_set_channel(A_0_CH);
690             ad_acquire(&ad);
691             a_sum_ch0+=ad;
692
693             ad_set_channel(A_1_CH);
694             ad_acquire(&ad);
695             a_sum_ch1+=ad;
696         }
697
698         ad_set_gain(volt_card_gain);
699         for(j=0;j<Average;j++)
```

```
700             {
701                 ad_set_channel(V_0_CH);
702                 ad_acquire(&ad);
703                 v_sum_ch0+=ad;
704
705                 ad_set_channel(V_1_CH);
706                 ad_acquire(&ad);
707                 v_sum_ch1+=ad;
708         }
709
710             a_sum_ch0/=Average;
711             v_sum_ch0/=Average;
712
713             a_sum_ch1/=Average;
714         v_sum_ch1/=Average;
715
716             v_sum_ch0 -= v_offset_ch0;
717             v_sum_ch1 -= v_offset_ch1;
718
719     //      a_sum_ch0 = 4096-a_sum_ch0;  //inverse the A/D counts
720     //      v_sum_ch0 = 4096-v_sum_ch0;
721
722     //      a_sum_ch1 = 4096-a_sum_ch1;
723     //      v_sum_ch1 = 4096-v_sum_ch1;
724
725             x_ch0[i]=(int)(a_sum_ch0*a_factor)+X_ORG;
726             y_ch0[i]=(int)((4095-v_sum_ch0)*v_factor)+Y_ORG;
727
728             x_ch1[i]=(int)(a_sum_ch1*a_factor)+X_ORG;
729             y_ch1[i]=(int)((4095-v_sum_ch1)*v_factor)+Y_ORG;
730
731             v_count_ch0[i]=v_sum_ch0;
732         a_count_ch0[i]=a_sum_ch0;
733
734             v_count_ch1[i]=v_sum_ch1;
735         a_count_ch1[i]=a_sum_ch1;
736
737
738             MSG message;
739
740             if(::PeekMessage(&message,NULL,0,0,PM_REMOVE))
741             {
742                 ::TranslateMessage(&message);
743                 ::DispatchMessage(&message);
744             }
745
746             Invalidate();
747             Beep(1000,500);
748             Sleep(Delay);
749       }
750
751         time(&t_end);
752
753         Invalidate();
754       ao(0,0);
```

```
755         ao(1,0);
756
757         FILE* fp_data_ch0;
758       fp_data_ch0=fopen("DATA_CH0.DAQ","w+");
759         fprintf(fp_data_ch0,"Data for CH0 testing:\n\n");
760         fprintf(fp_data_ch0,"BATTERY ID = %s\n\n",(const char*)Ch0_id);
761         fprintf(fp_data_ch0,"Begin time: %s", ctime( &t_begin ) );
762         fprintf(fp_data_ch0,"End   time: %s\n", ctime( &t_end ) );
763         fprintf(fp_data_ch0,"sample:      V           A \n");
764
765       Voc_ch0=0.0;
766       a_count_min=30000;
767       for(j=1;j<i;j++)
768       {
769         volt_ch0[j]= (v_count_ch0[j]-2048)*volt_factor;
770         amp_ch0[j]= (a_count_ch0[j]-2048)*amp_factor/AMP_RES_CH0;
771
772         if( amp_ch0[j]<0 )
773         {
774                   pwr_ch0[j]=volt_ch0[j]*(-amp_ch0[j]);
775         }
776         else
777         {
778                   pwr_ch0[j]=0;
779         }
780
781         if( (abs(a_count_ch0[j]-2048)<a_count_min) &&
782                     (a_count_ch0[j]>1900) && (a_count_ch0[j]<2200) )
783         {
784                   Voc_ch0=volt_ch0[j];
785                   a_count_min=abs(a_count_ch0[j]-2048);
786         }
787         fprintf(fp_data_ch0,"%04d:     %8.4lf    %9.6lf\n",j,volt_ch0[j],amp_ch0[j]);
788       }
789          fclose(fp_data_ch0);
790
791       Isc_ch0 = -amp_ch0[1];
792
793       for(j=1;j<i;j++)
794       {
795         if(pwr_ch0[j]>Pmax_ch0)
796         {
797                   Pmax_ch0 = pwr_ch0[j];
798                   Vmp_ch0 = volt_ch0[j];
799                   Imp_ch0 = -amp_ch0[j];
800         }
801       }
802
803       if(Voc_ch0*Isc_ch0 != 0)
804       {
805         FF_ch0 = Pmax_ch0 / (Voc_ch0 * Isc_ch0);
806       }
807       else
808       {
809         FF_ch0 = 0;
```

```
}

Eff_ch0 = Pmax_ch0 * 100 / ( Size * 0.1 * Sun);
Jsc_ch0 = Isc_ch0 / Size;


   FILE* fp_report_ch0;
fp_report_ch0=fopen("REPORT_CH0.DAR","w+");
fprintf(fp_report_ch0,"Report for CH0 testing:\n\n");
   fprintf(fp_report_ch0,"BATTERY ID = %s\n\n",(const char*)Ch0_id);
   fprintf(fp_report_ch0,"Begin time: %s", ctime( &t_begin ) );
   fprintf(fp_report_ch0,"End   time: %s\n", ctime( &t_end ) );

   fprintf(fp_report_ch0,"Average = %d\n",Average);
   fprintf(fp_report_ch0,"Vmax    = %-6.3f V\n",Vmax);
   fprintf(fp_report_ch0,"Sample  = %d\n",Sample);
   fprintf(fp_report_ch0,"Delay   = %d ms\n",Delay);
   fprintf(fp_report_ch0,"Size    = %-7.3lf cm**2\n",Size);
fprintf(fp_report_ch0,"Sun     = %-6.3lf  sun\n",Sun);
   fprintf(fp_report_ch0,"T_dry   = %-5.1lf   degree\n",T_dry);
   fprintf(fp_report_ch0,"T_wet   = %-5.1lf   degree\n",T_wet);
   fprintf(fp_report_ch0,"Humidity= %-5.1lf   %%\n\n\n",Hum);

   fprintf(fp_report_ch0,"FF =  %6.3lf\n",FF_ch0);
fprintf(fp_report_ch0,"Eff= %7.3lf %%\n",Eff_ch0);
fprintf(fp_report_ch0,"Jsc=   %6.5lf A/(cm**2) \n",Jsc_ch0);
fprintf(fp_report_ch0,"Isc=  %9.6lf A\n",Isc_ch0);
fprintf(fp_report_ch0,"Voc= %8.4lf V\n",Voc_ch0);
fprintf(fp_report_ch0,"Imp=  %9.6lf A\n",Imp_ch0);
fprintf(fp_report_ch0,"Vmp= %8.4lf V\n",Vmp_ch0);
fprintf(fp_report_ch0,"Pmax= %9.6lf W\n",Pmax_ch0);

fclose(fp_report_ch0);

   FILE* fp_data_ch1;
fp_data_ch1=fopen("DATA_CH1.DAQ","w+");
   fprintf(fp_data_ch1,"Data for CH1 testing:\n\n");
   fprintf(fp_data_ch1,"BATTERY ID = %s\n\n",(const char*)Ch1_id);
   fprintf(fp_data_ch1,"Begin time: %s", ctime( &t_begin ) );
   fprintf(fp_data_ch1,"End   time: %s\n", ctime( &t_end ) );
   fprintf(fp_data_ch1,"sample:      V           A \n");

Voc_ch1=0.0;
a_count_min=30000;
for(j=1;j<i;j++)
{
  volt_ch1[j]= (v_count_ch1[j]-2048)*volt_factor;
  amp_ch1[j]= (a_count_ch1[j]-2048)*amp_factor/AMP_RES_CH1;

  if( amp_ch1[j]<0 )
  {
          pwr_ch1[j]=volt_ch1[j]*(-amp_ch1[j]);
  }
  else
  {
```

```
865                 pwr_ch1[j]=0;
866         }
867
868         if( (abs(a_count_ch1[j]-2048)<a_count_min) &&
869                   (a_count_ch1[j]>1900) && (a_count_ch1[j]<2200) )
870         {
871                 Voc_ch1=volt_ch1[j];
872                 a_count_min=abs(a_count_ch1[j]-2048);
873         }
874         fprintf(fp_data_ch1,"%04d:     %8.4lf    %9.6lf\n",j,volt_ch1[j],amp_ch1[j]);
875       }
876          fclose(fp_data_ch1);
877
878       Isc_ch1 = -amp_ch1[1];
879
880       for(j=1;j<i;j++)
881       {
882         if(pwr_ch1[j]>Pmax_ch1)
883         {
884                 Pmax_ch1 = pwr_ch1[j];
885                 Vmp_ch1 = volt_ch1[j];
886                 Imp_ch1 = -amp_ch1[j];
887         }
888       }
889
890       if(Voc_ch1*Isc_ch1 != 0)
891       {
892         FF_ch1 = Pmax_ch1 / (Voc_ch1 * Isc_ch1);
893       }
894       else
895       {
896         FF_ch1 = 0;
897       }
898
899       Eff_ch1 = Pmax_ch1 * 100 / ( Size * 0.1 * Sun );
900       Jsc_ch1 = Isc_ch1 / Size;
901
902          FILE* fp_report_ch1;
903       fp_report_ch1=fopen("REPORT_CH1.DAR","w+");
904       fprintf(fp_report_ch1,"Report for CH1 testing:\n\n");
905          fprintf(fp_report_ch1,"BATTERY ID = %s\n\n",(const char*)Ch1_id);
906          fprintf(fp_report_ch1,"Begin time: %s", ctime( &t_begin ) );
907          fprintf(fp_report_ch1,"End   time: %s\n", ctime( &t_end ) );
908
909          fprintf(fp_report_ch1,"Average = %d\n",Average);
910          fprintf(fp_report_ch1,"Vmax    = %-6.3f V\n",Vmax);
911          fprintf(fp_report_ch1,"Sample  = %d\n",Sample);
912          fprintf(fp_report_ch1,"Delay   = %d ms\n",Delay);
913          fprintf(fp_report_ch1,"Size    = %-7.3lf cm**2\n",Size);
914       fprintf(fp_report_ch1,"Sun     = %-6.3lf  sun\n",Sun);
915          fprintf(fp_report_ch1,"T_dry   = %-5.1lf   degree\n",T_dry);
916          fprintf(fp_report_ch1,"T_wet   = %-5.1lf   degree\n",T_wet);
917          fprintf(fp_report_ch1,"Humidity= %-5.1lf   %%\n\n\n",Hum);
918
919       fprintf(fp_report_ch1,"FF =  %6.3lf\n",FF_ch1);
```

```
920     fprintf(fp_report_ch1,"Eff= %7.3lf %%\n",Eff_ch1);
921     fprintf(fp_report_ch1,"Jsc=   %6.4lf A/(cm**2) \n",Jsc_ch1);
922     fprintf(fp_report_ch1,"Isc=  %9.6lf A\n",Isc_ch1);
923     fprintf(fp_report_ch1,"Voc=  %8.4lf V\n",Voc_ch1);
924     fprintf(fp_report_ch1,"Imp=  %9.6lf A\n",Imp_ch1);
925     fprintf(fp_report_ch1,"Vmp= %8.4lf V\n",Vmp_ch1);
926     fprintf(fp_report_ch1,"Pmax= %9.6lf W\n",Pmax_ch1);
927
928     fclose(fp_report_ch1);
929
930       m_bEnable=TRUE;
931
932       msg = MessageBox( "测试已经结束\n\n查看数据文件吗?","数据文件",MB_YESNO );
933
934       if(msg == IDYES )
935       {
936            _spawnlp(_P_NOWAIT,"write.exe","write.exe","REPORT_CH0.DAR",NULL);
937            _spawnlp(_P_NOWAIT,"write.exe","write.exe","DATA_CH0.DAQ",NULL);
938
939            _spawnlp(_P_NOWAIT,"write.exe","write.exe","REPORT_CH1.DAR",NULL);
940            _spawnlp(_P_NOWAIT,"write.exe","write.exe","DATA_CH1.DAQ",NULL);
941       }
942
943       //   _spawnlp(_P_DETACH,"write.exe","write.exe","DAQDATA.DAQ",NULL);
944
945  //   system("write.exe DAQDATA.DAQ");
946
947  }
948
949
950  void CSolartesterView::OnTimer(UINT nIDEvent)
951  {
952       if( (nIDEvent==1) && (m_bEnable==TRUE) )
953       {
954            Beep(1000,500);
955
956            Invalidate();
957
958
959       }
960
961       CScrollView::OnTimer(nIDEvent);
962  }
963
964  int CSolartesterView::OnCreate(LPCREATESTRUCT lpCreateStruct)
965  {
966       if (CView::OnCreate(lpCreateStruct) == -1)
967            return -1;
968
969  //   SetTimer(1,5000,NULL);
970
971       return 0;
972  }
973
974  void CSolartesterView::OnDestroy()
```

```
975  {
976      CScrollView::OnDestroy();
977
978
979      // TODO: Add your message handler code here
980      KillTimer(1);
981  }
982
983
984  void CSolartesterView::OnUpdateBegin(CCmdUI* pCmdUI)
985  {
986      // TODO: Add your command update UI handler code here
987
988  //   pCmdUI->SetCheck(m_bEnable);
989
990      pCmdUI->Enable(m_bEnable);
991  }
992
993  void CSolartesterView::OnVmax()
994  {
995      CPara dlg;
996
997      do
998      {
999        dlg.DoModal();
1000
1001       if( dlg.m_szCh0_id.IsEmpty() || dlg.m_szCh1_id.IsEmpty() )
1002          {
1003            AfxMessageBox( "警告:批号值不能为空!\n\n请重新输入",MB_ICONERROR);
1004          }
1005     }while( dlg.m_szCh0_id.IsEmpty() || dlg.m_szCh1_id.IsEmpty() );
1006
1007     FILE* fp2;
1008     fp2=fopen("PARAS.DAP","w");
1009
1010     if(fp2 != NULL)
1011     {
1012         fprintf(fp2,"m_szCh0_id = %s\n",(const char*)dlg.m_szCh0_id);
1013         fprintf(fp2,"m_szCh1_id = %s\n",(const char*)dlg.m_szCh1_id);
1014         fprintf(fp2,"m_nAverage = %d\n",dlg.m_nAverage);
1015         fprintf(fp2,"m_fVmax = %f\n",dlg.m_fVmax);
1016         fprintf(fp2,"m_nSample = %d\n",dlg.m_nSample);
1017         fprintf(fp2,"m_nDelay = %d\n",dlg.m_nDelay);
1018         fprintf(fp2,"m_fSun = %f\n",dlg.m_fSun);
1019         fprintf(fp2,"m_fSize = %f\n",dlg.m_fSize);
1020         fprintf(fp2,"m_fT_dry = %f\n",dlg.m_fT_dry);
1021         fprintf(fp2,"m_fT_wet = %f\n",dlg.m_fT_wet);
1022         fprintf(fp2,"m_fHum = %f\n",dlg.m_fHum);
1023         fprintf(fp2,"m_nAmp_range = %d\n",dlg.m_nAmp_range);
1024         fclose(fp2);
1025     }
1026
1027     Ch0_id = dlg.m_szCh0_id;
1028     Ch1_id = dlg.m_szCh1_id;
1029     Ch0_id.FreeExtra();
```

```
1030        Ch1_id.FreeExtra();
1031        Average = dlg.m_nAverage;
1032        Vmax = dlg.m_fVmax;
1033        Sample = dlg.m_nSample;
1034        Delay = dlg.m_nDelay;
1035        Sun = dlg.m_fSun;
1036        Size = dlg.m_fSize;
1037        T_dry = dlg.m_fT_dry;
1038        T_wet = dlg.m_fT_wet;
1039        Hum = dlg.m_fHum;
1040        Amp_range = dlg.m_nAmp_range;
1041
1042        switch(Amp_range)
1043        {
1044            case(0)://2A
1045            {
1046                amp_card_gain=1;
1047                amp_board_gain=1;
1048                break;
1049            }
1050            case(1)://1A
1051            {
1052                amp_card_gain=2;
1053                amp_board_gain=1;
1054                break;
1055            }
1056            case(2)://500mA
1057            {
1058                amp_card_gain=4;
1059                amp_board_gain=1;
1060                break;
1061            }
1062            case(3)://200mA
1063            {
1064                amp_card_gain=1;
1065                amp_board_gain=10;
1066                break;
1067            }
1068            case(4)://100mA
1069            {
1070                amp_card_gain=2;
1071                amp_board_gain=10;
1072                break;
1073            }
1074            case(5)://50mA
1075            {
1076                amp_card_gain=4;
1077                amp_board_gain=10;
1078                break;
1079            }
1080            case(6)://20mA
1081            {
1082                amp_card_gain=1;
1083                amp_board_gain=100;
1084                break;
```

```
1085            }
1086            case(7)://10mA
1087            {
1088                amp_card_gain=2;
1089                amp_board_gain=100;
1090                break;
1091            }
1092            case(8)://5mA
1093            {
1094                amp_card_gain=4;
1095                amp_board_gain=100;
1096                break;
1097            }
1098            case(9)://AUTO
1099            {
1100                amp_card_gain=1;
1101                amp_board_gain=1;
1102                break;
1103            }
1104            default:
1105            {
1106                amp_card_gain=1;
1107                amp_board_gain=1;
1108                break;
1109            }
1110        }
1111
1112        if( Vmax <= AD_RANGE/4 )
1113        {
1114            volt_card_gain=4;
1115        }
1116        else
1117        {
1118            if( Vmax <= AD_RANGE/2 )
1119            {
1120                volt_card_gain=2;
1121            }
1122            else
1123            {
1124                volt_card_gain=1;
1125            }
1126        }
1127
1128        Invalidate();
1129
1130        TRACE("dlg.m_nAmp_range=%d\n",dlg.m_nAmp_range);
1131
1132
1133
1134    }
1135
1136    void CSolartesterView::OnEnd()
1137    {
1138        // TODO: Add your command handler code here
1139        int msg = MessageBox( "真的结束测试吗?","结束测试",MB_YESNO );
```

```
1140
1141        if(msg == IDNO )
1142        {
1143            return;
1144        }
1145        m_bEnable=TRUE;
1146    }
1147
1148    void CSolartesterView::OnUpdateEnd(CCmdUI* pCmdUI)
1149    {
1150        // TODO: Add your command update UI handler code here
1151
1152        pCmdUI->Enable(!m_bEnable);
1153    }
1154
1155    /* analog output
1156         data:0--4095  */
1157    int CSolartesterView::ao(int ch,UINT data)
1158    {
1159      switch(ch)
1160      {
1161        case(0):
1162        {
1163            _outpw(DA_CH0,(unsigned short) data);
1164
1165                return(0);
1166        }
1167
1168        case(1):
1169        {
1170            _outpw(DA_CH1,(unsigned short) data);
1171
1172                return(0);
1173        }
1174
1175        default:
1176          return(-1);
1177      }
1178    }
1179
1180    /* set A/D channel
1181         ad_ch_no:0--15  */
1182    int CSolartesterView::ad_set_channel(int ad_ch_no)
1183    {
1184      if( (ad_ch_no>=0) && (ad_ch_no<=15) )
1185      {
1186        _outp(AD_CH,ad_ch_no);
1187        return(0);
1188      }
1189      else
1190      {
1191        return(-1);
1192      }
1193    }
1194
```

```
1195  /* set A/D gain
1196       ad_gain:1,2,4,8,16 */
1197  int CSolartesterView::ad_set_gain(int ad_gain)
1198  {
1199    switch(ad_gain)
1200    {
1201      case(1):
1202      {
1203        _outp(AD_GAIN,0);     // set gain to X1
1204        return(0);
1205      }
1206      case(2):
1207      {
1208        _outp(AD_GAIN,1);     // set gain to X2
1209        return(0);
1210      }
1211      case(4):
1212      {
1213        _outp(AD_GAIN,2);     // set gain to X4
1214        return(0);
1215      }
1216      case(8):
1217      {
1218        _outp(AD_GAIN,3);     // set gain to X8
1219        return(0);
1220      }
1221      case(16):
1222      {
1223        _outp(AD_GAIN,4);     // set gain to X16
1224        return(0);
1225      }
1226      default:
1227        return(-1);
1228    }
1229  }
1230
1231  /* set A/D mode
1232       ad_mode:0--external trigger,software polling
1233            1--software trigger,software polling(default)
1234            2--timer trigger, DMA transfer
1235            3--external trigger, DMA transfer
1236            4--external trigger, interrupt transfer
1237            5--software trigger, interrupt transfer
1238            6--timer trigger, interrupt transfer
1239  */
1240  int CSolartesterView::ad_set_mode(int ad_mode)
1241  {
1242    if( (ad_mode>=0) && (ad_mode<=6) )
1243    {
1244      _outp(AD_MODE,ad_mode);
1245      return(0);
1246    }
1247    else
1248    {
1249      return(-1);
```

```
1250      }
1251    }
1252
1253    /* A/D software trig command */
1254    int CSolartesterView::ad_soft_trig(void)
1255    {
1256      _outp(AD_TRIG,0);
1257
1258      return(0);
1259    }
1260
1261    /* set A/D gain
1262         ad_gain:1,2,4,8,16 */
1263    int CSolartesterView::ad_acquire(int *ad_data)
1264    {
1265      union int2byte
1266      {
1267        UINT  value;
1268        unsigned char ubyte[4];
1269      };
1270
1271      union int2byte result;
1272      int i=0;
1273         result.ubyte[3]=0;
1274         result.ubyte[2]=0;
1275
1276         ad_soft_trig();
1277
1278      do
1279      {
1280        result.ubyte[1]=(unsigned char)_inp(AD_HIGH);
1281        i++;
1282      } while( (i<=1000) && (result.ubyte[1]>15) );
1283
1284      if(i==1000)
1285      {
1286        return(-1);
1287      }
1288      else
1289      {
1290        result.ubyte[0]=(unsigned char)_inp(AD_LOW);
1291        result.value &= 0x00000fff;
1292             (*ad_data)=result.value;
1293
1294        return(0);
1295      }
1296    }
1297
1298    /* digital output
1299                 port_number:DO_LOW_BYTE or DO_HIGH_BYTE
1300                 data:value will be written to digital output port */
1301    int CSolartesterView::Do(int port_number,unsigned char data)
1302    {
1303        switch( port_number )
1304        {
```

```
1305            case( DO_LOW_BYTE ):
1306            {
1307                _outp(DO_LOW,data);
1308                return(0);
1309                break;
1310            }
1311            case( DO_HIGH_BYTE ):
1312            {
1313                _outp(DO_HIGH,data);
1314                return(0);
1315                break;
1316            }
1317            default:
1318            {
1319                return(-1);
1320                break;
1321            }
1322        }
1323    }
1324
1325    void CSolartesterView::OnDraw(CDC* pDC)
1326    {
1327        CSolartesterDoc* pDoc = GetDocument();
1328        ASSERT_VALID(pDoc);
1329
1330        COLORREF crRed = RGB(255,0,0);
1331        COLORREF crGray = RGB(192,127,127);
1332        COLORREF crYellow = RGB(192,0,0);
1333        COLORREF crBlue = RGB(0,0,240);
1334        COLORREF crBlack = RGB(0,0,0);
1335
1336        CPen newPen1;
1337        CPen newPen2;
1338        CPen newPen3;
1339        CPen* oldPen;
1340
1341        CBrush  newBrush1;
1342        CBrush  newBrush2;
1343        CBrush* oldBrush;
1344
1345        CFont  newFont;
1346
1347        newFont.CreateFont(400,0,0,0,FW_NORMAL,FALSE,FALSE,0,
1348                           ANSI_CHARSET,OUT_DEFAULT_PRECIS,
1349                                                CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
1350                                                DEFAULT_PITCH|FF_SWISS,"Times New Roman");
1351        CFont* pOldFont=(CFont*)pDC->SelectObject(&newFont);
1352
1353        BlkGraph(pDC);
1354
1355        newPen1.CreatePen(PS_SOLID,1,crRed);
1356        newBrush1.CreateSolidBrush(crRed);
1357
1358        oldPen = pDC -> SelectObject(&newPen1);
1359        oldBrush = pDC -> SelectObject(&newBrush1);
```

```
1360
1361        for(jj=0;jj<i && i<=1001;jj++)
1362        {
1363            pDC->Rectangle(x_ch0[jj]-60,y_ch0[jj]-60,x_ch0[jj]+60,y_ch0[jj]+60);
1364        }
1365
1366        x1=X_ORG+X_LEN/20*4;y1=Y_END-600;
1367        for(jj=0;jj<30;jj++)
1368        {
1369            pDC->Rectangle(x1-60+jj*120,y1-60,x1+60+jj*120,y1+60);
1370        }
1371
1372        newPen2.CreatePen(PS_SOLID,1,crGray);
1373        pDC -> SelectObject(&newPen2);
1374
1375        for(jj=0;jj<i-1 && i<=1001;jj++)
1376        {
1377            pDC->MoveTo(x_ch0[jj],y_ch0[jj]);
1378            pDC->LineTo(x_ch0[jj+1],y_ch0[jj+1]);
1379        }
1380
1381        newPen3.CreatePen(PS_SOLID,1,crYellow);
1382        newBrush2.CreateSolidBrush(crYellow);
1383
1384        pDC -> SelectObject(&newBrush2);
1385        pDC -> SelectObject(&newPen3);
1386
1387        for(jj=0;jj<i && i<=1001;jj++)
1388        {
1389            pDC->Rectangle(x_ch1[jj]-45,y_ch1[jj]-45,x_ch1[jj]+45,y_ch1[jj]+45);
1390        }
1391
1392        x1=X_ORG+X_LEN/20*11;y1=Y_END-600;
1393        for(jj=0;jj<40;jj++)
1394        {
1395            pDC->Rectangle(x1-45+jj*90,y1-45,x1+60+jj*90,y1+45);
1396        }
1397
1398        pDC -> SelectObject(&newPen2);
1399
1400        for(jj=0;jj<i-1 && i<=1001;jj++)
1401        {
1402            pDC->MoveTo(x_ch1[jj],y_ch1[jj]);
1403            pDC->LineTo(x_ch1[jj+1],y_ch1[jj+1]);
1404        }
1405
1406        pDC->SetTextColor( crBlue );
1407
1408        char str[50];
1409
1410        sprintf(str,"X AXIS:  I = %6.0lf mA",a_range);
1411
1412        x1=X_ORG+X_LEN/20*14;y1=Y_END-400;
1413        pDC->TextOut(x1,y1,str);
1414
```

```
1415        sprintf(str,"Y AXIS:  U = %4.1lf V",v_range);
1416        x1+=4000;
1417        pDC->TextOut(x1,y1,str);
1418
1419        pDC->SetTextColor( crBlack );
1420
1421        sprintf(str,"Average = %5d",Average);
1422        x1=X_ORG;y1=Y_END-800;
1423        pDC->TextOut(x1,y1,str);
1424
1425        sprintf(str,"Vmax = %-6.3f V",Vmax);
1426        x1+=2800;
1427        pDC->TextOut(x1,y1,str);
1428
1429        sprintf(str,"Sample = %5d",Sample);
1430        x1+=2700;
1431        pDC->TextOut(x1,y1,str);
1432
1433        sprintf(str,"Delay = %5d ms",Delay);
1434        x1+=2700;
1435        pDC->TextOut(x1,y1,str);
1436
1437        sprintf(str,"Size = %-7.3lf cm**2",Size);
1438        x1+=3000;
1439        pDC->TextOut(x1,y1,str);
1440
1441        sprintf(str,"Sun = %-6.3lf sun",Sun);
1442        x1+=3400;
1443        pDC->TextOut(x1,y1,str);
1444
1445        sprintf(str,"T_dry = %-5.1lf degree",T_dry);
1446        x1+=2800;
1447        pDC->TextOut(x1,y1,str);
1448
1449        sprintf(str,"T_wet = %-5.1lf degree",T_wet);
1450        x1+=3200;
1451        pDC->TextOut(x1,y1,str);
1452
1453        sprintf(str,"Humidity = %-5.1lf %%",Hum);
1454        x1+=3200;
1455        pDC->TextOut(x1,y1,str);
1456
1457        pDC->SetTextColor( crRed );
1458
1459        sprintf(str,"FF0 = %6.3lf",FF_ch0);
1460        x1=X_ORG;y1=Y_END-1200;
1461        pDC->TextOut(x1,y1,str);
1462
1463        sprintf(str,"Eff0 = %7.3lf %%",Eff_ch0);
1464        x1+=2200;
1465        pDC->TextOut(x1,y1,str);
1466
1467        sprintf(str,"Jsc0 = %6.4lf A/(cm**2)",Jsc_ch0);
1468        x1+=2500;
1469        pDC->TextOut(x1,y1,str);
```

```
1470
1471        sprintf(str,"Isc0 = %9.6lf A",Isc_ch0);
1472        x1+=3700;
1473        pDC->TextOut(x1,y1,str);
1474
1475        sprintf(str,"Voc0 = %8.4lf V",Voc_ch0);
1476        x1+=3000;
1477        pDC->TextOut(x1,y1,str);
1478
1479        sprintf(str,"Imp0 = %9.6lf A",Imp_ch0);
1480        x1+=2800;
1481        pDC->TextOut(x1,y1,str);
1482
1483        sprintf(str,"Vmp0 = %8.4lf V",Vmp_ch0);
1484        x1+=3000;
1485        pDC->TextOut(x1,y1,str);
1486
1487        sprintf(str,"Pmax0 = %9.6lf W",Pmax_ch0);
1488        x1+=2800;
1489        pDC->TextOut(x1,y1,str);
1490
1491        x1=X_ORG+X_LEN/20*0;y1=Y_END-400;
1492        pDC->TextOut(x1,y1,"CH0:"+Ch0_id);
1493
1494        pDC->SetTextColor( crYellow );
1495
1496        sprintf(str,"FF1 = %6.3lf",FF_ch1);
1497        x1=X_ORG;y1=Y_END-1600;
1498        pDC->TextOut(x1,y1,str);
1499
1500        sprintf(str,"Eff1 = %7.3lf %%",Eff_ch1);
1501        x1+=2200;
1502        pDC->TextOut(x1,y1,str);
1503
1504        sprintf(str,"Jsc1 = %6.4lf A/(cm**2)",Jsc_ch1);
1505        x1+=2500;
1506        pDC->TextOut(x1,y1,str);
1507
1508        sprintf(str,"Isc1 = %9.6lf A",Isc_ch1);
1509        x1+=3700;
1510        pDC->TextOut(x1,y1,str);
1511
1512        sprintf(str,"Voc1 = %8.4lf V",Voc_ch1);
1513        x1+=3000;
1514        pDC->TextOut(x1,y1,str);
1515
1516        sprintf(str,"Imp1 = %9.6lf A",Imp_ch1);
1517        x1+=2800;
1518        pDC->TextOut(x1,y1,str);
1519
1520        sprintf(str,"Vmp1 = %8.4lf V",Vmp_ch1);
1521        x1+=3000;
1522        pDC->TextOut(x1,y1,str);
1523
1524        sprintf(str,"Pmax1 = %9.6lf W",Pmax_ch1);
```

```
1525        x1+=2800;
1526        pDC->TextOut(x1,y1,str);
1527
1528        x1=X_ORG+X_LEN/20*7;y1=Y_END-400;
1529        pDC->TextOut(x1,y1,"CH1:"+Ch1_id);
1530
1531        pDC -> SelectObject(oldPen);
1532        pDC -> SelectObject(oldBrush);
1533        pDC -> SelectObject(pOldFont);
1534    }
1535
1536
1537    void CSolartesterView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
1538    {
1539        pDC->SetMapMode(MM_HIMETRIC);
1540
1541        CScrollView::OnPrepareDC(pDC, pInfo);
1542    }
1543
1544    void  CSolartesterView::OnInitialUpdate()
1545    {
1546        CScrollView::OnInitialUpdate();
1547        CSize sizeTotal(28000,20000);
1548        CSize sizePage(sizeTotal.cx/2,sizeTotal.cx/2);
1549        CSize sizeLine(sizeTotal.cx/50,sizeTotal.cx/50);
1550
1551        SetScrollSizes(MM_HIMETRIC,sizeTotal,sizePage,sizeLine);
1552    }
1553
1554    void CSolartesterView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
1555    {
1556        switch(nChar)
1557        {
1558            case VK_HOME:
1559                OnVScroll(SB_TOP,0,NULL);
1560                OnHScroll(SB_LEFT,0,NULL);
1561                break;
1562
1563            case VK_END:
1564                OnVScroll(SB_BOTTOM,0,NULL);
1565                OnHScroll(SB_RIGHT,0,NULL);
1566                break;
1567
1568            case VK_UP:
1569                OnVScroll(SB_LINEUP,0,NULL);
1570                break;
1571
1572            case VK_DOWN:
1573                OnVScroll(SB_LINEDOWN,0,NULL);
1574                break;
1575
1576            case VK_PRIOR:
1577                OnVScroll(SB_PAGEUP,0,NULL);
1578                break;
1579
```

```
1580            case VK_NEXT:
1581                OnVScroll(SB_PAGEDOWN, 0, NULL);
1582                break;
1583
1584            case VK_LEFT:
1585                OnHScroll(SB_LINELEFT, 0, NULL);
1586                break;
1587
1588            case VK_RIGHT:
1589                OnHScroll(SB_LINERIGHT, 0, NULL);
1590                break;
1591
1592            default:
1593                break;
1594        }
1595    }
1596
1597    void CSolartesterView::OnSun()
1598    {
1599        int meter_sum=0;
1600        double sun_factor;
1601        double meter_volt;
1602
1603        int sun_gain=1;
1604        sun_factor=AD_RANGE/(2048*sun_gain);
1605        ad_set_channel(METER_CH);
1606        ad_set_gain(sun_gain);
1607
1608        for(int k=0;k<METER_CH_AVERAGE;k++)
1609        {
1610            ad_acquire(&ad);
1611            meter_sum+=ad;
1612     }
1613
1614        ad_set_gain(amp_card_gain);
1615
1616     meter_sum /= METER_CH_AVERAGE;
1617        meter_volt = (meter_sum-2048)*sun_factor;
1618        meter = meter_volt * 1000/(METER_AMP_FACTOR*METER_CONST);
1619
1620        meter=(-meter);
1621
1622        char str1[50];
1623        sprintf(str1,"入射光强 = %6.4f SUN\n\n保存结果吗?",meter);
1624
1625        int msg = MessageBox( str1,"入射光强",MB_YESNO );
1626
1627        if(msg == IDYES )
1628        {
1629            Sun = meter;
1630
1631            FILE* fp3;
1632            fp3=fopen("PARAS.DAP","w");
1633
1634            if(fp3 != NULL)
```

```
1635            {
1636                fprintf(fp3,"m_szCh0_id = %s\n",(const char*)Ch0_id);
1637                fprintf(fp3,"m_szCh1_id = %s\n",(const char*)Ch1_id);
1638                fprintf(fp3,"m_nAverage = %d\n",Average);
1639                fprintf(fp3,"m_fVmax = %f\n",Vmax);
1640                fprintf(fp3,"m_nSample = %d\n",Sample);
1641                fprintf(fp3,"m_nDelay = %d\n",Delay);
1642                fprintf(fp3,"m_fSun = %f\n",Sun);
1643                fprintf(fp3,"m_fSize = %f\n",Size);
1644                fprintf(fp3,"m_fT_dry = %f\n",T_dry);
1645                fprintf(fp3,"m_fT_wet = %f\n",T_wet);
1646                fprintf(fp3,"m_fHum = %f\n",Hum);
1647                fprintf(fp3,"m_nAmp_range = %d\n",Amp_range);
1648                fclose(fp3);
1649            }
1650
1651            Invalidate();
1652        }
1653    }
1654
1655    void CSolartesterView::OnTemp()
1656    {
1657        int t_dry_sum=0;
1658        int t_wet_sum=0;
1659        double ad_factor;
1660        double t_dry_volt;
1661        double t_wet_volt;
1662
1663
1664        const double hum_table_0_5[41] =
1665            { 0, 0, 0,89,89,90,90,91,91,92,92,92,93,93,93,93,94,94,94,94,94,    // 0-20
1666                95,95,95,95,95,95,95,96,96,96,96,96,96,96,96,97,97,97,97,97 }; //21-40
1667        const double hum_table_1_0[41] =
1668            {    0, 0, 0,77,78,79,80,81,82,83,84,84,85,86,86,87,87,88,88,88,89,    // 0-20
1669                89,89,90,90,90,91,91,91,92,92,92,92,92,92,93,93,93,93,93,93 }; //21-40
1670        const double hum_table_1_5[41] =
1671            {    0, 0, 0,66,68,70,71,72,74,75,76,77,77,78,79,80,81,81,82,82,83,    // 0-20
1672                84,84,85,85,86,86,86,87,87,88,88,88,88,89,89,89,89,89,90,90 }; //21-40
1673        const double hum_table_2_0[41] =
1674            {    0, 0, 0,55,57,59,61,63,65,66,68,69,70,71,73,74,75,76,76,77,78,    // 0-20
1675                79,79,80,80,81,82,82,83,83,83,84,84,84,85,85,85,86,86,86,87 }; //21-40
1676        const double hum_table_2_5[41] =
1677            {    0, 0, 0,44,46,49,52,54,56,58,60,61,63,64,66,67,69,70,71,72,73,    // 0-20
1678                74,74,75,76,76,77,78,78,79,79,80,80,80,81,81,82,82,82,82,83 }; //21-40
1679        const double hum_table_3_0[41] =
1680            {    0, 0, 0,33,36,39,42,44,47,50,52,54,56,57,59,60,62,64,65,66,67,    // 0-20
1681                68,69,70,71,72,73,74,74,75,75,76,77,77,78,78,78,79,79,79,80 }; //21-40
1682        const double hum_table_3_5[41] =
1683            {    0, 0, 0,22,26,29,33,36,39,42,44,47,49,51,53,54,56,58,59,61,62,    // 0-20
1684                63,64,65,66,68,69,70,70,71,72,72,73,74,75,75,75,76,76,76,77 }; //21-40
1685        const double hum_table_4_0[41] =
1686            {    0, 0, 0,11,15,20,23,27,30,34,37,39,42,44,46,48,50,52,54,56,57,    // 0-20
1687                58,60,61,62,63,64,66,66,67,68,69,69,70,71,71,72,72,73,73,74 }; //21-40
1688        const double hum_table_4_5[41] =
1689            {    0, 0, 0, 0, 5,10,14,18,22,26,29,32,35,38,40,42,44,46,48,50,52,    // 0-20
```

```
1690                53,55,57,58,59,60,62,63,63,64,65,66,67,68,68,69,69,70,70,71 }; //21-40
1691        const double hum_table_5_0[41] =
1692            {    0, 0, 0, 0, 0, 0, 5,10,14,18,22,25,28,31,34,37,39,41,43,45,47,    // 0-20
1693                48,50,52,53,55,56,58,59,59,60,61,62,63,64,65,65,66,67,67,68 }; //21-40
1694        const double hum_table_5_5[41] =
1695            {    0, 0, 0, 0, 0, 0, 0, 1, 6,10,14,18,21,25,28,30,33,35,38,41,43,    // 0-20
1696                44,46,48,49,51,52,54,55,56,57,58,59,60,61,62,62,63,64,64,65 }; //21-40
1697        const double hum_table_6_0[41] =
1698            {    0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 7,11,15,18,21,25,27,30,33,36,38,    // 0-20
1699                40,41,43,45,46,48,50,51,52,53,54,55,56,57,58,59,60,61,61,62 }; //21-40
1700        const double hum_table_6_5[41] =
1701            {    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 8,12,15,19,22,25,28,31,33,    // 0-20
1702                35,37,39,40,42,44,46,47,48,50,51,52,53,54,55,56,57,58,59,60 }; //21-40
1703        const double hum_table_7_0[41] =
1704            {    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 6, 9,13,16,19,23,26,28,    // 0-20
1705                31,33,34,36,38,40,42,43,44,46,47,49,50,51,52,53,54,55,56,57 }; //21-40
1706        const double hum_table_7_5[41] =
1707            {    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 7,11,14,18,21,23,    // 0-20
1708                26,28,30,32,34,36,38,39,41,43,44,46,47,48,49,50,51,52,53,54 }; //21-40
1709
1710        int temp_gain=1;
1711        ad_factor=AD_RANGE/(2048*temp_gain);
1712        ad_set_gain(temp_gain);
1713
1714        //ad_set_channel(T_DRY_CH);
1715        for(int k=0;k<TEMP_CH_AVERAGE;k++)
1716        {
1717            ad_set_channel(T_DRY_CH);
1718            ad_acquire(&ad);
1719            t_dry_sum+=ad;
1720
1721            ad_set_channel(T_WET_CH);
1722            ad_acquire(&ad);
1723            t_wet_sum+=ad;
1724    }
1725        ad_set_gain(amp_card_gain);
1726
1727        t_dry_sum /= TEMP_CH_AVERAGE;
1728        t_dry_volt = (t_dry_sum-2048)*ad_factor;
1729        t_dry = t_dry_volt * 1000000/T_DRY_RES - 273.15;
1730
1731        t_wet_sum /= TEMP_CH_AVERAGE;
1732        t_wet_volt = (t_wet_sum-2048)*ad_factor;
1733        t_wet = t_wet_volt * 1000000/T_WET_RES - 273.15;
1734
1735        hum=0.0;
1736
1737        if( (t_dry>t_wet) && (t_dry>=0) && (t_dry<=40) )
1738        {
1739            if( (t_dry-t_wet>0.25) && (t_dry-t_wet<=0.75) )
1740            {
1741                hum=hum_table_0_5[(int)t_dry];
1742            }
1743            if( (t_dry-t_wet>0.75) && (t_dry-t_wet<=1.25) )
1744            {
```

```
1745                hum=hum_table_1_0[(int)t_dry];
1746            }
1747            if( (t_dry-t_wet>1.25) && (t_dry-t_wet<=1.75) )
1748            {
1749                hum=hum_table_1_5[(int)t_dry];
1750            }
1751            if( (t_dry-t_wet>1.75) && (t_dry-t_wet<=2.25) )
1752            {
1753                hum=hum_table_2_0[(int)t_dry];
1754            }
1755            if( (t_dry-t_wet>2.25) && (t_dry-t_wet<=2.75) )
1756            {
1757                hum=hum_table_2_5[(int)t_dry];
1758            }
1759            if( (t_dry-t_wet>2.75) && (t_dry-t_wet<=3.25) )
1760            {
1761                hum=hum_table_3_0[(int)t_dry];
1762            }
1763            if( (t_dry-t_wet>3.25) && (t_dry-t_wet<=3.75) )
1764            {
1765                hum=hum_table_3_5[(int)t_dry];
1766            }
1767            if( (t_dry-t_wet>3.75) && (t_dry-t_wet<=4.25) )
1768            {
1769                hum=hum_table_4_0[(int)t_dry];
1770            }
1771            if( (t_dry-t_wet>4.25) && (t_dry-t_wet<=4.75) )
1772            {
1773                hum=hum_table_4_5[(int)t_dry];
1774            }
1775            if( (t_dry-t_wet>4.75) && (t_dry-t_wet<=5.25) )
1776            {
1777                hum=hum_table_5_0[(int)t_dry];
1778            }
1779            if( (t_dry-t_wet>5.25) && (t_dry-t_wet<=5.75) )
1780            {
1781                hum=hum_table_5_5[(int)t_dry];
1782            }if( (t_dry-t_wet>5.75) && (t_dry-t_wet<=6.25) )
1783            {
1784                hum=hum_table_6_0[(int)t_dry];
1785            }
1786            if( (t_dry-t_wet>6.25) && (t_dry-t_wet<=6.75) )
1787            {
1788                hum=hum_table_6_5[(int)t_dry];
1789            }
1790            if( (t_dry-t_wet>6.75) && (t_dry-t_wet<=7.25) )
1791            {
1792                hum=hum_table_7_0[(int)t_dry];
1793            }
1794            if( (t_dry-t_wet>7.25) && (t_dry-t_wet<=7.75) )
1795            {
1796                hum=hum_table_7_5[(int)t_dry];
1797            }
1798        }
1799
```

```
1800        char str1[100];
1801        sprintf(str1,
1802            "干表温度 = %5.1f ℃\n湿表温度 = %5.1f ℃\n相对湿度 = %5.1f %  \n\n保存结果
1803    吗?\n",t_dry,t_wet,hum);
1804
1805        int msg = MessageBox( str1,"温度/湿度",MB_YESNO );
1806
1807        if(msg == IDYES )
1808        {
1809            T_dry=t_dry;
1810            T_wet=t_wet;
1811            Hum=hum;
1812
1813            FILE* fp4;
1814            fp4=fopen("PARAS.DAP","w");
1815
1816            if(fp4 != NULL)
1817            {
1818                fprintf(fp4,"m_szCh0_id = %s\n",(const char*)Ch0_id);
1819                fprintf(fp4,"m_szCh1_id = %s\n",(const char*)Ch1_id);
1820                fprintf(fp4,"m_nAverage = %d\n",Average);
1821                fprintf(fp4,"m_fVmax = %f\n",Vmax);
1822                fprintf(fp4,"m_nSample = %d\n",Sample);
1823                fprintf(fp4,"m_nDelay = %d\n",Delay);
1824                fprintf(fp4,"m_fSun = %f\n",Sun);
1825                fprintf(fp4,"m_fSize = %f\n",Size);
1826                fprintf(fp4,"m_fT_dry = %f\n",T_dry);
1827                fprintf(fp4,"m_fT_wet = %f\n",T_wet);
1828                fprintf(fp4,"m_fHum = %f\n",Hum);
1829                fprintf(fp4,"m_nAmp_range = %d\n",Amp_range);
1830                fclose(fp4);
1831            }
1832
1833            Invalidate();
1834        }
1835    }
1836
1837
1838
1839
1840    void CSolartesterView::OnTrace()
1841    {
1842        CRotate dlg;
1843
1844        dlg.m_fAngle = 1;
1845        dlg.m_nDir = 0;     // 0 = clockwise,1 = counter clockwise
1846
1847      dlg.DoModal();
1848    }
```

13、文件 StdAfx.h

```
1    // stdafx.h : include file for standard system include files,
2    //  or project specific include files that are used frequently, but
```

```
3   //      are changed infrequently
4   //
5
6   #if !defined(AFX_STDAFX_H__49A03BCB_3FD8_11D3_B89D_0080C875FA7E__INCLUDED_)
7   #define AFX_STDAFX_H__49A03BCB_3FD8_11D3_B89D_0080C875FA7E__INCLUDED_
8
9   #if _MSC_VER > 1000
10  #pragma once
11  #endif // _MSC_VER > 1000
12
13  #define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers
14
15  #include <afxwin.h>         // MFC core and standard components
16  #include <afxext.h>         // MFC extensions
17  #include <afxdisp.h>        // MFC Automation classes
18  #include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common Controls
19  #ifndef _AFX_NO_AFXCMN_SUPPORT
20  #include <afxcmn.h>             // MFC support for Windows Common Controls
21  #endif // _AFX_NO_AFXCMN_SUPPORT
22
23
24  //{{AFX_INSERT_LOCATION}}
25  // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
26
27  #endif // !defined(AFX_STDAFX_H__49A03BCB_3FD8_11D3_B89D_0080C875FA7E__INCLUDED_)
```

14、文件 StdAfx.cpp

```
1   // stdafx.cpp : source file that includes just the standard includes
2   //   solartester.pch will be the pre-compiled header
3   //   stdafx.obj will contain the pre-compiled type information
4
5   #include "stdafx.h"
```

附录 H 微控制器源程序清单

文件 solartv1.c

```
/* This is the root module for SOLAR TRACER
     in MCU side.
   Version 1.0, software delay, get counts from T0/T1.
             speed controlled by T0.
   Function dly100ms() calibrated.
   Stepping motor model:75BF003(1.5 degree/3.0 degree).
   Stepping motor driver model:SH-3F090M(40 division).
   Written by CHUAN-DONG XIA, January 8th, 2000.
*/

# include "reg51.h"
# include "intrins.h"
# include "stdio.h"
# include "math.h"
# include "stdlib.h"
# include "string.h"

# define uchar unsigned char
# define uint  unsigned int
# define ulong unsigned long int

/* UNIT_PULSE:pulses for stepping motor with respect
     to one carrier movement unit (1.5/40=0.0375 degree). */
# define UNIT_PULSE 1

/* SPEED_PPS:pulses per second for stepping motor with
     respect to one speed unit (4.5 degree/second). */
# define SPEED_PPS 5*UNIT_PULSE * 24

# define MOVE_CONST 80

# define X_AXIS 0
# define Y_AXIS 1
# define Z_AXIS 2

# define FWD 0
# define BWD 1

sbit  CP_X=P1^2;
sbit  DIR_X=P1^3;
sbit  CP_Y=P1^4;
sbit  DIR_Y=P1^5;
sbit  CP_Z=P1^6;
sbit  DIR_Z=P1^7;
```

```
46  sbit  BUZZER=P3^3;
47
48  uchar srlrcv(void);
49  void  srlsend(uchar samptime);
50
51  void  newinitsrl(void);
52
53  void  rcvparapc(uchar parabuf[]);
54  void  newsendcntpc(uchar countvalue[]);
55
56  void  newinitcpu(void);
57  void  newinitcnter(void);
58  void  newcntmenu(uchar smp_time[]);
59
60  void  move(uchar axis,uchar dir,uint units,uchar spd);
61
62  void  dly10us(void);
63  void  dly100us(void);
64  uchar dly1ms(uint n_1ms);
65  void  dly100ms(void);
66
67  /* This function delays 10us (at 12MHz) */
68  void dly10us(void)
69  {
70    _nop_();
71    _nop_();
72    _nop_();
73    _nop_();
74    _nop_();
75    _nop_();
76  }
77
78  /* This function delays 100us (at 12MHz) */
79  void dly100us(void)
80  {
81    dly10us();
82    dly10us();
83    dly10us();
84    dly10us();
85    dly10us();
86    dly10us();
87    dly10us();
88    dly10us();
89    dly10us();
90    _nop_();
91    _nop_();
92    _nop_();
93    _nop_();
94    _nop_();
95    _nop_();
96  }
```

```
97
98  /* This function delays n_1ms*1ms (at 12MHz) */
99  uchar dly1ms(uint n_1ms)
100 {
101   uint k;
102
103   if( (n_1ms>65500) || (n_1ms==0) )
104   {
105     return(0xff);
106   }
107
108   for(k=0;k<n_1ms;k++)
109   {
110     dly100us();
111     dly100us();
112     dly100us();
113     dly100us();
114     dly100us();
115     dly100us();
116     dly100us();
117     dly100us();
118     dly100us();
119     dly100us();
120   }
121
122   return(0);
123 }
124
125 /* This function delays 100ms (at 12MHz) */
126 void dly100ms(void)
127 {
128   uint k;
129
130   for(k=0;k<885;k++)
131   {
132     dly100us();
133   }
134 }
135
136
137 void newinitsrl(void)
138 {
139   TH1=0xe6;
140   TL1=0xe6;
141   SCON=0x50;
142   PCON=0x00;
143
144   TR1=1;
145 }
146
147 /* This function sends one byte to serial port.*/
```

```
148  void srlsend(uchar charsend)
149  {
150    uchar pf;
151
152    do
153    {
154      SBUF=0xaa;
155      while(TI==0);
156      TI=0;
157
158      while(RI==0);
159      RI=0;
160
161    } while((SBUF^0xbb)!=0);
162
163    do
164    {
165      pf=0;
166      SBUF=charsend;
167      pf+=charsend;
168      while(TI==0);
169      TI=0;
170      SBUF=pf;
171      while(TI==0);
172      TI=0;
173      while(RI==0);
174      RI=0;
175    } while(SBUF!=0);
176  }
177
178  /* This function receives one byte from serial port.*/
179  uchar srlrcv(void)
180  {
181    uchar pf;
182    uchar charrcv;
183
184    do
185    {
186      while(RI==0);
187      RI=0;
188    } while((SBUF^0xaa)!=0);
189
190    SBUF=0xbb;
191    while(TI==0);
192    TI=0;
193
194    while(1)
195    {
196      pf=0;
197      while(RI==0);
198      RI=0;
```

```
199       charrcv=SBUF;
200       pf+=charrcv;
201       while(RI==0);
202       RI=0;
203       if((SBUF^pf)==0)
204       {
205         SBUF=0x00;
206         while(TI==0);
207         TI=0;
208         break;
209       }
210       else
211       {
212         SBUF=0xff;
213         while(TI==0);
214         TI=0;
215       }
216     }
217
218     return(charrcv);
219   }
220
221   /* This function sends uesr parameters to PC. */
222   void newsendcntpc(uchar countvalue[])
223   {
224     uchar i;
225
226     for(i=0;i<4;i++)
227     {
228       srlsend(countvalue[i]);
229     }
230   }
231
232   /* This function receives user parameters from PC. */
233   void rcvparapc(uchar parabuf[])
234   {
235     uchar i;
236
237     for(i=0;i<12;i++)
238     {
239       parabuf[i]=srlrcv();
240     }
241   }
242
243   void newinitcnter(void)
244   {
245     TMOD=0x25;
246     TH0=0;
247     TL0=0;
248   }
249
```

```
250  /* This function is for the 'CNT' menu. */
251  /* lsmp_time=100ms. */
252  void newcntmenu(uchar smp_time[])
253  {
254    uint  cntcopy;
255    uchar command;
256    ulong sum;
257    uchar i;
258
259    if( (smp_time[0]==0) || (smp_time[0]>250) )
260    {
261      for(i=0;i<4;i++)
262      {
263        smp_time[i]=0;
264      }
265      return;
266    }
267
268    TR1=0;
269    TR0=0;
270
271    TMOD=0x55;
272    TH0=0;
273    TL0=0;
274    TH1=0;
275    TL1=0;
276
277    TR0=1;
278    TR1=1;
279
280    for(i=0;i<smp_time[0];i++)
281    {
282      dly100ms();
283    }
284
285    TR0=0;
286    TR1=0;
287
288    smp_time[0]=TL0;
289    smp_time[1]=TH0;
290    smp_time[2]=TL1;
291    smp_time[3]=TH1;
292  }
293
294  void newinitcpu(void)
295  {
296    EA=0;
297    ES=0;
298    ET0=0;
299    ET1=0;
300  }
```

```
301
302  void main(void)
303  {
304    uint  i,j,k;
305    uchar parabuf[12];
306    uint  unit_x,unit_y,unit_z;
307    uchar dir_x,dir_y,dir_z;
308    uchar speed,smp_time,mode;
309
310    uchar counts[4];
311
312    newinitcpu();
313
314    newinitcnter();
315
316    newinitsrl();
317
318    BUZZER=0;
319
320    while(1)
321    {
322      rcvparapc(parabuf);
323
324      unit_x=parabuf[0]+parabuf[1]*256;
325      unit_y=parabuf[2]+parabuf[3]*256;
326      unit_z=parabuf[4]+parabuf[5]*256;
327      dir_x=parabuf[6];
328      dir_y=parabuf[7];
329      dir_z=parabuf[8];
330      speed=parabuf[9];
331      smp_time=parabuf[10];
332      mode=parabuf[11];
333
334      move(X_AXIS,dir_x,unit_x,speed);
335      move(Y_AXIS,dir_y,unit_y,speed);
336      move(Z_AXIS,dir_z,unit_z,speed);
337
338      counts[0]=smp_time;
339      newcntmenu(counts);
340
341      newinitcnter();
342      newinitsrl();
343      newsendcntpc(counts);
344    }
345  }
346
347  void move(uchar axis,uchar dir,uint units,uchar spd)
348  {
349    uint  i;
350    uchar speed;
351    uint  period=13;
```

```
352     uchar period_h,period_l;
353
354     if( (dir!=0) && (dir!=1) )
355     {
356       return;
357     }
358
359     if( spd==0 )
360     {
361       speed=1;
362     }
363     else
364     {
365       if( spd > 100 )
366       {
367         speed=100;
368       }
369       else
370       {
371         speed=spd;
372       }
373     }
374
375     period=1000000/((uint)speed*SPEED_PPS);
376     period=65535-period;
377     period+=MOVE_CONST;
378     period_h=(uchar)(period/256);
379     period_l=(uchar)(period%256);
380
381     TR1=0;
382     TR0=0;
383
384     TMOD=0x21;
385
386     switch(axis)
387     {
388       case(0):
389       {
390         DIR_X=dir;
391         for(i=0;i<units*UNIT_PULSE;i++)
392         {
393           CP_X=0;
394           dly10us();
395           dly10us();
396           CP_X=1;
397           TH0=period_h;
398           TL0=period_l;
399           TR0=1;
400           while(!TF0);
401           TF0=0;
402           TR0=0;
```

```
403         }
404
405         break;
406       }
407       case(1):
408       {
409         DIR_Y=dir;
410         for(i=0;i<units*UNIT_PULSE;i++)
411         {
412           CP_Y=0;
413           dly10us();
414           dly10us();
415           CP_Y=1;
416           TH0=period_h;
417           TL0=period_l;
418           TR0=1;
419           while(!TF0);
420           TF0=0;
421           TR0=0;
422         }
423
424         break;
425       }
426       case(2):
427       {
428         DIR_Z=dir;
429         for(i=0;i<units*UNIT_PULSE;i++)
430         {
431           CP_Z=0;
432           dly10us();
433           dly10us();
434           CP_Z=1;
435           TH0=period_h;
436           TL0=period_l;
437           TR0=1;
438           while(!TF0);
439           TF0=0;
440           TR0=0;
441         }
442
443         break;
444       }
445       default:
446         return;
447     }
448       }
```