Model MCS6A

DLL Software Interface

User Manual

Grünwalder Weg 28a, D-82041 Oberhaching
Germany

Version 1.0, 1. July 2011

Model MCS6A
DLL Software Interface Manual
Table of Contents

1. Introduction

The MPANT software for the 5/6-input multiscaler MCS6A consists of a hardware-dependent server program and a general graphics program that controls the hardware via a DLL. Any other Windows application can also control the hardware via the DLL. To support the programming of such customer-specific user interfaces, as an option we deliver this documentation including sourcecode and example programs for LabVIEW, Visual Basic, C and Delphi. The complete sourcecode of the DLL that controls the hardware via the server program is included in the appendix. A special FMPA3.DLL including source code with examples like F*(x-x‘)/(x+x‘) where x‘ is marked by a routing bit allows to display calculated spectra including calculated error bars in the MPANT program.

The server program MCS6A.EXE is a rather compact Windows application. It controls the hardware and data and allows to perform measurements with the system. MPANT is just a user interface to control the server program. It has access to the data in a shared memory region and can display spectra. It is not necessary that MPANT is running during an acquisition. It can be exited and restarted without stopping a running acquisition. If you don't want MPANT automatically started when starting the server, just rename the file MPANT.EXE in the working directory (default C:\MCS6A) for example into MYMPANT.EXE.

The DLL DMCS6.DLL is an interface providing functions to communicate with the server program. Most of these functions send messages to the server as you do it when operating the server program by sending Windows messages via mouse clicks. Please do not expect any functions in this DLL for controlling the hardware directly. All software described in this manual requires that the server program is running. The DLL was mainly developed as an interface between the server program and MPANT, not as a nice developing tool for customers. But by looking at the programming examples and the following hints it should be easy to develop own programs that are able to control the server like MPANT does.

1.1 Some hints

The server program has a built-in command interpreter. The syntax of these commands is described in the MCS6A manual, chapter 5.2, and in the MPANT on-line help (look for: "How to use the command language.."). It is recommended to send commands like "range=16384" to the server via the RunCmd DLL function, i.e. RunCmd(0, "range=16384"); if you want to set parameters like a spectra length. The alternative method is to store all settings parameters into the DLL by calling the DLL function StoreSettingData(setting, 0); and then calling NewSetting(0); to send a message to the server to read new settings from the DLL. This method will not work for changing a spectra length to avoid the problem of any undefined memory pointers. The range parameter should be changed by the server program only (or by sending a "range=.." command). The recommended usage of the DLL is reading parameters like Status, Settings, Strings, Cnt numbers, ROI boundaries using the corresponding DLL functions, but for any actions or setting any parameters the command interpreter should be used.

If your application that controls the MCS6A server via the DLL is a true Windows application with a main window and corresponding message loop handling messages sent to this window, you can fetch a special Windows message to react immediately on actions like an acquisition status change without permanently polling the status:

Declare a global int MM_STATUS and somewhere when initializing your program register a Windows message using a code line like:

```
MM_STATUS=RegisterWindowMessage("MCS6AStatus");
```

furthermore, declare somewhere in your headers constants

```
#define ID_NEWSTATUS   162
#define ID_NEWSETTING  139
#define ID_NEWDATA     160
```

I assume your main Window Procedure is declared like

```
DWORD WINAPI MyMainWndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
```

you can then insert here code like

```
if (msg == MM_STATUS) {
  if (wParam == ID_NEWSTATUS) {
     // status change, read acquisition status and react accordingly...
  }
  else if (wParam == ID_NEWDATA) {
    //  release all pointers, the server will reallocate some spectra..
  }
  else if (wParam == ID_NEWSETTING) {
    // the server has reallocated some spectra, get new pointers..
  }
}
```

On any status change the server sends a NEW_STATUS message. The lParam value is usually zero, but after a stop of an acquisition the lParam is equal to 1, so your program is able to react accordingly.

It is important that the DLL is loaded first by the Server program and that it is loaded from the same path by all programs using it. Otherwise it does not work to access the shared memory. The dmcs6.dll is installed into the Windows\System32 directory. Please make sure that there is nowhere else any file dmcs6.dll. Start MCS6A.exe by hand before starting your program, or by a call from your program for example like

```
{
  STARTUPINFO startupinfo = {0};
  PROCESS_INFORMATION procinfo = {0};
  startupinfo.cb = sizeof(STARTUPINFO);
  return CreateProcess ( "MCS6A.EXE", NULL, NULL, NULL, FALSE, NORMAL_PRIORITY_CLASS,
NULL, NULL, &startupinfo,&procinfo);
}
```

, but before your program loads the DLL. It is recommended not to link the DLL to your program using a dmcs6.LIB file, but explicitely load it at runtime as demonstrated in our example tstmcs6a.c.

2. Using the DMCS6 DLL from LabVIEW

To access the MCS6A data directly from LabView via the DLL, some LabView VI's („Virtual Instruments") contained in MCS6ALV.LLB and the MCS6ATEST.VI are provided.

2.1 Installation

Files: MCS6ALV.LLB, MCS6ATEST.VI

The distribution medium contains in a directory \LV the following files: MCS6ALV.LLB and MCS6ATEST.VI. Copy these files into your working directory of the MPANT software. Please start now MCS6A.EXE and then LabVIEW and open the MCS6ATEST.VI (or just double click on MCS6ATEST.VI). You may load some data with MPANT or the server and then run the VI.

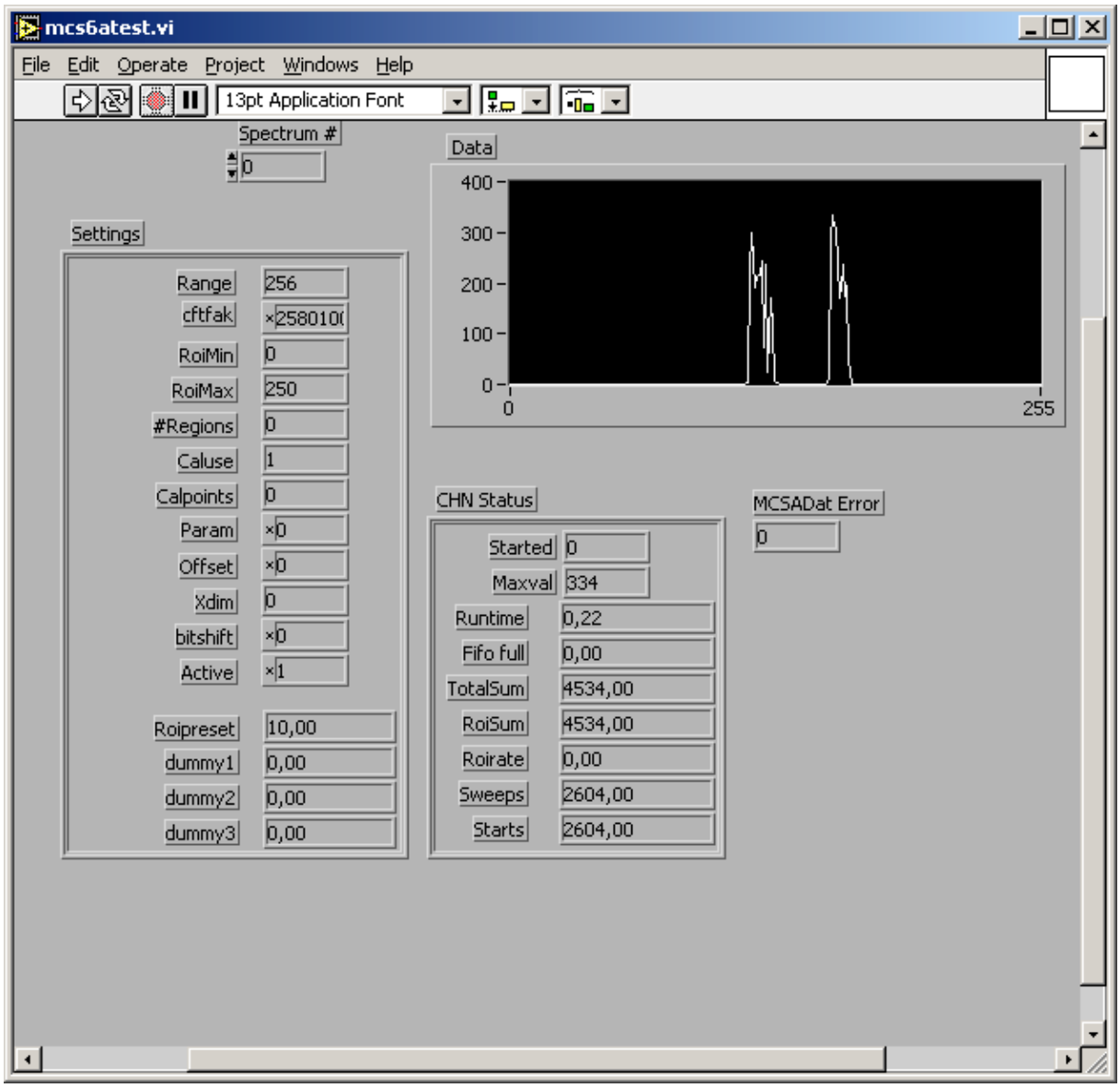


Figure 2.1 The MCS6Atest VI.

In the Windows menu, click on Show Diagram to display the diagram, and on Show Help Window to display the Help window.

The Demo VI contains the VI's to get the settings, status and spectrum data.

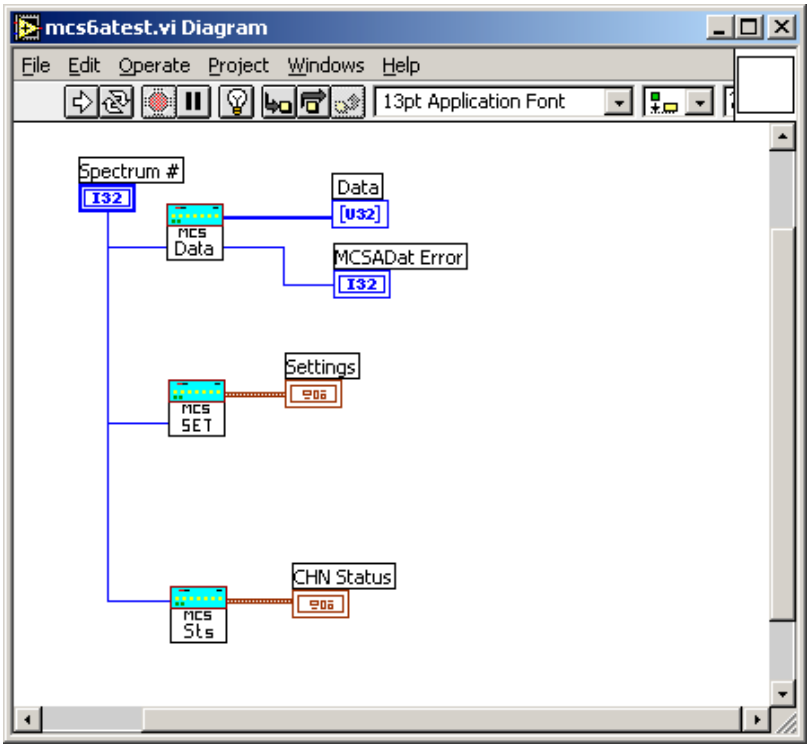
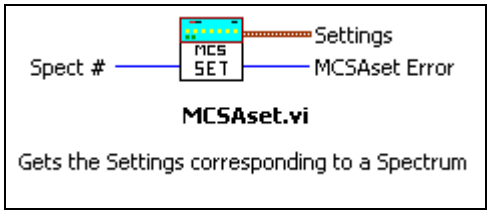


Fig. 2.2: Diagram of MCS6Atest.VI

2.2 Getting Parameters



The Settings are obtained with the MCSAset.vi. It results a 32 bit integer as an error code and the settings contained in a cluster.

You can use the help window to get information: on the front panel as the active window, just move the mouse over the item you are interested and observe the help window. The cluster has the components known from the DLL structure definitions:

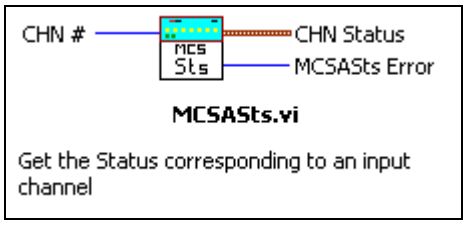
1. Range (I32)	2. Prena (I32)
3. RoiMin (I32)	4. RoiMax (I32)
5. #Regions (I32)	6. Caluse (I32)
7. Calpoints (I32)	8. Param (hex) (I32)
9. Offset (hex) (I32)	10. Xdim (I32)
11. Timesh (I32)	12. Active(hex) (I32)
13. Roipreset (DBL)	14. Ltpreset (DBL)
15. TimeOffs (DBL)	16. Dwelltime (DBL)

For the detailed meaning of each parameter please refer to the LabView on-line help window or the DLL description in this manual.

The error code has the following meanings:

1: DMCS6.DLL not found
2: GetSettingData function in DLL not found
4: No Parameters available

2.3 Getting the Status



The Status corresponding to an input channel is obtained with the MCSASts.VI. The Input is the Channel number with 0 for STOP1 and so on. It results a 32 bit integer as an error code and the status parameters contained in a cluster.

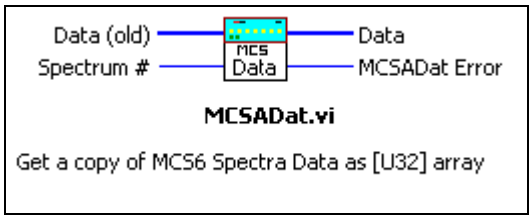
The cluster has the following components:

1: Started (I32):	0 == OFF, 1 == ON, 3 == READ OUT
2: Maxval (U32):	Maximum value in spectra, only available when stopped
3: Runtime (DBL)	in seconds
4: Fifo full (DBL)	counted number of datalost bits, only available for special dataword formats
5: TotalSum (DBL)	
7: RoiSum (DBL)	
8: RoiRate (DBL)	
9: Sweeps (DBL)	
10: Starts (DBL)	

The error code has the following meanings:

1: DMCS6.DLL not found
2: GetStatusData function in DLL not found
4: No Parameters available

2.4 Getting the Spectrum Data

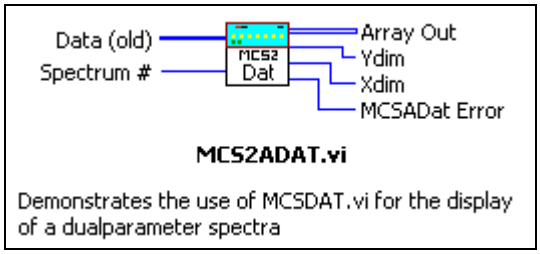


The Spectrum data is obtained with the MCSADat.vi. It results a 32 bit integer as an error code and the spectrum as a [U32] array.

The error code has the following meanings:

1: DMCS6.DLL not found
2: GetSettingData or LVGetDat function in DLL not found
4: No Data available

How to use MCSADat.vi to get a display of a dualparameter spectra in LabVIEW is demonstrated in MCS2ADat.vi. Note that the x- and y-axes are exchanged compared with the display in MPANT.



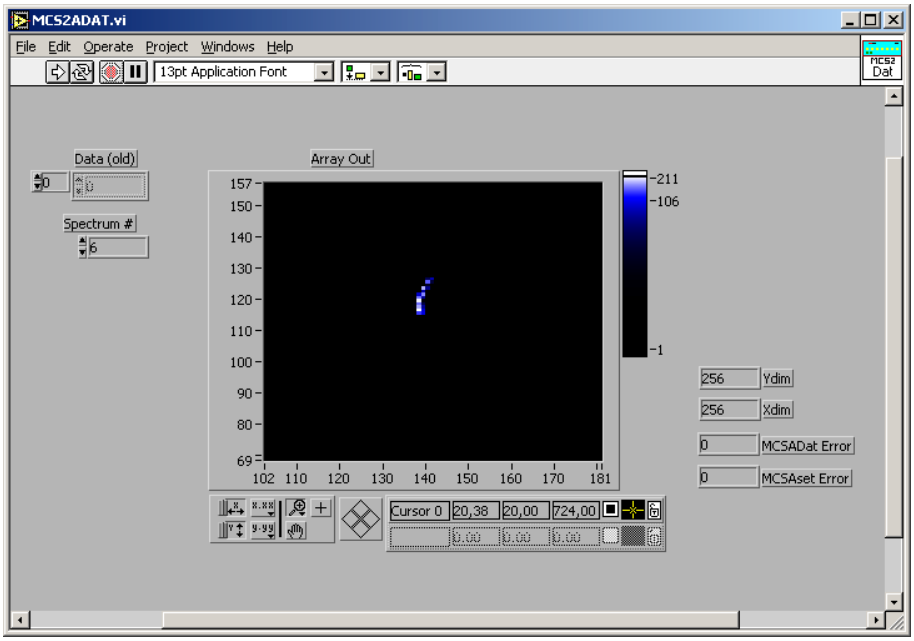
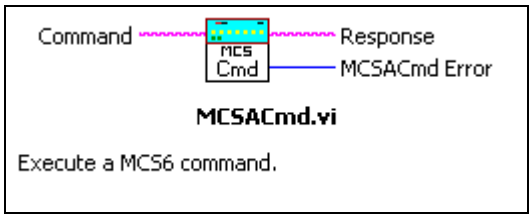


Fig. 2.3: MCS2ADat.VI

2.5 Executing a command

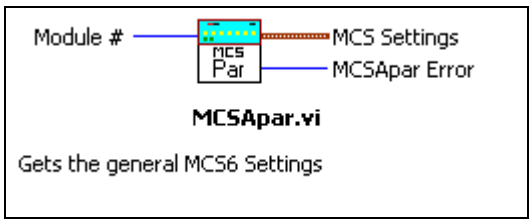


The MCS6ALV.LLB contains some more VIs that are not used by MCS6Atest.VI. Any command for the MCS6A server can be executed by MCSACmd.VI. It results an response string and a 32 bit integer as an error code.

The error code has the following meanings:

1: DMCS6.DLL not found
2: RunCmd function in DLL not found
4: Memory Allocation failure

2.6 Getting the MCS6A general parameters



The general MCS6A settings can be obtained by MCSApar.vi. It results the MCS6A Settings in a cluster and a 32 bit integer as an error code.

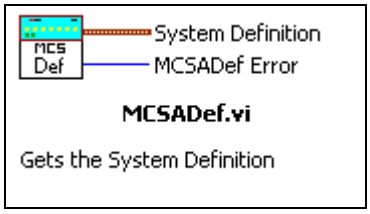
The MCS6A Settings have following components:

1. sweepmode (I32)	2. prena (I32)
3. cycles (I32)	4. sequences (I32)
5. syncout (I32)	6. digio (I32)
7. digval (I32)	8. dac0 (I32)
9. dac1 (I32)	10. dac2 (I32)
11. dac3 (I32	12. dac4 (I32)
13. dac5 (I32)	14. extclk (I32)
15. maxchan (I32)	16. serno (I32)
17. ddruse (I32)	18. active (I32)
19. holdafter (DBL)	20. swpreset (DBL)
21. fstchan (DBL)	22. rtpreset (DBL)

For the detailed meaning of each parameter please refer to the LabView on-line help window or the DLL description in this manual. The error code has the following meanings:

1: DMCS6.DLL not found
2: GetMCSSettingData function in DLL not found
4: No parameters available

2.7 Getting the MCS6A System definition



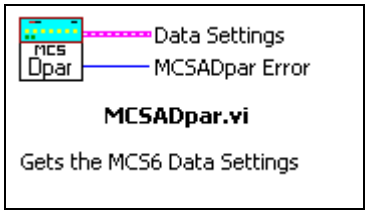
The MCS6A System Definition are obtained by MCSADef.vi. It results the System Definition parameters in a cluster and a 32 bit integer as an error code. The System Definition has following components:

1. nDevices (I32)	nDevices is the number of physical channels (6)
2. nDisplays (I32)	nDisplays is the number of spectra = 6 + calc. spectra
3. nSystems (I32)	nSystems the number of independent systems which is 1 in the present version
4. bRemote (I32)	bRemote indicates whether the MCS6A server is controlled by MPANT
5. sys (I32)	System definition word, 0 (reserved for future applications with more devices)

The error code has the following meanings:

1: DMCS6.DLL not found
2: GetDefData function in DLL not found
4: No parameters available

2.8 Getting the MCS6A Data Settings



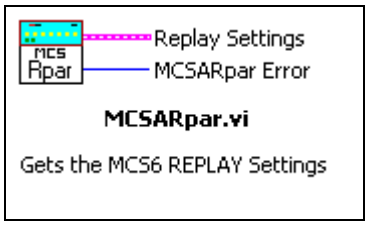
The MCS6A Data Settings can be obtained by MCSADpar.vi. It results the Data Settings in a cluster and a 32 bit integer as an error code. The Data Settings are:

1. savedata (I32): 1 means auto save after stop
2. autoinc (I32): autoincrement MPA data filename
3. fmt (I32): MPA format type (0=ASCII, 1=binary, 2=CSV)
4. sepfmt (I32): format type for seperate spectra
5. sephead (I32) 1 means seperate header
6. smpts (I32) number of points for smoothing operation
7. caluse (I32) 1 means using calibration for shifted spectra summing
8. filename (STR) MPA data file name
9. specfile (STR) spectrum file name
10. command (STR)

The error code has the following meanings:

1: DMCS6.DLL not found
2: GetDatSetting function in DLL not found
4: No parameters available

2.9 Getting the Replay Settings



The Replay Settings can be obtained by MCSARpar.VI. It needs as input the spectrum number and results a 32 bit integer as an error code and the Replay Settings:
1. use (I32): 1 if Replay Mode ON
2. modified (I32): 1 if different settings are used from measurement time
3. limit (I32): 0=all, 1 = limited time range
4. speed (I32): replay speed in units of 100 kByte per sec
5. timefrom (DBL): first time (sec)
6. timeto (DBL): last time (sec)
7. timepreset (DBL): last time - first time
8. filename (STR): listfile for replay

The error code has the following meanings:

1: DMCS6.DLL not found

2: GetReplaySetting function in DLL not found
4: No parameters available

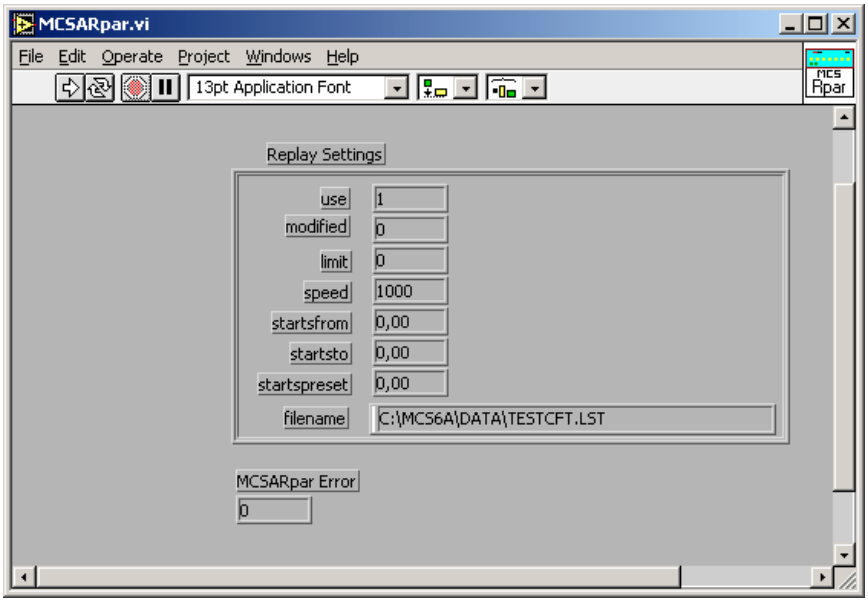
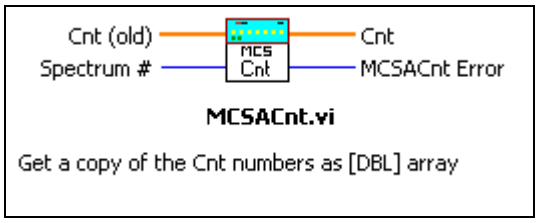


Fig. 2.4: MCSARpar.VI

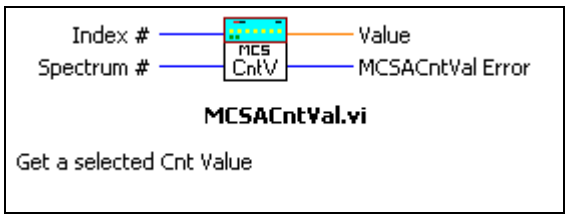
2.10 Getting the Cnt numbers



The Cnt numbers can be obtained by MCSACnt.VI. It needs as input the spectrum number and results a 32 bit integer as an error code and the Cnt numbers as an array of 448 DBL containing the Cnt numbers.

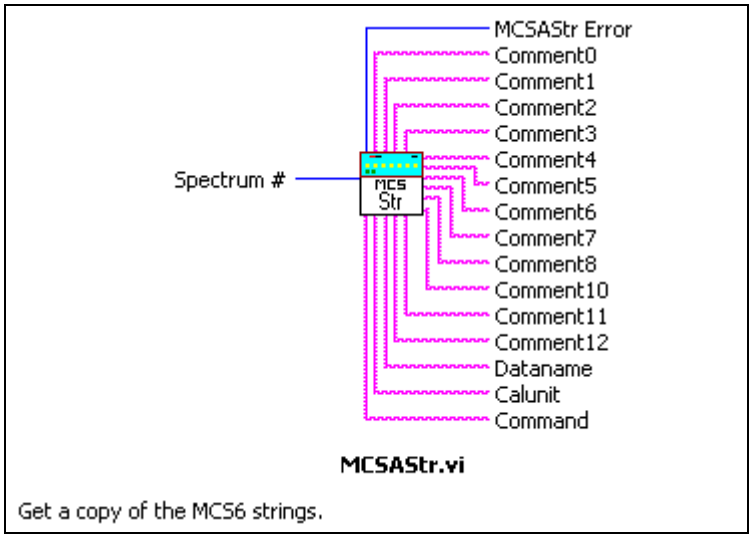
Array of 448 DBL containing the Cnt numbers.
Cnt[0] = Realtime, Cnt[1] = Totalsum,
Cnt[2] = ROIsum, Cnt[3] = ROIrate,
Cnt[4] = Fifofull, Cnt[5] = Sweeps,
Cnt[6] = Starts,
Cnt[11] = c0 cal. coeff., Cnt[12] = c1,
Cnt[13] = c2, Cnt[14] = c3,
Cnt[15] = board.status1:
after conversion to integer: bit5=DATA_LOST
set when fifo full at any time, cleared with HALT.
Cnt[16] = Temperature(FPGA) + Temp(Board)/1000.
Cnt[17] = FIFOCNT, indicates filling of large FIFO
in units of 8 bytes
Cnt[18] = Sum of all events
Cnt[19] = calch0, Cnt[35] = calval0, calib. points
Cnt[20] = calch1, Cnt[36] = calval1,...
Cnt[64]..Cnt[191] : Peak values in
corresponding Roi 0..127 for Calibration
Cnt[192], Cnt[193] ,... Cnt[447]: Roi Sum and Roi Net Sum
in corresponding Roi 0 (, 1, ..127)
(actualized by MPANT when selected)

The error code has the following meanings:
1: DMCS6.DLL not found
2: LVGetCnt function in DLL not found
4: No Data available



A selected Cnt Value can be obtained by MCSACntVal.VI. Inputs are the spectrum number and the index into the Cnt array. It results a 32 bit integer as an error code and the selected Cnt value as DBL.

2.11 Getting the Strings

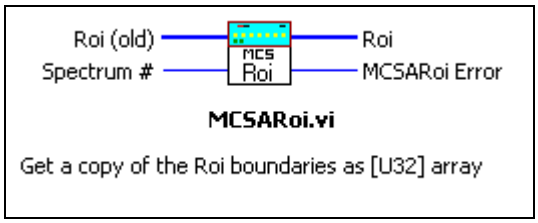


The Strings can be obtained by MCSAStr.VI. It results a 32 bit integer as an error code and the Strings.

The error code has the following meanings:

1: DMCS6.DLL not found
2: LVGetStr function in DLL not found
4: No Data available

2.12 Getting the ROI boundaries



The ROI boundaries can be obtained by MCSARoi.VI. It results a 32 bit integer as an error code and the ROI boundaries for single spectra or

rectangle ROIs for dualparameter spectra, respectively, contained in a [U32] array.

The error code has the following meanings:

1: DMCS6.DLL not found
2: LVGetRoi function in DLL not found
4: No Data available

3. Using the DMCS6 DLL from Visual Basic

In the following an example is shown how to control the MCS6A from a simple program in Visual Basic version 5.0.

3.1 The Include File

The include file DECLMCS6.BAS contains the structure and function definitions of the DLL.

```
Attribute VB_Name = "DECLMCS6"
Type Acqstatus
   Val As Long
   Val1 As Long
   Cnt(0 To 7) As Double
End Type

Type Acqsetting
   Range As Long
   Cftfak As Long
   Roimin As Long
   Roimax As Long
   Nregions As Long
   Caluse As Long
   Calpoints As Long
   Param As Long
   Offset As Long
   Xdim As Long
   Bitshift As Long
   Active As Long
   Roipreset As Double
   Dummy1 As Double
   Dummy2 As Double
   Dummy3 As Double
End Type

Type Replaysetting
   Use As Long
   Modified As Long
   Limit As Long
   Speed As Long
   Startsfrom As Double
   Startsto As Double
   Startspreset As Double
   Filename As String * 256
End Type

Type Datsetting
   SaveData As Long
   Autoinc As Long
   Fmt As Long
   Mpafmt As Long
   Sephead As Long
   Smpts As Long
   Caluse As Long
   Filename As String * 256
   Specfile As String * 256
   Command As String * 256
End Type
```

```
Type Boardsetting
   Sweepmode As Long
   Prena As Long
   Cycles As Long
   Sequences As Long
   Syncout As Long
   Digio As Long
   Digval As Long
   Dac0 As Long
   Dac1 As Long
   Dac2 As Long
   Dac3 As Long
   Dac4 As Long
   Dac5 As Long
   Fdac As Long
   Tagbits As Long
   Extclk As Long
   Maxchan As Long
   Serno As Long
   Ddruse As Long
   Active As Long
   Holdafter As Double
   Swpreset As Double
   Fstchan As Double
   Timepreset As Double
End Type

Type Acqdef
  Ndevices As Long
  Ndisplays As Long
  Nsystems As Long
  Bremote As Long
  Sys As Long
  Sys0(16) As Long
  Sys1(16) As Long
End Type

Declare Sub StoreSettingData Lib "DMCS6.DLL" Alias "#2" (Setting As Acqsetting, ByVal Ndisplay As Long)
Declare Function GetSettingData Lib "DMCS6.DLL" Alias "#3" (Setting As Acqsetting, ByVal Ndisplay As
Long) As Long
Declare Function GetStatusData Lib "DMCS6.DLL" Alias "#5" (Status As Acqstatus, ByVal Ndevice As Long)
As Long
Declare Sub Start Lib "DMCS6.DLL" Alias "#6" (ByVal Nsystem As Long)
Declare Sub Halt Lib "DMCS6.DLL" Alias "#7" (ByVal Nsystem As Long)
Declare Sub Continue Lib "DMCS6.DLL" Alias "#8" (ByVal Nsystem As Long)
Declare Sub NewSetting Lib "DMCS6.DLL" Alias "#9" (ByVal Ndisplay As Long)
Declare Function ServExec Lib "DMCS6.DLL" Alias "#10" (ByVal Clwnd As Long) As Long
Declare Function GetSpec Lib "DMCS6.DLL" Alias "#13" (ByVal I As Long, ByVal Ndisplay As Long) As
Long
Declare Sub SaveSetting Lib "DMCS6.DLL" Alias "#14" ()
Declare Function GetStatus Lib "DMCS6.DLL" Alias "#15" (ByVal Ndevice As Long) As Long
Declare Sub EraseData Lib "DMCS6.DLL" Alias "#16" (ByVal Nsystem As Long)
Declare Sub SaveData Lib "DMCS6.DLL" Alias "#17" (ByVal Ndevice As Long, ByVal All As Long)
Declare Sub GetBlock Lib "DMCS6.DLL" Alias "#18" (Hist As Long, ByVal Start As Long, ByVal Size As
Long, ByVal Stp As Long, ByVal Ndisplay As Long)
Declare Function GetDefData Lib "DMCS6.DLL" Alias "#20" (Def As Acqdef) As Long
Declare Sub LoadData Lib "DMCS6.DLL" Alias "#21" (ByVal Ndevice As Long, ByVal All As Long)
Declare Sub NewData Lib "DMCS6.DLL" Alias "#22" ()
Declare Sub HardwareDlg Lib "DMCS6.DLL" Alias "#23" (ByVal Item As Long)
Declare Sub UnregisterClient Lib "DMCS6.DLL" Alias "#24" ()
Declare Sub DestroyClient Lib "DMCS6.DLL" Alias "#25" ()
Declare Sub RunCmd Lib "DMCS6.DLL" Alias "#28" (ByVal Ndevice As Long, ByVal Cmd As String)
```

```
Declare Sub AddData Lib "DMCS6.DLL" Alias "#29" (ByVal Ndisplay As Long, ByVal All As Long)
Declare Function LVGetRoi Lib "DMCS6.DLL" Alias "#30" (Roi As Long, ByVal Ndisplay As Long) As Long
Declare Function LVGetCnt Lib "DMCS6.DLL" Alias "#31" (Cnt As Double, ByVal Ndisplay As Long) As
Long
Declare Function LVGetOneCnt Lib "DMCS6.DLL" Alias "#32" (Cnt As Double, ByVal Ndisplay As Long,
ByVal Cntnum As Long) As Long
Declare Function LVGetStr Lib "DMCS6.DLL" Alias "#33" (ByVal Comment As String, ByVal Ndisplay As
Long) As Long
Declare Sub SubData Lib "DMCS6.DLL" Alias "#34" (ByVal Ndisplay As Long, ByVal All As Long)
Declare Sub Smooth Lib "DMCS6.DLL" Alias "#35" (ByVal Ndisplay As Long)
Declare Function GetMCSSetting Lib "DMCS6.DLL" Alias "#39" (Msetting As Boardsetting) As Long
Declare Function GetDatSetting Lib "DMCS6.DLL" Alias "#41" (Dsetting As Datsetting) As Long
Declare Function GetReplaySetting Lib "DMCS6.DLL" Alias "#43" (Rsetting As Replaysetting) As Long
```

3.2 The Visual Basic demo program

The simple Visual Basic program is shown here: It allows to get Status, Settings, spectrum data and strings for any MCS6 spectra, perform actions like start, halt, continue, erase, or any control command. To use the example, it is recommended to copy the DMCS6.DLL into your WINNT\SYSTEM32 directory and delete or rename it in your working directory. It is essential that the DLL is loaded from Visual Basic and the MCS6 server program from the same path.

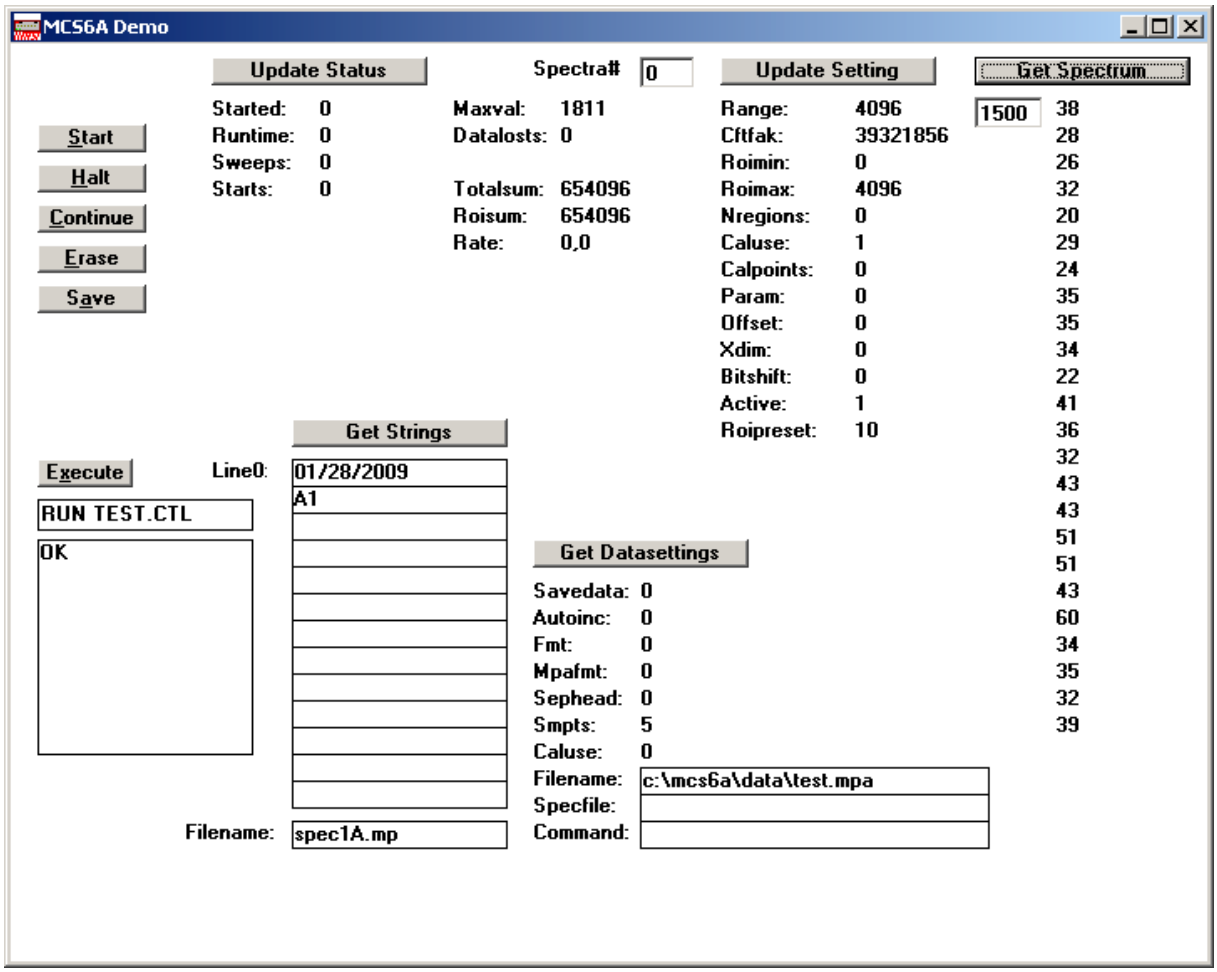


Fig. 3.1: The Visual Basic MCS6A Demo program

This is the complete program code beside form data:

```
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim Status As Acqstatus
Dim Setting As Acqsetting
Dim Dsetting As Datsetting
Dim OldStarted As Integer
Dim Mcano As Long
Dim Sysno As Long
Dim Chan As Long
Dim Hist(24) As Long
Dim Toggle As Integer
```

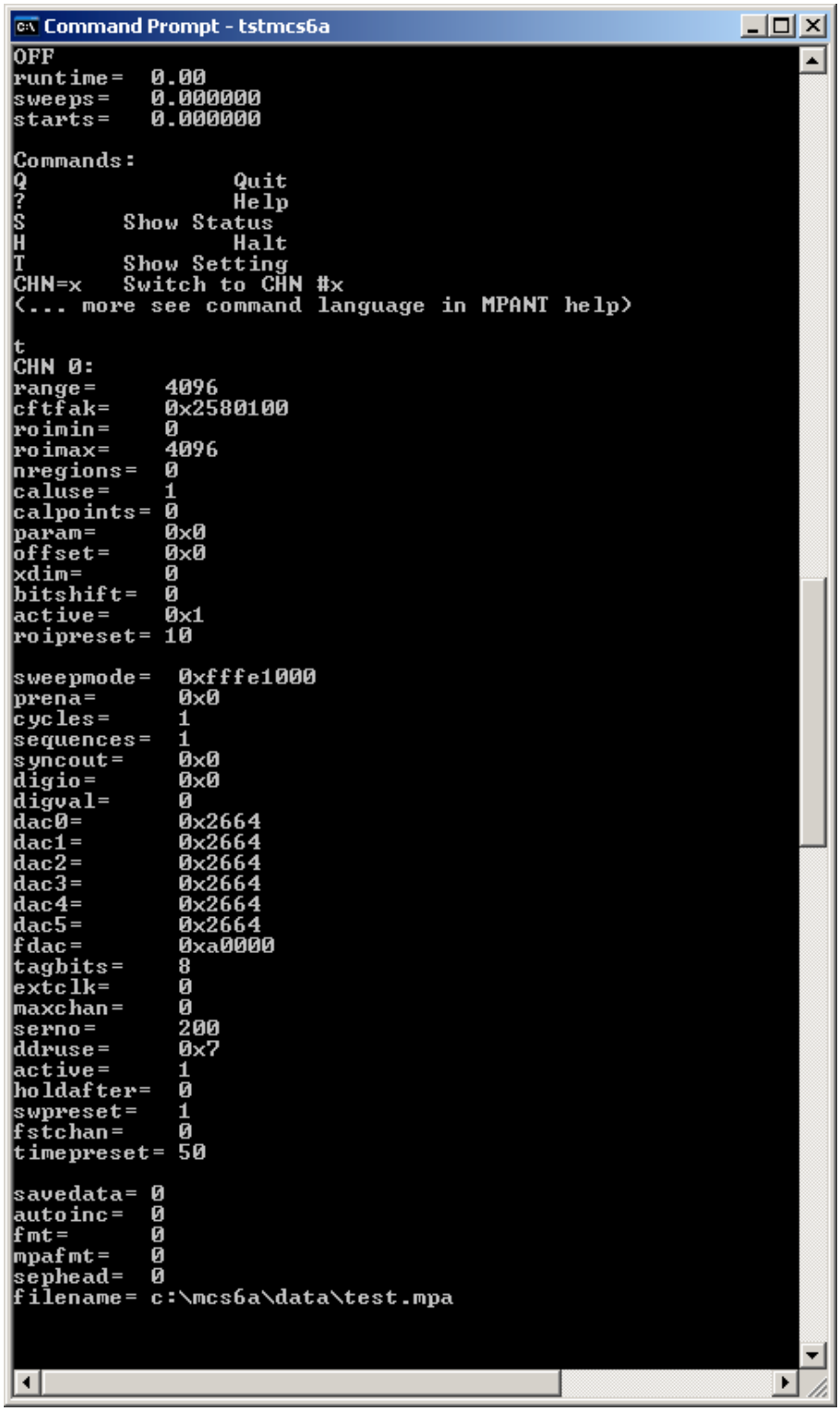
```
Private Sub CommandContinue_Click()
  Call Continue(0)
End Sub

Private Sub CommandDatasettings_Click()
  Call GetDatSetting(Dsetting)
  LabelSavedata.Caption = Dsetting.SaveData
  LabelAutoinc.Caption = Dsetting.Autoinc
  LabelFmt.Caption = Dsetting.Fmt
  LabelMpafmt.Caption = Dsetting.Mpafmt
  LabelSephead.Caption = Dsetting.Sephead
  LabelSmpts.Caption = Dsetting.Smpts
  LabelAddcal.Caption = Dsetting.Caluse
  LabelMpafilename.Caption = Dsetting.Filename
  Labelspecfile.Caption = Dsetting.Specfile
  LabelCommand.Caption = Dsetting.Command
End Sub

Private Sub CommandErase_Click()
  Call EraseData(0)
End Sub

Private Sub CommandExecute_Click()
  Dim a As String * 1024
  Mid$(a, 1) = TextCommand.Text
  Call RunCmd(0, a)
  LabelRespons.Caption = a
  Mcano = Val(TextMC.Text)
  Ret = GetStatus(Mcano)
  Ret = GetStatusData(Status, 0)
  Call UpdateMpStatus
  Ret = GetStatusData(Status, Mcano)
  Call UpdateStatus
End Sub

Private Sub CommandGetspec_Click()
  Mcano = Val(TextMC.Text)
  Chan = Val(TextChan.Text)
  Call GetBlock(Hist(0), Chan, Chan + 24, 1, Mcano)
  For I = 0 To 23 Step 1
    LabelData(I).Caption = Hist(I)
  Next I
End Sub

Private Sub CommandGetstring_Click()
  Dim b As String * 1024
  Mcano = Val(TextMC.Text)
  Ret = LVGetStr(b, Mcano)
  LabelLine(0).Caption = Mid$(b, 1, 60)
  LabelLine(1).Caption = Mid$(b, 61, 60)
  LabelLine(2).Caption = Mid$(b, 121, 60)
  LabelLine(3).Caption = Mid$(b, 181, 60)
  LabelLine(4).Caption = Mid$(b, 241, 60)
  LabelLine(5).Caption = Mid$(b, 301, 60)
  LabelLine(6).Caption = Mid$(b, 361, 60)
  LabelLine(7).Caption = Mid$(b, 421, 60)
  LabelLine(8).Caption = Mid$(b, 481, 60)
  LabelLine(9).Caption = Mid$(b, 541, 60)
  LabelLine(10).Caption = Mid$(b, 601, 60)
  LabelLine(11).Caption = Mid$(b, 881, 60)
```

```
  LabelLine(12).Caption = Mid$(b, 961, 60)
  LabelLine(13).Caption = Mid$(b, 661, 100)
End Sub

Private Sub CommandHalt_Click()
  Call Halt(0)
End Sub


Private Sub CommandSave_Click()
  Call SaveData(0, 1)
End Sub

Private Sub CommandSetting_Click()
  Mcano = Val(TextMC.Text)
  If GetSettingData(Setting, Mcano) = 1 Then
     Call UpdateSetting
  End If
End Sub

Private Sub CommandStart_Click()
  Call Start(0)
End Sub

Private Sub CommandUpdate_Click()
  Mcano = Val(TextMC.Text)
  Ret = GetStatus(Mcano)
  If GetStatusData(Status, Mcano) = 1 Then
     Call UpdateStatus
  End If
End Sub

Private Sub Form_Load()
  OldStarted = 0
  Mcano = 0
  Sysno = 0
  Chan = 0
  Ret = ServExec(0)
  Ret = GetStatus(0)
  Ret = GetStatusData(Status, 0)
  Call UpdateMpStatus
  Ret = GetStatusData(Status, Mcano)
  Call UpdateStatus
  Ret = GetSettingData(Setting, 0)
  Call UpdateSetting
End Sub



Private Sub Timer1_Timer()
  Mcano = Val(TextMC.Text)
  If Mcano > 16 Then
    Mcano = 16
  End If
  If Mcano < 0 Then
    Mcano = 0
  End If
  Ret = GetStatus(0)
  If GetStatusData(Status, 0) = 1 Then
    If Status.Val > 0 Or OldStarted > 0 Then
      Toggle = Not Toggle
```

```
      OldStarted = Status.Val
      Call UpdateMpStatus
      If GetStatusData(Status, Mcano) = 1 Then
        Call UpdateStatus
      End If
    End If
  End If
End Sub

Private Sub UpdateMpStatus()
      LabelStarted.Caption = Status.Val
      LabelRuntime.Caption = Status.Cnt(0)
      LabelSweeps.Caption = Status.Cnt(5)
      LabelStarts.Caption = Status.Cnt(6)
End Sub

Private Sub UpdateStatus()
      LabelMaxval.Caption = Status.Val1
      LabelOfls.Caption = Status.Cnt(1)
      LabelTotalsum.Caption = Status.Cnt(2)
      LabelRoisum.Caption = Status.Cnt(3)
      LabelTotalrate.Caption = Format$(Status.Cnt(4), "######0.0#")
End Sub

Private Sub UpdateSetting()
      LabelRange.Caption = Setting.Range
      LabelCftfak.Caption = Setting.Cftfak
      LabelRoimin.Caption = Setting.Roimin
      LabelRoimax.Caption = Setting.Roimax
      LabelNregions.Caption = Setting.Nregions
      LabelCaluse.Caption = Setting.Caluse
      LabelCalpoints.Caption = Setting.Calpoints
      LabelParam.Caption = Setting.Param
      LabelOffset.Caption = Setting.Offset
      LabelXdim.Caption = Setting.Xdim
      LabelBitshift.Caption = Setting.Bitshift
      LabelActive.Caption = Setting.Active
      LabelRoipreset.Caption = Setting.Roipreset
End Sub
```

4. Using the DMCS6 DLL from C

In the following an example is shown how to control the MCS6A from a simple console application written in Microsoft C.



```
Command Prompt - tstmcs6a
OFF
runtime=  0.00
sweeps=   0.000000
starts=   0.000000

Commands:
Q                Quit
?                Help
S        Show Status
H                Halt
T        Show Setting
CHN=x    Switch to CHN #x
<... more see command language in MPANT help>

t
CHN 0:
range=      4096
cftfak=     0x2580100
roimin=     0
roimax=     4096
nregions=   0
caluse=     1
calpoints=  0
param=      0x0
offset=     0x0
xdim=       0
bitshift=   0
active=     0x1
roipreset=  10

sweepmode=  0xfffe1000
prena=      0x0
cycles=     1
sequences=  1
syncout=    0x0
digio=      0x0
digval=     0
dac0=       0x2664
dac1=       0x2664
dac2=       0x2664
dac3=       0x2664
dac4=       0x2664
dac5=       0x2664
fdac=       0xa0000
tagbits=    8
extclk=     0
maxchan=    0
serno=      200
ddruse=     0x7
active=     1
holdafter=  0
swpreset=   1
fstchan=    0
timepreset= 50

savedata= 0
autoinc=  0
fmt=      0
mpafmt=   0
sephead=  0
filename= c:\mcs6a\data\test.mpa
```

Fig. 4.1: The MCS6A Demo program written in C

4.1 The Include File

The include file DMCS6.h contains the function definitions of the DLL. It includes also the structure definitions from struct.h listed in the appendix A.1.

```
#ifdef __cplusplus
extern "C"
{
#endif

#include "struct.h"
#define MAXCNT          448
#define MAXDSP 30
#define MAXDEV          2


#define ID_SAVE         103
#define ID_CONTINUE     106
#define ID_START        109
#define ID_BREAK        137
#define ID_NEWSETTING   139
#define ID_GETSTATUS    141
#define ID_SAVEFILE     151
#define ID_ERASE        154
#define ID_LOADFILE     155
#define ID_NEWDATA      160
#define ID_HARDWDLG     161
#define ID_SAVEFILE2    194
#define ID_LOADFILE2    203
#define ID_SAVEFILE3    217
#define ID_LOADFILE3    219
#define ID_SAVEFILE4    223
#define ID_LOADFILE4    225
#define ID_LOADFILE5    226
#define ID_LOADFILE6    227
#define ID_LOADFILE7    228
#define ID_LOADFILE8    229
#define ID_SAVEFILE5    230
#define ID_SAVEFILE6    231
#define ID_SAVEFILE7    232
#define ID_SAVEFILE8    233
#define ID_SUMFILE      234
#define ID_SUMFILE2     235
#define ID_SUMFILE3     236
#define ID_SUMFILE4     237
#define ID_SUMFILE5     238
#define ID_SUMFILE6     239
#define ID_SUMFILE7     240
#define ID_SUMFILE8     241
#define ID_SUBTRACT     289
#define ID_SMOOTH       290
#define ID_SUBTRACT2    296
#define ID_SMOOTH2      297
#define ID_SUBTRACT3    298
#define ID_SMOOTH3      299
#define ID_SUBTRACT4    300
#define ID_SMOOTH4      301
#define ID_SUBTRACT5    302
#define ID_SMOOTH5      303
#define ID_SUBTRACT6    304
#define ID_SMOOTH6      305
```

```
#define ID_SUBTRACT7        306
#define ID_SMOOTH7          307
#define ID_SUBTRACT8        308
#define ID_SMOOTH8          309
#define ID_SAVEFILE9        310
#define ID_SAVEFILE10       311
#define ID_SAVEFILE11       312
#define ID_SAVEFILE12       313
#define ID_SAVEFILE13       314
#define ID_SAVEFILE14       315
#define ID_SAVEFILE15       316
#define ID_SAVEFILE16       317
#define ID_LOADFILE9        318
#define ID_LOADFILE10       319
#define ID_LOADFILE11       320
#define ID_LOADFILE12       321
#define ID_LOADFILE13       322
#define ID_LOADFILE14       323
#define ID_LOADFILE15       324
#define ID_LOADFILE16       325
#define ID_SUMFILE9         326
#define ID_SUMFILE10        327
#define ID_SUMFILE11        328
#define ID_SUMFILE12        329
#define ID_SUMFILE13        330
#define ID_SUMFILE14        331
#define ID_SUMFILE15        332
#define ID_SUMFILE16        333
#define ID_SUBTRACT9        334
#define ID_SUBTRACT10       335
#define ID_SUBTRACT11       336
#define ID_SUBTRACT12       337
#define ID_SUBTRACT13       338
#define ID_SUBTRACT14       339
#define ID_SUBTRACT15       340
#define ID_SUBTRACT16       341
#define ID_COMBDLG          401
#define ID_DATADLG          402
#define ID_MAPLSTDLG        403
#define ID_REPLDLG          404
#define ID_ERASE2           1108
#define ID_ERASE3           1109
#define ID_ERASE4           1110
#define ID_ERASEFILE2       1111
#define ID_ERASEFILE3       1112
#define ID_ERASEFILE4       1113
#define ID_START2           1114
#define ID_BREAK2           1115
#define ID_CONTINUE2        1116
#define ID_START3           1117
#define ID_BREAK3           1118
#define ID_CONTINUE3        1119
#define ID_START4           1120
#define ID_BREAK4           1121
#define ID_CONTINUE4        1122
#define ID_RUNCMD           1123
#define ID_RUNCMD2          1124
#define ID_RUNCMD3          1125
#define ID_RUNCMD4          1126
#define ID_RUNCMD5          1127
#define ID_RUNCMD6          1128
#define ID_RUNCMD7          1129
```

```
#define ID_RUNCMD8              1130
#define ID_ERASEFILE5           1131
#define ID_ERASEFILE6           1132
#define ID_ERASEFILE7           1133
#define ID_ERASEFILE8           1134

/*** FUNCTION PROTOTYPES (do not change) ***/
#ifdef DLL
BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called, LPVOID lpReserved);

VOID APIENTRY StoreSettingData(ACQSETTING FAR *Setting, int nDisplay);
                        // Stores Settings into the DLL
int APIENTRY GetSettingData(ACQSETTING FAR *Setting, int nDisplay);
                        // Get Settings stored in the DLL
VOID APIENTRY StoreExtSettingData(EXTACQSETTING FAR *Setting, int nDisplay);
                      // Stores extended Settings into the DLL
int APIENTRY GetExtSettingData(EXTACQSETTING FAR *Setting, int nDisplay);
                      // Get extended Settings stored in the DLL
VOID APIENTRY StoreStatusData(ACQSTATUS FAR *Status, int nDisplay);
                        // Store the Status into the DLL
int APIENTRY GetStatusData(ACQSTATUS FAR *Status, int nDisplay);
                        // Get the Status
VOID APIENTRY Start(int nSystem);       // Start
VOID APIENTRY Halt(int nSystem);        // Halt
VOID APIENTRY Continue(int nSystem);    // Continue
VOID APIENTRY NewSetting(int nDevice);  // Indicate new Settings to Server
UINT APIENTRY ServExec(HWND ClientWnd); // Execute the Server
VOID APIENTRY StoreData(ACQDATA FAR *Data, int nDisplay);
                        // Stores Data pointers into the DLL
int APIENTRY GetData(ACQDATA FAR *Data, int nDisplay);
                        // Get Data pointers
long APIENTRY GetSpec(long i, int nDisplay);
                        // Get a spectrum value
VOID APIENTRY SaveSetting(void);        // Save Settings
int APIENTRY GetStatus(int nDevice);    // Request actual Status from Server
VOID APIENTRY Erase(int nSystem);       // Erase spectrum
VOID APIENTRY SaveData(int nDisplay, int all);    // Saves data
VOID APIENTRY GetBlock(long FAR *hist, int start, int end, int step,
  int nDisplay);                // Get a block of spectrum data
VOID APIENTRY StoreDefData(ACQDEF FAR *Def);
                        // Store System Definition into DLL
int APIENTRY GetDefData(ACQDEF FAR *Def);
                        // Get System Definition
VOID APIENTRY LoadData(int nDisplay, int all);    // Loads data
VOID APIENTRY AddData(int nDisplay, int all);     // Adds data
VOID APIENTRY SubData(int nDisplay, int all);     // Subtracts data
VOID APIENTRY Smooth(int nDisplay);     // Smooth data
VOID APIENTRY NewData(void);            // Indicate new ROI or string Data
VOID APIENTRY HardwareDlg(int item);    // Calls the Settings dialog box
VOID APIENTRY UnregisterClient(void);   // Clears remote mode from MCDWIN
VOID APIENTRY DestroyClient(void);      // Close MCDWIN
UINT APIENTRY ClientExec(HWND ServerWnd);
                        // Execute the Client MCDWIN.EXE
int APIENTRY LVGetDat(unsigned long HUGE *datp, int nDisplay);
                        // Copies the spectrum to an array
VOID APIENTRY RunCmd(int nDisplay, LPSTR Cmd);
                        // Executes command
int APIENTRY LVGetRoi(unsigned long FAR *roip, int nDisplay);
                      // Copies the ROI boundaries to an array
int APIENTRY LVGetOneRoi(int nDisplay, int roinum, long *x1, long *x2);
                                                        // Get one ROI boundary
int APIENTRY LVGetCnt(double far *cntp, int nDisplay);
```

```
                                // Copies Cnt numbers to an array
int APIENTRY LVGetStr(char far *strp, int nDisplay);
                                // Copies strings to an array
VOID APIENTRY StoreMCSSetting(BOARDSETTING *Defmc, int ndev);
                            // Store BOARDSETTING Definition into DLL
int APIENTRY GetMCSSetting(BOARDSETTING *Defmc, int ndev);
                            // Get BOARDSETTING Definition from DLL
VOID APIENTRY StoreDatSetting(DATSETTING *Defdat);
                            // Store Data Format Definition into DLL
int APIENTRY GetDatSetting(DATSETTING *Defdat);
                            // Get Data Format Definition from DLL
VOID APIENTRY StoreReplaySetting(REPLAYSETTING *Repldat);
                            // Store Replay Settings into DLL
int APIENTRY GetReplaySetting(REPLAYSETTING *Repldat);
                            // Get Replay Settings from DLL
int APIENTRY GetDatInfo(int nDisplay, long *xmax, long *ymax);
                            // returns spectra length;
int APIENTRY GetDatPtr(int nDisplay, long *xmax, long *ymax, unsigned long * *pt);
                            // Get a temporary pointer to spectra data
int APIENTRY ReleaseDatPtr(void);
                                // Release temporary data pointer
long APIENTRY GetSVal(int DspID, long xval);
                                // Get special display data like projections or slices from MPANT
int APIENTRY BytearrayToShortarray(short *Shortarray, char *Bytearray, int length);
                                // auxiliary function for VB.NET to convert strings
int APIENTRY LedBlink(int nDev);
                                // Lets the front leds blink for a while
int APIENTRY DigInOut(int value, int enable);
                                // controls Dig I/0 ,
                                // returns digin

#else
typedef int (WINAPI *IMPAGETSETTING) (ACQSETTING FAR *Setting, int nDisplay);
                                // Get Spectra Settings stored in the DLL
typedef int (WINAPI *IMPAGETSTATUS) (ACQSTATUS FAR *Status, int nDisplay);
                                // Get the Status
typedef VOID (WINAPI *IMPARUNCMD) (int nDisplay, LPSTR Cmd);
                                // Executes command
typedef int (WINAPI *IMPAGETCNT) (double FAR *cntp, int nDisplay);
                                // Copies Cnt numbers to an array
typedef int (WINAPI *IMPAGETROI) (unsigned long FAR *roip, int nDisplay);
                                // Copies the ROI boundaries to an array
typedef int (WINAPI *IMPAGETDEF) (ACQDEF FAR *Def);
                                // Get System Definition
typedef int (WINAPI *IMPAGETDAT) (unsigned long HUGE *datp, int nDisplay);
                                // Copies the spectrum to an array
typedef int (WINAPI *IMPAGETSTR) (char FAR *strp, int nDisplay);
                                // Copies strings to an array
typedef UINT (WINAPI *IMPASERVEXEC) (HWND ClientWnd);  // Register client at server MCS6.EXE

typedef int (WINAPI *IMPANEWSTATUS) (int nDev);     // Request actual Status from Server

typedef int (WINAPI *IMPAGETMCSSET) (BOARDSETTING *Board, int nDevice);
                            // Get MCSSettings from DLL
typedef int (WINAPI *IMPAGETDATSET) (DATSETTING *Defdat);
                            // Get Data Format Definition from DLL
typedef int (WINAPI *IMPADIGINOUT) (int value, int enable);  // controls Dig I/0 ,
                                                                // returns digin
typedef int (WINAPI *IMPADACOUT) (int value);   // output Dac value as analogue voltage
typedef VOID (WINAPI *IMPASTART) (int nSystem);        // Start
typedef VOID (WINAPI *IMPAHALT) (int nSystem);         // Halt
typedef VOID (WINAPI *IMPACONTINUE) (int nSystem);     // Continue
```

```
typedef VOID (WINAPI *IMPAERASE) (int nSystem);        // Erase spectrum
#endif

#ifdef __cplusplus
}
#endif
```

4.2 The C demo program

The source of the simple C program is shown here: It shows how to access the DLL and to get Status, Settings and spectrum data. To perform actions like start, halt, continue, erase, just send the corresponding commands using the command language.

```
// ---------------------------------------------------------------------------
// TSTMCS6A.C : DMCS6.DLL Software driver C example
// ---------------------------------------------------------------------------

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <time.h>

#undef DLL
#include "DMCS6.h"


HANDLE         hDLL = 0;

IMPAGETSETTING  lpSet=NULL;
IMPANEWSTATUS   lpNewStat=NULL;
IMPAGETSTATUS   lpStat=NULL;
IMPARUNCMD      lpRun=NULL;
IMPAGETCNT      lpCnt=NULL;
IMPAGETROI      lpRoi=NULL;
IMPAGETDAT      lpDat=NULL;
IMPAGETSTR      lpStr=NULL;
IMPASERVEXEC    lpServ=NULL;
IMPAGETDATSET   lpGetDatSet=NULL;
IMPAGETMCSSET   lpGetMCSSet=NULL;
IMPADIGINOUT    lpDigInOut=NULL;
IMPADACOUT      lpDacOut=NULL;
IMPASTART               lpStart=NULL;
IMPAHALT                lpHalt=NULL;
IMPACONTINUE    lpContinue=NULL;
IMPAERASE               lpErase=NULL;


ACQSETTING     Setting={0};
ACQDATA        Data={0};
ACQDEF         Def={0};
ACQSTATUS      Status={0};
DATSETTING     DatSetting={0};
BOARDSETTING   MCSSetting={0};

short nDev=0;

void help()
{
```

```
            printf("Commands:\n");
            printf("Q                    Quit\n");
            printf("?            Help\n");
            printf("S      Show Status\n");
            printf("H                    Halt\n");
            printf("T      Show Setting\n");
            printf("CHN=x   Switch to CHN #x \n");
    printf("(... more see command language in MPANT help)\n");
    printf("\n");
}

void PrintMpaStatus(ACQSTATUS *Stat)
{
  if(Stat->started == 1) printf("ON\n");
  else if(Stat->started == 3) printf("READ OUT\n");
  else printf("OFF\n");
  printf("runtime=  %.2lf\n", Stat->cnt[ST_RUNTIME]);
  printf("sweeps=   %lf\n", Stat->cnt[ST_SWEEPS]);
  printf("starts=   %lf\n\n", Stat->cnt[ST_STARTS]);
}

void PrintStatus(ACQSTATUS *Stat)
{
  printf("totalsum= %lf\n", Stat->cnt[ST_TOTALSUM]);
  printf("roisum=   %lf\n", Stat->cnt[ST_ROISUM]);
  printf("rate=     %.2lf\n", Stat->cnt[ST_ROIRATE]);
  printf("ofls=     %.2lf\n\n", Stat->cnt[ST_OFLS]);
}

void PrintDatSetting(DATSETTING *Set)
{
  printf("savedata= %d\n", Set->savedata);
  printf("autoinc=  %d\n", Set->autoinc);
  printf("fmt=      %d\n", Set->fmt);
  printf("mpafmt=   %d\n", Set->mpafmt);
  printf("sephead=  %d\n", Set->sephead);
  printf("filename= %s\n\n", Set->filename);
}

void PrintMCSSetting(BOARDSETTING *Set)
{
  printf("sweepmode=  0x%x\n", Set->sweepmode);
  printf("prena=      0x%x\n", Set->prena);
  printf("cycles=     %d\n", Set->cycles);
  printf("sequences=  %d\n", Set->sequences);
  printf("syncout=    0x%x\n", Set->syncout);
  printf("digio=      0x%x\n", Set->digio);
  printf("digval=     %d\n", Set->digval);
  printf("dac0=       0x%x\n", Set->dac0);
  printf("dac1=       0x%x\n", Set->dac1);
  printf("dac2=       0x%x\n", Set->dac2);
  printf("dac3=       0x%x\n", Set->dac3);
  printf("dac4=       0x%x\n", Set->dac4);
  printf("dac5=       0x%x\n", Set->dac5);
  printf("fdac=       0x%x\n", Set->fdac);
  printf("tagbits=    %d\n", Set->tagbits);
  printf("extclk=     %d\n", Set->extclk);
  printf("maxchan=    %d\n", Set->maxchan);
  printf("serno=      %d\n", Set->serno);
  printf("ddruse=     0x%x\n", Set->ddruse);
  printf("active=     %d\n", Set->active);
  printf("holdafter=  %lg\n", Set->holdafter);
```

```
  printf("swpreset=   %lg\n", Set->swpreset);
  printf("fstchan=    %lg\n", Set->fstchan);
  printf("timepreset= %lg\n\n", Set->timepreset);
}

void PrintSetting(ACQSETTING *Set)
{
  printf("range=     %ld\n", Set->range);
  printf("cftfak=    0x%x\n", Set->cftfak);
  printf("roimin=    %ld\n", Set->roimin);
  printf("roimax=    %ld\n", Set->roimax);
  printf("nregions=  %d\n", Set->nregions);
  printf("caluse=    %d\n", Set->caluse);
  printf("calpoints= %d\n", Set->calpoints);
  printf("param=     0x%lx\n", Set->param);
  printf("offset=    0x%lx\n", Set->offset);
  printf("xdim=      %d\n", Set->xdim);
  printf("bitshift=  %d\n", Set->bitshift);
  printf("active=    0x%x\n", Set->active);
  printf("roipreset= %lg\n\n", Set->eventpreset);
}

int run(char *command)
{
            int err;
            if (!stricmp(command, "?"))          help();
            else if (!stricmp(command,"Q"))       return 1;
            else if (!stricmp(command,"S")) {
      err = (*lpStat)(&Status, nDev);
              if (nDev) PrintStatus(&Status);
              else PrintMpaStatus(&Status);
            }
            else if (!stricmp(command,"T")) {
                        // spectra settings
       err = (*lpSet)(&Setting, nDev);
                        printf("CHN %d:\n", nDev);
                        PrintSetting(&Setting);

              if (nDev==0) {        // MPA settings
       err = (*lpGetMCSSet)(&MCSSetting, 0);
                        PrintMCSSetting(&MCSSetting);
                                  // DATSettings
       err = (*lpGetDatSet)(&DatSetting);
                        PrintDatSetting(&DatSetting);
              }
            }
            else if (!stricmp(command,"H")) {
      (*lpHalt)(0);
            }
            else if(!strnicmp(command, "CHN=", 4)) {
              sscanf(command+4, "%d", &nDev);
              (*lpRun)(0, command);
            }
            else if (!stricmp(command,"MPA")) {
              nDev=0;
              (*lpRun)(0, command);
            }
            else {
                        (*lpRun)(0, command);
                        printf("%s\n", command);
            }
            return 0;
```

```
}

int readstr(char *buff, int buflen)
{
  int i=0,ic;

  while ((ic=getchar()) != 10) {
    if (ic == EOF) {
      buff[i]='\0';
      return 1;
    }
    if (ic == 13) ic=0;
    buff[i]=(char)ic;
    i++;
    if (i==buflen-1) break;
  }
  buff[i]='\0';
  return 0;
}

//int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmd, int nShow)
void main(int argc, char *argv[])
{
  long Errset=0, Erracq=0, Errread=0;
  char command[80];

  hDLL = LoadLibrary("DMCS6.DLL");
  if(hDLL){
     lpSet=(IMPAGETSETTING)GetProcAddress(hDLL,"GetSettingData");
              lpNewStat=(IMPANEWSTATUS)GetProcAddress(hDLL,"GetStatus");
              lpStat=(IMPAGETSTATUS)GetProcAddress(hDLL,"GetStatusData");
              lpRun=(IMPARUNCMD)GetProcAddress(hDLL,"RunCmd");
              lpCnt=(IMPAGETCNT)GetProcAddress(hDLL,"LVGetCnt");
              lpRoi=(IMPAGETROI)GetProcAddress(hDLL,"LVGetRoi");
              lpDat=(IMPAGETDAT)GetProcAddress(hDLL,"LVGetDat");
              lpStr=(IMPAGETSTR)GetProcAddress(hDLL,"LVGetStr");
              lpServ=(IMPASERVEXEC)GetProcAddress(hDLL,"ServExec");
              lpGetDatSet=(IMPAGETDATSET)GetProcAddress(hDLL,"GetDatSetting");
              lpGetMCSSet=(IMPAGETMCSSET)GetProcAddress(hDLL,"GetMCSSetting");
             // lpDigInOut=(IMPADIGINOUT)GetProcAddress(hDLL,"DigInOut");
             // lpDacOut=(IMPADACOUT)GetProcAddress(hDLL,"DacOut");
              lpStart=(IMPASTART)GetProcAddress(hDLL,"Start");
              lpHalt=(IMPAHALT)GetProcAddress(hDLL,"Halt");
              lpContinue=(IMPACONTINUE)GetProcAddress(hDLL,"Continue");
              lpErase=(IMPAERASE)GetProcAddress(hDLL,"Erase");
  }
  else return;

  // Initialize parameters
//  Errset = (*lpServ)(0);
  Errset = (*lpNewStat)(0);
  Errset = (*lpStat)(&Status, 0);
  PrintMpaStatus(&Status);

  help();

  while(TRUE)
            {
//                      scanf("%s", command);
                        readstr(command, 80);
                        if (run(command)) break;
            }
```

```
  FreeLibrary(hDLL);

  return;
}
```

5. Using the DMCS6 DLL from Delphi

In the following an example is shown how to control the MCS6A from a simple console application written in Delphi.

```
program Testmcs6a;
{$APPTYPE CONSOLE}
{$X+}
uses
  Windows, sysutils;

const   ST_RUNTIME   = 0;
        ST_OFLS      = 1;
        ST_TOTALSUM  = 2;
        ST_ROISUM    = 3;
        ST_ROIRATE   = 4;
        ST_SWEEPS    = 5;
        ST_STARTS    = 6;

type    SmallIntPointer = ^SmallInt;


{Diese Typdefinitionen wurden vom File struct.h übernommen und in Delphi
 übersetzt}
  AcqStatusTyp = RECORD          //Status information
            started   : Cardinal;          // acquisition status, 0==OFF, 1==ON,
                              // 3==READ OUT
            maxval    : Cardinal;   //
                      cnt       : array [0..7] of Double; // status: runtime in msec,
                              // ofls, total sum,
                              // roi sum, roi rate, sweeps, starts
            End;
  AcqStatusTypPointer = ^AcqStatusTyp;

  DatSettingTyp = RECORD              // Data settings
            savedata    : Cardinal; // bit 0: auto save after stop
                              // bit 1: write list file
                              // bit 2: list file only, no evaluation
            autoinc     : Cardinal; // 1 if auto increment filename
            fmt         : Cardinal; // format type: 0 == ASCII, 1 == binary,
                              // 2 == CSV
            mpafmt      : Cardinal; // format used in mpa datafiles
            sephead     : Cardinal; // seperate Header
            smpts       : Cardinal; // number of points for smoothing operation
            caluse      : Cardinal; // 1 for using calibration for shifted spectra summing
            filename    : array [0..255] of Char; // mpa data filename
            specfile    : array [0..255] of Char; // seperate spectra filename
            command     : array [0..255] of Char; // command
            End;

  ReplaySettingTyp = RECORD           // Replay settings
            use         : Cardinal; // 1 if Replay Mode ON
            modified    : Cardinal; // 1 if different settings are used
            limit       : Cardinal; // 0: all, 1: limited time range
            speed       : Cardinal; // replay speed in units of 100 kB / sec
            startsfrom  : Double;   // first start no.
            startsto    : Double;   // last start no.
            startspreset : Double;  // last start - first
            filename    : array [0..255] of Char;
            End;
```

```
  AcqSettingTyp = RECORD                    // ADC or spectra Settings
                  range      : Cardinal;   // spectrum length, ADC range
                  cftfak     : Cardinal;   // LOWORD: 256 * cft factor (t_after_peak / t_to_peak)
                             // HIWORD: max pulse width for CFT
                  roimin      : Cardinal;   // lower ROI limit
                  roimax     : Cardinal;   // upper limit: roimin <= channel < roimax
                  nregions   : Cardinal; // number of regions
                  caluse     : Cardinal; // bit 0 == 1 if calibration used, higher bits: formula
                  calpoints  : Cardinal; // number of calibration points
                  param          : Cardinal; // for MAP and POS: LOWORD=x, HIGHWORD=y (rtime=256,
RTC=257)
          offset     : Cardinal; // zoomed MAPS: LOWORD: xoffset, HIGHWORD: yoffset
                  xdim         : Cardinal; // x resolution of maps
          bitshift   : Cardinal; // LOWORD: Binwidth = 2 ^ (bitshift)
                                 // HIWORD: Threshold for Coinc
                  active     : Cardinal; // Spectrum definition words for CHN1..6:
                 // active & 0xF  ==0 not used
                 //             ==1 enabled
                                 // bit 8: Enable Tag bits
                                 // bit 9: start with rising edge
                                 // bit 10: time under threshold for pulse width
                                 // bit 11: pulse width mode for any spectra with both edges enabled
                   // Spectrum definition words for calc. spectra:
                 // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                                 //        bit4=1: x zoomed MAP
                                 //        bit5=1: y zoomed MAP
                 //             ==5 SUM, (x + y)>>xsh
                                 //             ==6 DIFF,(x - y + range)>>xsh
                                 //             ==7 ANY, (for compare)
                                 //             ==8 COPY, x
                                 //             ==10 SW-HIS, Sweep History
                 // bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                                 // HIWORD(active) = condition no. (0=no condition)
                  roipreset  : Double;   // ROI preset value
                  dummy1     : Double;   // (for future use..)
                  dummy2     : Double;   //
          dummy3     : Double;   //
          End;
  AcqSettingTypPointer = ^AcqSettingTyp;

  ExtAcqSettingTyp = RECORD                 // Settings
                  range      : Cardinal;   // spectrum length, ADC range
                  cftfak     : Cardinal;   // LOWORD: 256 * cft factor (t_after_peak / t_to_peak)
                             // HIWORD: max pulse width for CFT
                  roimin      : Cardinal;   // lower ROI limit
                  roimax     : Cardinal;   // upper limit: roimin <= channel < roimax
                  nregions   : Cardinal; // number of regions
                  caluse     : Cardinal; // bit 0 == 1 if calibration used, higher bits: formula
                  calpoints  : Cardinal; // number of calibration points
                  param          : Cardinal; // for MAP and POS: LOWORD=x, HIGHWORD=y (rtime=256,
RTC=257)
          offset     : Cardinal; // zoomed MAPS: LOWORD: xoffset, HIGHWORD: yoffset
                  xdim         : Cardinal; // x resolution of maps
          bitshift   : Cardinal; // LOWORD: Binwidth = 2 ^ (bitshift)
                                 // HIWORD: Threshold for Coinc
                  active     : Cardinal; // Spectrum definition words for CHN1..6:
                 // active & 0xF  ==0 not used
                 //             ==1 enabled
                                 // bit 8: Enable Tag bits
                                 // bit 9: start with rising edge
                                 // bit 10: time under threshold for pulse width
                                 // bit 11: pulse width mode for any spectra with both edges enabled
```

```
                // Spectrum definition words for calc. spectra:
            // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                        //        bit4=1: x zoomed MAP
                        //        bit5=1: y zoomed MAP
            //              ==5 SUM, (x + y)>>xsh
                        //              ==6 DIFF,(x - y + range)>>xsh
                        //              ==7 ANY, (for compare)
                        //              ==8 COPY, x
                        //              ==10 SW-HIS, Sweep History
            // bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                        // HIWORD(active) = condition no. (0=no condition)
              roipreset   : Double;   // ROI preset value
              dummy1      : Double;   // (for future use..)
              dummy2      : Double;   //
        dummy3      : Double;   //
              vtype       : Cardinal; // 0=single, 1=MAP, 2=ISO...
        ydim        : Cardinal; // y resolution of maps
              reserved    : array [0..12] of LongInt;
        End;
ExtAcqSettingTypPointer = ^ExtAcqSettingTyp;

AcqDataTyp = RECORD
              s0          : Array of LongInt; // pointer to spectrum
              region        : Array of Cardinal; // pointer to regions
              comment0    : Array of Char;     // pointer to strings
              cnt           : Array of Double;   // pointer to counters
              hs0           : Integer;
              hrg           : Integer;
              hcm           : Integer;
              hct           : Integer;
              End;

AcqMCSTyp = RECORD
        sweepmode   : Cardinal;       // sweepmode & 0xF: 0 = normal,
              // 1=differential (relative to first stop in sweep)
                        // 4=sequential
                        // 5=seq.+diff (Ch1), bit0 = differential mode
                        // 6 = CORRELATIONS
                        // 7 = diff.+Corr.
                        // 9=differential to stop in Ch2, bit3 = Ch2 ref (diff.mode)
                        // 0xD = seq.+diff (Ch2)
                        // 0xF = Corr.+diff (Ch2)
                        // bit 4: Softw. Start
                        // bit 6: Endless
                        // bit 7: Start event generation
                        // bit 8: Enable Tag bits
                        // bit 9: start with rising edge
                        // bit 10: time under threshold for pulse width
                        // bit 11: pulse width mode for any spectra with both edges enabled
                        // bit 12: abandon Sweepcounter in Data
                        // bit 13: "one-hot" mode with tagbits
                        // bit 14: ch6 ref (diff.mode)
                        // bit 15: enable ch6 input
              // bit 16..bit 20 ~(input channel enable)
                        // bit 24: require data lost bit in data
                        // bit 25: don't allow 6 byte datalength
        prena       : Cardinal;
            // bit 0: realtime preset enabled
                                    // bit 1: reserved
            // bit 2: sweep preset enabled
            // bit 3: ROI preset enabled
            // bit 4: Starts preset enabled
```

```
        // bit 5: ROI2 preset enabled
        // bit 6: ROI3 preset enabled
        // bit 7: ROI4 preset enabled
        // bit 8: ROI5 preset enabled
        // bit 9: ROI6 preset enabled
    cycles      : Cardinal;      // for sequential mode
    sequences   : Cardinal;      //
    syncout     : Cardinal;
          // LOWORD: sync out; bit 0..5 NIM syncout, bit 8..13 TTL syncout
                    // bit7: NIM syncout_invert, bit15: TTL syncout_invert
                    // 0="0", 1=10 MHz, 2=78.125 MHz, 3=100 MHz, 4=156.25 MHz,
                    // 5=200 MHz, 6=312.5 MHz, 7=Ch0, 8=Ch1, 9=Ch2, 10=Ch3,
                    // 11=Ch4, 12=Ch5, 13=GO, 14=Start_of_sweep, 15=Armed,
                    // 16=SYS_ON, 17=WINDOW, 18=HOLD_OFF, 19=EOS_DEADTIME
                    // 20=TIME[0],...,51=TIME[31], 52...63=SWEEP[0]..SWEEP[11]
    digio       : Cardinal;      // LOWORD: Use of Dig I/O, GO Line:
        // bit 0: status dig 0..3
        // bit 1: Output digval and increment digval after stop
        // bit 2: Invert polarity
        //  (bit 3: reserved)
        // bit 4..7:  Input pins 4..7 Trigger System 1..4
        // bit 8: GOWATCH
                    // bit 9: GO High at Start
                    // bit 10: GO Low at Stop
                    // bit 11: Clear at triggered start
                    // bit 12: Only triggered start
    digval      : Cardinal;      // digval=0..255 value for samplechanger
    dac0        : Cardinal;      // DAC0 value (START)
                                   //  bit 16: Start with rising edge
    dac1        : Cardinal;      // DAC1 value (STOP 1)
    dac2        : Cardinal;      // DAC2 value (STOP 2)
    dac3        : Cardinal;      // DAC3 value (STOP 3)
    dac4        : Cardinal;      // DAC4 value (STOP 4)
    dac5        : Cardinal;      // DAC5 value (STOP 5)
                    // bit (14,15) of each DAC: 0=falling, 1=rising, 2=both, 3=both+CFT
                    // bit 17 of each: pulse width mode under threshold
    fdac        : Cardinal;      // Feature DAC 0..16383 --> 0..2.5V
    tagbits     : Cardinal;      // Number of tagbits
    extclk      : Cardinal;      // use external clock
    maxchan     : Cardinal;      // number of input channels (=6)
    serno       : Cardinal;      // serial number
    ddruse      : Cardinal;      // bit0: DDR_USE, bit1: DDR_2GB
                                   // bits[2:3]: usb_usage
                                   // bits[4:5]: wdlen
    active      : Cardinal;      // module in system
    holdafter   : Double;        // Hold off
    swpreset    : Double;        // Sweep Preset
    fstchan     : Double;        // Acquisition delay
    timepreset  : Double;        // Realtime Preset
    End;


AcqDefTyp = RECORD
          nDevices    : Cardinal;    // Number of connected ADC Interfaces = max. 16
          nDisplays   : Cardinal;    // Number of histograms = nDevices + Positions + Maps
          nSystems    : Cardinal;    // Number of independent systems = 1
          bRemote          : Cardinal;    // 1 if server controlled by MPANT
          sys         : Cardinal;    // System definition word
        // bit0=0, bit1=0: dev#0 in system 1
        // bit0=1, bit1=0: dev#0 in system 2
        // bit0=0, bit1=1: dev#0 in system 3
        // bit0=1, bit1=1: dev#0 in system 4
```

```
                    // bit2..bit6:
                    // bit6=1, bit7=1: dev#3 in system 4
            sys0        : array [0..15] of Cardinal;
                        // System definition words for CHN1..16:
                        // see active definition in ACQSETTING
            sys1        : array [0..15] of Cardinal;
                        // CHN in System (always 1)
            End;
  AcqDefTypPointer = ^AcqDefTyp;


  LpGet = function (var Setting : AcqSettingTyp;  // Get Settings stored in the DLL
                nDisplay : Cardinal) : LongInt; stdcall;

  LpStat = function (var Status  : AcqSTatusTyp;   // Get Status stored in the DLL
                nDisplay : Cardinal) : LongInt; stdcall;

  LpRun  = procedure (nDisplay : LongInt;      // Executes command
                        Cmd    : PChar) ; stdcall;

  LpCnt  = function (var cntp    : Double;       // Copies Cnt numbers to an array
                nDisplay : Cardinal) : LongInt; stdcall;

  LpRoi  = function (var roip    : Cardinal;     // Copies the ROI boundaries to an array
                nDisplay : Cardinal) : LongInt; stdcall;

  LpDat  = function (var datp    : LongInt;      // Copies the spectrum to an array
                nDisplay : Cardinal) : LongInt; stdcall;

  LpStr  = function (var strp    : Char;         // Copies strings to an array
               nDisplay : Cardinal) : LongInt; stdcall;

  LpServ = function (ClientWnd : Cardinal) : Cardinal; stdcall;  // Register Client MCDWIN.EXE

  LpNewStat = function (nDevice : Cardinal) : LongInt; stdcall;  // Request actual Status from Server

  LpGetMCS = function (var Defmp3 : AcqMCSTyp;         // Get MCS Settings from DLL
                  nDevice : Cardinal) : LongInt; stdcall;

  LpGetDatSet = function (var Defdat : DatSettingTyp) : LongInt; stdcall;
                              // Get Data Format Definition from DLL


var  Handle         : Integer;
    TGet : LpGet;
    TStat             : LpStat;
    TRun: LpRun;
    TCnt : LpCnt;
    TRoi       : LpRoi;
    TDat       : LpDat;
    TStr       : LpStr;
    TServ      : LpServ;
    TNewStat   : LpNewStat;
    TMcs       : LpGetMCS;
    TDatset    : LpGetDatSet;
    Setting    : AcqSettingTyp;
    MCSset     : AcqMCSTyp;
    {Data             : AcqDataTyp;
    Def        : AcqDefTyp;}
    Status     : AcqStatusTyp;
    Adc        : Cardinal;
    cmd        : String;
```

```
    Err       : LongInt;
    Spec      : Array[0..8191] of LongInt;

procedure PrintMpaStatus(var stat: AcqStatusTyp);
begin
  with stat do
  begin
    if started = 1 then
      writeln('ON')
    else if started = 3 then
      writeln('READ OUT')
    else
      writeln('OFF');
    writeln('runtime=  ', cnt[ST_RUNTIME]);
    writeln('sweeps=   ', cnt[ST_SWEEPS]);
    writeln('starts=   ', cnt[ST_STARTS]);
  end;
end;

procedure PrintStatus(var stat: AcqStatusTyp);
begin
  with stat do
  begin
    writeln('totalsum=  ', cnt[ST_TOTALSUM]);
    writeln('roisum=    ', cnt[ST_ROISUM]);
    writeln('rate=      ', cnt[ST_ROIRATE]);
    writeln('ofls=      ', cnt[ST_OFLS]);
  end;
end;

procedure PrintDatSetting(var datsett: DatSettingTyp);
begin
  with datsett do
  begin
    writeln('savedata= ', savedata);
    writeln('autoinc=  ', autoinc);
    writeln('fmt=      ', fmt);
    writeln('mpafmt=   ', mpafmt);
    writeln('sephead=  ', sephead);
    writeln('filename= ', String(filename));
  end;
end;

procedure PrintMCSSetting(var mpsett: AcqMCSTyp);
begin
  with mpsett do
  begin
    writeln('sweepmode=  ', sweepmode);
    writeln('prena=      ', prena);
    writeln('cycles=     ', cycles);
    writeln('sequences=  ', sequences);
    writeln('syncout=    ', syncout);
    writeln('digio=      ', digio);
    writeln('digval=     ', digval);
    writeln('dac0=       ', dac0);
    writeln('dac1=       ', dac1);
    writeln('dac2=       ', dac2);
    writeln('dac3=       ', dac3);
    writeln('dac4=       ', dac4);
    writeln('dac5=       ', dac5);
    writeln('fdac=       ', fdac);
    writeln('tagbits=    ', tagbits);
```

```
    writeln('extclk=     ', extclk);
    writeln('maxchan=    ', maxchan);
    writeln('serno=      ', serno);
    writeln('ddruse=     ', ddruse);
    writeln('active=     ', active);
    writeln('holdafter=  ', holdafter);
    writeln('swpreset=   ', swpreset);
    writeln('fstchan=    ', fstchan);
    writeln('timepreset= ', timepreset);
  end;
end;

procedure PrintSetting(var sett: AcqSettingTyp);
begin
  with sett do
  begin
    writeln('range=    ', range);
    writeln('cftfak=   ', cftfak);
    writeln('roimin=   ', roimin);
    writeln('roimax=   ', roimax);
    writeln('nregions= ', nregions);
    writeln('caluse=   ', caluse);
    writeln('calpoints=', calpoints);
    writeln('param=    ', param);
    writeln('offset=   ', offset);
    writeln('xdim=     ', xdim);
    writeln('bitshift= ', bitshift);
    writeln('active=   ', active);
    writeln('roipreset=', roipreset);
  end;
end;

procedure PrintDat(len: Cardinal);
  var i: Integer;
begin
  writeln('first 30 of ', len, ' datapoints:');
  for i:= 0 to 29 do
    writeln(Spec[i]);
end;

procedure help;
begin
  writeln('Commands:');
  writeln('Q     Quit');
  writeln('H     Help');
  writeln('S     Status');
  writeln('G     Settings');
  writeln('CHN=x Switch to CHN #x');
  writeln('D     Get Data');
  writeln('... more see command language in MPANT Help');
end;

function run(command : String) : LongInt;
begin
  run := 0;
  if command = 'H' then
  help
  else if command = 'Q' then
  begin
    run := 1;
  end
  else if command = 'S' then
```

```
  begin
    if @TStat <> nil then
    begin
      Err := TStat(Status, Adc);
      PrintStatus(Status);
    end;
  end
  else if command = 'G' then
  begin
    if @TGet <> nil then
    begin
        Err := TGet(Setting, Adc);
        PrintSetting(Setting);
      if Adc = 0 then
      begin
        Err := TMcs(MCSset, 0);
        PrintMCSSetting(MCSset);
      end;
    end;
  end
  else if AnsiPos('CHN=',command) = 1 then
  begin
     Val(String(PChar(command)+4), Adc, Err);
     TRun(0, PChar(command));
  end
  else if command = 'D' then
  begin
    if @TGet <> nil then
    begin
        Err := TGet(Setting, Adc);
        if @TDat <> nil then
        begin
          Err := TDat(Spec[0], Adc);
          PrintDat(Setting.range);
        end;
    end;
  end
  else
  begin
    if @TRun <> nil then
    begin
      TRun(0, PChar(command));
      writeln(command);
    end;
  end;
end;

begin
  SetLength(cmd, 100);
  Adc := 0;
  Handle := LoadLibrary('dmcs6.dll');
  if Handle <> 0 then
  begin
    @TGet := GetProcAddress(Handle, 'GetSettingData');
    @TStat := GetProcAddress(Handle, 'GetStatusData');
    @TRun := GetProcAddress(Handle, 'RunCmd');
    @TCnt := GetProcAddress(Handle, 'LVGetCnt');
    @TRoi := GetProcAddress(Handle, 'LVGetRoi');
    @TDat := GetProcAddress(Handle, 'LVGetDat');
    @TStr := GetProcAddress(Handle, 'LVGetStr');
    @TServ := GetProcAddress(Handle, 'ServExec');
    @TNewStat := GetProcAddress(Handle, 'GetStatus');
```

```
    @TDatset := GetProcAddress(Handle, 'GetDatSetting');
    @TMCS := GetProcAddress(Handle, 'GetMCSSetting');

    if @TNewStat <> nil then
      Err := TNewStat(0);

 {   if @TStat <> nil then
    begin
      Err := TStat(Status, 0);
      PrintStatus(Status);
    end;

    if @TGet <> nil then
    begin
      Err := TGet(Setting, 0);
      PrintSetting(Setting);
    end; }
    help;

    repeat
      readln(cmd);
    until run(cmd) <> 0;

    FreeLibrary(Handle);
  end;
end.
```

APPENDIX: The DMCS6 DLL

The Dynamic Link Library DMCS6.DLL provides an interface to the server program MCS6.EXE that is used by the MPANT software, but can also be used by any Windows program. Custom DLL functions allow user-defined calculated parameter spectra. In the following this DLL is described in detail including the complete sourcecode.

A.1 The Structures

In struct.h some important structures are defined. A structure of type ACQSTATUS contains parameters describing the status of an acquisition. There is an array of these structures stored in the DLL, status[0] contains general mpa status data, and status[1]..status[16] ADC status data.

```
#define  WINDOWSNT
#undef  WINDOWS95
#undef  WINDOWS31

#ifdef WINDOWS31

#define GET_WM_COMMAND_ID(w)  w
#define GET_WM_COMMAND_CMD(w,l) HIWORD(l)
#define GET_WM_COMMAND_HWND(l) LOWORD(l)
#define GET_WM_SCRHWND(l) HIWORD(l)
#define GET_WM_SCROLLPOS(w,l) LOWORD(l)
#define FIND_WINDOW(a,b) FindWindow(b,a)
#define HUGE huge
#define USHORT unsigned short
#define SetForegroundWindow(w)
#define APIENTRY FAR PASCAL
#define Sleep(t) waitmsec(t)

#else

#define GET_WM_COMMAND_ID(w)  LOWORD(w)
#define GET_WM_COMMAND_CMD(w,l) HIWORD(w)
#define GET_WM_COMMAND_HWND(l) l
#define GET_WM_SCRHWND(l) l
#define GET_WM_SCROLLPOS(w,l) (short)HIWORD(w)
#define FIND_WINDOW(a,b) FindWindow(a,b)
#define HUGE
#define _fmemcpy memcpy
#define _fstrcpy strcpy

#endif

typedef struct {
	int use;
	int port;
	unsigned long baud;
	int dbits;
	int sbits;
	int parity;
	int echo;
	HWND hwndserver;
	LPSTR cmd;
} COMCTL, far *LPCOMCTL;

#define ST_RUNTIME	0
#define ST_OFLS		1
```

```
#define ST_TOTALSUM   2
#define ST_ROISUM     3
#define ST_ROIRATE    4
#define ST_SWEEPS     5
#define ST_STARTS     6

typedef struct{
  unsigned long started;     // aquisition status
  unsigned long maxval;      // maxval
  double cnt[8];             // status: runtime in msec, ofls,
                             // total sum, roi sum, roi rate, sweeps, starts
} ACQSTATUS;
```

DATSETTING is a structure type containing data format settings.

```
typedef struct {
  long savedata;        // bit 0: auto save after stop
                                                    // bit 1: write listfile
                                                    // bit 2: listfile only, no evaluation
  long autoinc;         // 1 if auto increment filename
  long fmt;             // format type (seperate spectra):
                                                    // 0 == ASCII, 1 == binary,
                    // 2 == CSV
  long mpafmt;          // format used in mpa datafiles
  long sephead;         // seperate Header
  long smpts;
  long caluse;
  char filename[256];
  char specfile[256];
  char command[256];
} DATSETTING;
```

REPLAYSETTING is a structure type containing Replay settings.

```
typedef struct {
  long use;             // 1 if Replay Mode ON
  long modified;        // Bit 0: 1 if different settings are used
                                                    // (Bit 1: Write ASCII, reserved)
  long limit;           // 0: all,
                    // 1: limited sweep range
  long speed;                   // replay speed in units of 100 kB / sec
  double startsfrom;    // first start#
  double startsto;      // last start#
  double startspreset;  // last start - first start
  char filename[256];
} REPLAYSETTING;
```

ACQSETTING is a structure type containing all the spectra settings .

```
typedef struct{
  long range;           // spectrum length
  long cftfak;          // LOWORD: 256 * cft factor (t_after_peak / t_to_peak)
                                                    // HIWORD: max pulse width for CFT
  long roimin;          // lower ROI limit
  long roimax;          // upper limit: roimin <= channel < roimax
  long nregions;        // number of regions
  long caluse;          // bit0: 1 if calibration used, higher bits: formula
  long calpoints;       // number of calibration points
  long param;           // (reserved:) for MAP and POS: LOWORD=x, HIWORD=y
  long offset;          // (reserved:) zoomed MAPS: LOWORD: xoffset, HIWORD, yoffset
  long xdim;                        //  (reserved:) x resolution of maps
```

```
  unsigned long bitshift;  // LOWORD: Binwidth = 2 ^ (bitshift)
                                                    // HIWORD: Threshold for Coinc
  long active;      // Spectrum definition words for CHN1..6:
                    // active & 0xF  ==0 not used
                    //               ==1 single
                                                    // bit 8: Enable Tag bits
                                                    // bit 9: start with rising edge
                                                    // bit 10: time under threshold for pulse width
                                                    // bit 11: pulse width mode for any spectra with both edges
enabled
                                          // Spectrum definition words for calc. spectra:
                    // active & 0xF  ==3 MAP, ((x-xoffs)>>xsh) x ((y-yoffs)>>ysh)
                    //                ((x-xoffs)>>xsh) x ((y-timeoffs)>>timesh)
                    //              or ((x-timeoffs)>>timesh x ((y-yoffs)>>ysh)
                                                    //       bit4=1: x zoomed MAP
                                                    //       bit5=1: y zoomed MAP
                                                    //            ==5 SUM, (x + y)>>xsh
                                                    //            ==6 DIFF,(x - y + range)>>xsh
                                                    //            ==7 ANY, (for compare)
                                                    //            ==8 COPY, x
                                                    //            ==9 DLL  fDLL(x,y,z),
                    //              ==0xA Sweep HISTORY, Sweepnum(x)
// bit 8..11 xsh, bit 12..15 ysh or bit 8..15 xsh
                                                    // HIWORD(active) = condition no. (0=no condition)
  double eventpreset;   // ROI preset value
  double dummy1;        // (for future use..)
  double dummy2;                    //
  double dummy3;                    //
} ACQSETTING;
```

ACQDATA is a structure type containing pointers to the data belonging to a measurement. The data is stored in a named memory-mapped file (see DLL source).

```
typedef struct{
  unsigned long HUGE *s0;          // pointer to spectrum
  unsigned long far *region;       // pointer to regions
  unsigned char far *comment0;     // pointer to strings
  double far *cnt;                 // pointer to counters
  HANDLE hs0;
  HANDLE hrg;
  HANDLE hcm;
  HANDLE hct;
} ACQDATA;
```

Data[nDisplay].s0 points to a block memory of unsigned long numbers containing the spectra data.

Data[nDisplay].region points to a block of 256 unsigned long numbers containing the Roi (Region of interest) boundaries as defined in the MPANT program. The first Roi is:
Data[nDisplay].region[0] <= x < Data[nDisplay].region[1], the second Data[nDisplay].region[2] <= x < Data[nDisplay].region[3] and so on, 128 Rois are possible. These Rois have nothing to do with the special Roi defined in the ACQSETTING structure for the Roi Preset.

Data[nDisplay].comment0 points to a block of 1024 bytes containing the strings.
Data[nDisplay].comment0[0] is the first byte of the 0. commentline,
Data[nDisplay].comment0[60] is the first byte of the 1. commentline,

Data[nDisplay].comment0[120] is the first byte of the 2. commentline,
Data[nDisplay].comment0[180] is the first byte of the 3. commentline,
Data[nDisplay].comment0[240] is the first byte of the 4. commentline,
Data[nDisplay].comment0[300] is the first byte of the 5. commentline,
Data[nDisplay].comment0[360] is the first byte of the 6. commentline,
Data[nDisplay].comment0[420] is the first byte of the 7. commentline,
Data[nDisplay].comment0[480] is the first byte of the 8. commentline,
Data[nDisplay].comment0[540] is the first byte of the 9. commentline,
Data[nDisplay].comment0[600] is the first byte of the 10. commentline,
Data[nDisplay].comment0[660] is the first byte of the data filename,
Data[nDisplay].comment0[760] is the first byte of the calibration unit name,
Data[nDisplay].comment0[800] is the first byte of the commandstring.
Data[nDisplay].comment0[880] is the first byte of the 11. commentline,
Data[nDisplay].comment0[960] is the first byte of the 12. commentline

Data[nDisplay].cnt points to a block of 448 double numbers containing:
Data[nDisplay].cnt[0] = Realtime
Data[nDisplay].cnt[1] = Totalsum
Data[nDisplay].cnt[2] = ROIsum
Data[nDisplay].cnt[3] = Totalrate
Data[nDisplay].cnt[4] = Net ROIsum
Data[nDisplay].cnt[5] = Livetime
Data[nDisplay].cnt[6] = Deadtime (%)
Data[nDisplay].cnt[11] = c0 Calibration parameter
Data[nDisplay].cnt[12] = c1 Calibration parameter
Data[nDisplay].cnt[13] = c2 Calibration parameter
Data[nDisplay].cnt[14] = c3 Calibration parameter
Data[nDisplay].cnt[19] = Channel number of first calibration Point
Data[nDisplay].cnt[35] = Energy value at first calibration Point
Data[nDisplay].cnt[20] = Channel number of 2. calibration Point
Data[nDisplay].cnt[36] = Energy value at 2. calibration Point...

Data[nDisplay].cnt[64..191] = Energy value for calibration peak in ROI 0..127
Data[nDisplay].cnt[192] = ROI Sum in ROI 0 (actualized by MPANT when selected in any spectra display)
Data[nDisplay].cnt[193] = ROI Net Sum in ROI 0 ...
Data[nDisplay].cnt[447] = ROI Net Sum in ROI 127

BOARDSETTING is a structure type describing special MCS6 hardware settings.

```
typedef struct {
  long sweepmode;      // sweepmode & 0xF: 0 = normal,
                                          // 1=differential (relative to first stop in sweep)
                                          // 4=sequential
                                          // 5=seq.+diff (Ch1), bit0 = differential mode
                                          // 6 = CORRELATIONS
                                          // 7 = diff.+Corr.
                                          // 9=differential to stop in Ch2, bit3 = Ch2 ref (diff.mode)
                                          // 0xD = seq.+diff (Ch2)
                                          // 0xF = Corr.+diff (Ch2)
                                          // bit 4: Softw. Start
                                          // bit 6: Endless
                                          // bit 7: Start event generation
                                          // bit 8: Enable Tag bits
                                          // bit 9: start with rising edge
                                          // bit 10: time under threshold for pulse width
                                          // bit 11: pulse width mode for any spectra with both edges
enabled
                                          // bit 12: abandon Sweepcounter in Data
```

```
                                                        // bit 13: "one-hot" mode with tagbits
                                                        // bit 14: ch6 ref (diff.mode)
                                                        // bit 15: enable ch6 input
                // bit 16..bit 20 ~(input channel enable)
                                                        // bit 24: require data lost bit in data
                                                        // bit 25: don't allow 6 byte datalength
 long prena;        // bit 0: realtime preset enabled
                                                        // bit 1:
                // bit 2: sweep preset enabled
                // bit 3: ROI preset enabled
                                                        // bit 4: Starts preset enabled
                // bit 5: ROI2 preset enabled
                // bit 6: ROI3 preset enabled
                // bit 7: ROI4 preset enabled
                // bit 8: ROI5 preset enabled
                // bit 9: ROI6 preset enabled
 long cycles;       // for sequential mode
 long sequences;      //
 long syncout;       // LOWORD: sync out; bit 0..5 NIM syncout, bit 8..13 TTL syncout
                                                        // bit7: NIM syncout_invert, bit15: TTL syncout_invert
                                                        // 0="0", 1=10 MHz, 2=78.125 MHz, 3=100 MHz,
4=156.25 MHz,
                                                        // 5=200 MHz, 6=312.5 MHz, 7=Ch0, 8=Ch1, 9=Ch2,
10=Ch3,
                                                        // 11=Ch4, 12=Ch5, 13=GO, 14=Start_of_sweep,
15=Armed,
                                                        // 16=SYS_ON, 17=WINDOW, 18=HOLD_OFF,
19=EOS_DEADTIME
                                                        // 20=TIME[0],...,51=TIME[31],
52...63=SWEEP[0]..SWEEP[11]
                //
 long digio;        // LOWORD: Use of Dig I/O, GO Line:
                // bit 0: status dig 0..3
                // bit 1: Output digval and increment digval after stop
                // bit 2: Invert polarity
                                                        // (bit 3: Push-Pull output, not possible)
                // bit 4..7:  Input pins 4..7 Trigger System 1..4
                                                        // bit 8: GOWATCH
                                                        // bit 9: GO High at Start
                                                        // bit 10: GO Low at Stop
                                                        // bit 11: Clear at triggered start
                                                        // bit 12: Only triggered start
 long digval;                   // digval=0..255 value for samplechanger
 long dac0;        //  DAC0 value (START)
                                                //  bit 16: Start with rising edge
 long dac1;        //  DAC1 value (STOP 1)
 long dac2;        //  DAC2 value (STOP 2)
 long dac3;        //  DAC3 value (STOP 3)
 long dac4;        //  DAC4 value (STOP 4)
 long dac5;        //  DAC5 value (STOP 5)
                                                        // bit (14,15) of each word: 0=falling, 1=rising, 2=both,
3=both+CFT
                                                        // bit 17 of each: pulse width mode under threshold
 int fdac;                              // Feature DAC 0..16383 --> 0..2.5V
 int tagbits;       // number of tagbits
 int extclk;                    // use external clock
 long maxchan;                  // number of input channels (=6)
 long serno;                    // serial number
 long ddruse;       // bit0: DDR_USE, bit1: DDR_2GB
                                                        // bits[2:3]: usb_usage
                                                        // bits[4:5]: wdlen
 long active;       // module in system
```

```
  double holdafter;          // Hold off
  double swpreset;     // sweep preset value
  double fstchan;          // acquisition delay
  double timepreset;   // time preset
} BOARDSETTING;
```

ACQDEF is a structure type describing the system definition.

```
typedef struct {
  int nDevices;        // Number of channels = number of modules * 6
  int nDisplays;       // Number of histograms = nDevices + Positions + Maps
  int nSystems;        // Number of independent systems = 1
  int bRemote;         // 1 if server controlled by MPANT
  unsigned int sys;    // System definition word:
                       // bit0=0, bit1=0: dev#0 in system 1
                       // bit0=1, bit1=0: dev#0 in system 2
                       // bit0=0, bit1=1: dev#0 in system 3
                       // bit0=1, bit1=1: dev#0 in system 4
                       // bit2..bit6:
                       // bit6=1, bit7=1: dev#3 in system 4
  int sys0[56];        // (reserved:) System definition words for CHN1..18:
                       // bit 0 CHN active
                       // bit 1 =1 CHN coinc, =0 single
                       // bit 2..4 CHN in system1..7
  int sys1[56];        // (reserved:) CHN in System
} ACQDEF;
```

A.2 The Library Functions

In the header file DMCS6.h all functions are declared. The arguments named nDevice, nDisplay pertain to the Channel number and is zero for the STOP1 channel, 5 for START/ch6. It is already listed in chapter 4.1.

A.3 The Ordinal numbers of the functions

In the Definition file DMCS6.def the ordinal numbers of the library fuctions are defined:

```
;*DMCS6.def
;*Version:        NT/9x 1.0
;*Date:           NOV-21-2007
;*Hardware:       MCS6
;*Op System:      Windows NT 4.0
;*Compiler:       MSVC++ 4.2

;*

LIBRARY DMCS6

SECTIONS
        DMCS6sh READ WRITE SHARED

EXPORTS
;       Functions in DMCS6.c
        StoreSettingData        @2
        GetSettingData          @3
        StoreStatusData         @4
        GetStatusData           @5
        Start                   @6
        Halt                    @7
        Continue                @8
        NewSetting              @9
        ServExec                @10
        StoreData               @11
        GetData                 @12
        GetSpec                 @13
        SaveSetting             @14
        GetStatus               @15
        Erase                   @16
        SaveData                @17
        GetBlock                @18
        StoreDefData            @19
        GetDefData              @20
        LoadData                @21
        NewData                 @22
        HardwareDlg             @23
        UnregisterClient        @24
        DestroyClient           @25
        ClientExec              @26
        LVGetDat                @27
        RunCmd                  @28
        AddData                 @29
        LVGetRoi                @30
        LVGetCnt                @31
        LVGetOneCnt             @32
        LVGetStr                @33
        SubData                 @34
```

Smooth	@35
StoreExtSettingData	@36
GetExtSettingData	@37
StoreMCSSetting	@38
GetMCSSetting	@39
StoreDatSetting	@40
GetDatSetting	@41
StoreReplaySetting	@42
GetReplaySetting	@43
GetDatPtr	@44
ReleaseDatPtr	@45
LVGetOneRoi	@46
GetSVal	@47
GetDatInfo	@48
BytearrayToShortarray	@49
LedBlink	@50
DigInOut	@51

A.4 The sourcecode of the functions

In the source file DMCS6.c the body of the library functions is coded:

```
/*************************************************************************
  MODULE:   DMCS6.C
  PURPOSE:  DLL to communicate with MCS6A Server
*************************************************************************/

#include "windows.h"
#include <string.h>
#include <stdio.h>
#define DLL
#include "DMCS6.h"

#ifndef WINDOWS31
#pragma data_seg("DMCS6sh")
#endif

ACQSTATUS DLLStatus[MAXDSP] = {0};
EXTACQSETTING DLLSetting[MAXDSP] = {0};
#ifdef WINDOWS31
ACQDATA DLLData[MAXDSP] = {0};
HANDLE  hInst=0;
#endif
ACQDEF DLLDef = {0};
BOARDSETTING DLLmc[MAXDEV] = {0};
DATSETTING DLLdat = {0};
REPLAYSETTING DLLRepl = {0};

BOOL    bRemote=0;
BOOL    bStatus[MAXDSP]={0};
BOOL    bSetting[MAXDSP]={0};
BOOL    bDef=FALSE;
BOOL    bmc[MAXDEV]={0};
BOOL    bDat=FALSE;
BOOL    bRepl=FALSE;
HWND    hwndServer=0;
HWND    hwndClient=0;
HWND    hwndMPANT=0;
UINT    MM_NEARCONTROL=0;
UINT    MM_GETVAL=0;
HWND    hwndMCDWIN=0;

#ifndef WINDOWS31
#pragma data_seg()
#endif

#ifdef WINDOWS31
/*************************************************************************
   FUNCTION:  WEP(int)

   PURPOSE:  Performs cleanup tasks when the DLL is unloaded.  WEP() is
             called automatically by Windows when the DLL is unloaded (no
             remaining tasks still have the DLL loaded).  It is strongly
             recommended that a DLL have a WEP() function, even if it does
             nothing but returns success (1), as in this example.

*************************************************************************/
int FAR PASCAL WEP (int bSystemExit)
{
```

```
   return(1);
}



/****************************************************************************
  FUNCTION: LibMain(HANDLE, WORD, WORD, LPSTR)

  PURPOSE:  Is called by LibEntry.  LibEntry is called by Windows when
            the DLL is loaded.  The LibEntry routine is provided in
            the LIBENTRY.OBJ in the SDK Link Libraries disk.  (The
            source LIBENTRY.ASM is also provided.)

            LibEntry initializes the DLL's heap, if a HEAPSIZE value is
            specified in the DLL's DEF file.  Then LibEntry calls
                LibMain.

                LibMain should return a value of 1 if the initialization is
                successful.
****************************************************************************/
int FAR PASCAL LibMain(hModule, wDataSeg, cbHeapSize, lpszCmdLine)
HANDLE          hModule;
WORD    wDataSeg;
WORD    cbHeapSize;
LPSTR   lpszCmdLine;
{

  hInst=hModule;
  MM_NEARCONTROL = RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  if(cbHeapSize)
    UnlockData(0);
/*
  DLLDef.nDevices = 1;
  DLLDef.nDisplays = 1;
  DLLDef.nSystems = 1;
  DLLDef.sys = 0;
  bDef = TRUE;
  bStatus = FALSE;
  bSetting = FALSE;
*/
   return 1;
}

#else

BOOL APIENTRY DllMain(HANDLE hInst, DWORD ul_reason_being_called, LPVOID lpReserved)
{
   return 1;
      UNREFERENCED_PARAMETER(hInst);
      UNREFERENCED_PARAMETER(ul_reason_being_called);
      UNREFERENCED_PARAMETER(lpReserved);
}

#endif

VOID FAR PASCAL StoreDefData(ACQDEF FAR *Def)
{
 int i;
 if(Def == NULL) {
   bDef = FALSE;
   for (i=0; i<MAXDSP; i++) {
     bSetting[i] = FALSE;
```

```
      bStatus[i] = FALSE;
    }
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLDef,(LPSTR FAR *)Def,sizeof(ACQDEF));
    bDef = TRUE;
  }
}

int FAR PASCAL GetDefData(ACQDEF FAR *Def)
{
  if (bDef) {
    DLLDef.bRemote = bRemote;
    _fmemcpy((LPSTR FAR *)Def,(LPSTR FAR *)&DLLDef,sizeof(ACQDEF));
  }
  return bDef;
}

VOID APIENTRY StoreDatSetting(DATSETTING *Defdat)
{
  if(Defdat == NULL) {
    bDat = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLdat,(LPSTR FAR *)Defdat,sizeof(DATSETTING));
    bDat = TRUE;
  }
}

int APIENTRY GetDatSetting(DATSETTING *Defdat)
{
  if (bDat) {
    _fmemcpy((LPSTR FAR *)Defdat,(LPSTR FAR *)&DLLdat,sizeof(DATSETTING));
  }
  return bDat;
}

VOID APIENTRY StoreReplaySetting(REPLAYSETTING *Repldat)
{
  if(Repldat == NULL) {
    bRepl = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLRepl,(LPSTR FAR *)Repldat,sizeof(REPLAYSETTING));
    bRepl = TRUE;
  }
}

int APIENTRY GetReplaySetting(REPLAYSETTING *Repldat)
{
  if (bRepl) {
    _fmemcpy((LPSTR FAR *)Repldat,(LPSTR FAR *)&DLLRepl,sizeof(REPLAYSETTING));
  }
  return bRepl;
}

VOID APIENTRY StoreMCSSetting(BOARDSETTING *Defmc, int ndev)
{
  if (ndev < 0 || ndev >= MAXDEV) return;
  if(Defmc == NULL) {
    bmc[ndev] = FALSE;
  }
```

```
  else{
    _fmemcpy((LPSTR FAR *)&DLLmc[ndev],(LPSTR FAR *)Defmc,sizeof(BOARDSETTING));
    bmc[ndev] = TRUE;
  }
}

int APIENTRY GetMCSSetting(BOARDSETTING *Defmc, int ndev)
{
  if (ndev < 0 || ndev >= MAXDEV) return 0;
  if (bmc[ndev]) {
    _fmemcpy((LPSTR FAR *)Defmc,(LPSTR FAR *)&DLLmc[ndev],sizeof(BOARDSETTING));
  }
  return bmc[ndev];
}

VOID FAR PASCAL StoreSettingData(ACQSETTING FAR *Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Setting == NULL) {
    bSetting[nDisplay] = FALSE;
    bStatus[nDisplay] = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLSetting[nDisplay],
          (LPSTR FAR *)Setting,sizeof(ACQSETTING));
    bSetting[nDisplay] = TRUE;
    if(Setting->range == 0L) {
      bSetting[nDisplay] = FALSE;
      bStatus[nDisplay] = FALSE;
    }
  }
}

VOID FAR PASCAL StoreExtSettingData(EXTACQSETTING FAR *Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Setting == NULL) {
    bSetting[nDisplay] = FALSE;
    bStatus[nDisplay] = FALSE;
  }
  else{
    _fmemcpy((LPSTR FAR *)&DLLSetting[nDisplay],
          (LPSTR FAR *)Setting,sizeof(EXTACQSETTING));
    bSetting[nDisplay] = TRUE;
    if(Setting->range == 0L) {
      bSetting[nDisplay] = FALSE;
      bStatus[nDisplay] = FALSE;
    }
  }
}

int APIENTRY GetSettingData(ACQSETTING FAR *Setting, int nDisplay)
{
//DebugBreak();
            if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Setting,
          (LPSTR FAR *)&DLLSetting[nDisplay],sizeof(ACQSETTING));
  }
  return bSetting[nDisplay];
}
```

```
int APIENTRY GetExtSettingData(EXTACQSETTING FAR *Setting, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Setting,
          (LPSTR FAR *)&DLLSetting[nDisplay],sizeof(EXTACQSETTING));
  }
  return bSetting[nDisplay];
}

VOID APIENTRY StoreData(ACQDATA FAR *Data, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Data == NULL) {
    bSetting[nDisplay] = FALSE;
    bStatus[nDisplay] = FALSE;
  }
#ifdef WINDOWS31
  else
    _fmemcpy((LPSTR FAR *)&DLLData[nDisplay],(LPSTR FAR *)Data,sizeof(ACQDATA));
#endif
}

int APIENTRY GetData(ACQDATA FAR *Data, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Data,(LPSTR FAR *)&DLLData[nDisplay],sizeof(ACQDATA));
  }
#endif
  return bSetting[nDisplay];
}

long APIENTRY GetSpec(long i, int nDisplay)
{
#ifdef WINDOWS31
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bSetting[nDisplay] && i < DLLSetting[nDisplay].range)
    return (DLLData[nDisplay].s0[i]);
  else return 0L;
#else
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  unsigned long val;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (!bSetting[nDisplay]) return 0;
  if (i > DLLSetting[nDisplay].range) return 0;
  sprintf(sz,"MCS6A_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 0;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hs0);
    return 0;
  }
  val = s0[i];
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
  return val;
#endif
```

```
}

VOID APIENTRY GetBlock(long FAR *hist, int start, int end, int step,
  int nDisplay)
{
#ifdef WINDOWS31
  int i,j=0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (end > DLLSetting[nDisplay].range) end = DLLSetting[nDisplay].range;
  for (i=start; i<end; i+=step, j++)
    *(hist + j) = DLLData[nDisplay].s0[i];
#else
  int i,j=0;
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!bSetting[nDisplay]) return;
  if (end > DLLSetting[nDisplay].range) end = (int)DLLSetting[nDisplay].range;
  sprintf(sz,"MCS6A_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hs0);
    return;
  }
  for (i=start; i<end; i+=step, j++)
    *(hist + j) = s0[i];
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
  return;
#endif
}

int APIENTRY LVGetDat(unsigned long HUGE *datp, int nDisplay)
{
#ifdef WINDOWS31
  long i;
  if (bSetting[nDisplay]) {
    for (i=0; i<DLLSetting[nDisplay].range; i++)
      datp[i] = DLLData[nDisplay].s0[i];
    return 0;
  }
  else return 4;
#else
  long i;
  char sz[40];
  HANDLE hs0;
  unsigned long *s0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MCS6A_S0_%d",nDisplay);
  if (!(hs0 = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(s0 = (unsigned long *)MapViewOfFile(hs0,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hs0);
    return 4;
  }
  for (i=0; i<DLLSetting[nDisplay].range; i++)
      datp[i] = s0[i];
```

```
  UnmapViewOfFile(s0);
  CloseHandle(hs0);
#endif
}

HANDLE hEXMDisplay=0;
unsigned long * EXMDisplay=NULL;

int APIENTRY GetDatInfo(int nDisplay, long *xmax, long *ymax)
{
//DebugBreak();
  if (nDisplay < 0 || nDisplay >= MAXDSP) return -1;
  if (!bSetting[nDisplay]) return -1;
  *xmax = DLLSetting[nDisplay].xdim;
  *ymax = DLLSetting[nDisplay].range;
  if (*xmax) *ymax /= *xmax;
  else {*xmax = *ymax; *ymax = 1;}
  return DLLSetting[nDisplay].range;
}

int APIENTRY GetDatPtr(int nDisplay, long *xmax, long *ymax, unsigned long * *pt)
{
  char sz[40];
//DebugBreak();
  if (nDisplay < 0 || nDisplay >= MAXDSP) return -1;
  if (!bSetting[nDisplay]) return -1;
  *xmax = DLLSetting[nDisplay].xdim;
  *ymax = DLLSetting[nDisplay].range;
  if (*xmax) *ymax /= *xmax;
  else {*xmax = *ymax; *ymax = 1;}
  sprintf(sz,"MCS6A_S0_%d",nDisplay);
  ReleaseDatPtr();
  if (!(hEXMDisplay = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 0;
  if (!(EXMDisplay = MapViewOfFile(hEXMDisplay,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hEXMDisplay);
    return 0;
  }
  *pt = EXMDisplay;
  return (int)hEXMDisplay;
}

int APIENTRY ReleaseDatPtr()
{
  if(EXMDisplay)
            UnmapViewOfFile(EXMDisplay);
  EXMDisplay = NULL;
  if(hEXMDisplay)
            CloseHandle(hEXMDisplay);
  hEXMDisplay = 0;
  return 0;
}

int APIENTRY LVGetRoi(unsigned long FAR *roip, int nDisplay)
{
#ifdef WINDOWS31
  int i,n;
  n = 2 * DLLSetting[nDisplay].nregions;
  if (bSetting[nDisplay]) {
    for (i=0; i<n; i++)
      roip[i] = DLLData[nDisplay].region[i];
```

```
    return 0;
  }
  else return 4;
#else
  int i,n;
  char sz[40];
  HANDLE hrg;
  unsigned long *region;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MCS6A_RG_%d",nDisplay);
  if (!(hrg = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(region = (unsigned long *)MapViewOfFile(hrg,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hrg);
    return 4;
  }
  n = 2 * DLLSetting[nDisplay].nregions;
  for (i=0; i<n; i++)
    roip[i] = region[i];
  UnmapViewOfFile(region);
  CloseHandle(hrg);
  return 0;
#endif
}

int APIENTRY LVGetOneRoi(int nDisplay, int roinum, long *x1, long *x2)
{
#ifdef WINDOWS31
  if (bSetting[nDisplay] && (roinum > 0 && (roinum <= 128)) {
    *x1 = DLLData[nDisplay].region[2*(roinum-1)];
    *x2 = DLLData[nDisplay].region[2*(roinum-1)+1];
    return 0;
  }
  else return 4;
#else
  char sz[40];
  HANDLE hrg;
  unsigned long *region;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay] || (roinum < 1) || (roinum > 128)) return 4;
  sprintf(sz,"MCS6A_RG_%d",nDisplay);
  if (!(hrg = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(region = (unsigned long *)MapViewOfFile(hrg,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hrg);
    return 4;
  }
  *x1 = region[2*(roinum-1)];
  *x2 = region[2*(roinum-1)+1];
  UnmapViewOfFile(region);
  CloseHandle(hrg);
  return 0;
#endif
}

int FAR PASCAL LVGetCnt(double FAR *cntp, int nDisplay)
{
#ifdef WINDOWS31
  int i;
```

```
  if (bSetting[nDisplay]) {
    for (i=0; i<MAXCNT; i++)
      cntp[i] = DLLData[nDisplay].cnt[i];
    return 0;
  }
  else return 4;
#else
  int i;
  char sz[40];
  HANDLE hct;
  double *cnt;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MCS6A_CT_%d",nDisplay);
  if (!(hct = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(cnt = (double *)MapViewOfFile(hct,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hct);
    return 4;
  }
  for (i=0; i<MAXCNT; i++)
    cntp[i] = cnt[i];
  UnmapViewOfFile(cnt);
  CloseHandle(hct);
  return 0;
#endif
}

int APIENTRY LVGetOneCnt(double *cntp, int nDisplay, int cntnum)
                        // Get one Cnt number
{
#ifdef WINDOWS31
  if (bSetting[nDisplay]) {
    *cntp = DLLData[nDisplay].cnt[cntnum];
    return 0;
  }
  else return 4;
#else
  char sz[40];
  HANDLE hct;
  double *cnt;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MCS6A_CT_%d",nDisplay);
  if (!(hct = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(cnt = (double *)MapViewOfFile(hct,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hct);
    return 4;
  }
  *cntp = cnt[cntnum];
  UnmapViewOfFile(cnt);
  CloseHandle(hct);
  return 0;
#endif
}

int APIENTRY LVGetStr(char FAR *strp, int nDisplay)
{
#ifdef WINDOWS31
```

```
  int i;
  if (bSetting[nDisplay]) {
    for (i=0; i<1024; i++)
      strp[i] = DLLData[nDisplay].comment0[i];
    return 0;
  }
  else return 4;
#else
  int i;
  char sz[40];
  HANDLE hcm;
  char *comment0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 4;
  if (!bSetting[nDisplay]) return 4;
  sprintf(sz,"MCS6A_CM_%d",nDisplay);
  if (!(hcm = OpenFileMapping(FILE_MAP_READ, FALSE, sz)))
            return 4;
  if (!(comment0 = (char *)MapViewOfFile(hcm,
        FILE_MAP_READ, 0, 0, 0))) {
    CloseHandle(hcm);
    return 4;
  }
  for (i=0; i<1024; i++)
    strp[i] = comment0[i];
  UnmapViewOfFile(comment0);
  CloseHandle(hcm);
  return 0;
#endif
}

VOID APIENTRY StoreStatusData(ACQSTATUS FAR *Status, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if(Status == NULL)
    bStatus[nDisplay] = FALSE;
  else{
    _fmemcpy((LPSTR FAR *)&DLLStatus[nDisplay],
        (LPSTR FAR *)Status,sizeof(ACQSTATUS));
    bStatus[nDisplay] = TRUE;
  }
}

int APIENTRY GetStatusData(ACQSTATUS FAR *Status, int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return 0;
  if (bStatus[nDisplay]) {
    _fmemcpy((LPSTR FAR *)Status,
        (LPSTR FAR *)&DLLStatus[nDisplay],sizeof(ACQSTATUS));
  }
  return bStatus[nDisplay];
}

VOID APIENTRY Start(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_START, 0L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_START2, 0L);
```

```
    break;
   case 2:
    PostMessage(hwndServer, WM_COMMAND, ID_START3, 0L);
    break;
   case 3:
    PostMessage(hwndServer, WM_COMMAND, ID_START4, 0L);
    break;
  }
}

VOID APIENTRY Halt(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  switch (nSystem) {
   case 0:
    PostMessage(hwndServer, WM_COMMAND, ID_BREAK, 0L);
    break;
   case 1:
    PostMessage(hwndServer, WM_COMMAND, ID_BREAK2, 0L);
    break;
   case 2:
    PostMessage(hwndServer, WM_COMMAND, ID_BREAK3, 0L);
    break;
   case 3:
    PostMessage(hwndServer, WM_COMMAND, ID_BREAK4, 0L);
    break;
  }
}

VOID APIENTRY Continue(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  switch (nSystem) {
   case 0:
    PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE, 0L);
    break;
   case 1:
    PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE2, 0L);
    break;
   case 2:
    PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE3, 0L);
    break;
   case 3:
    PostMessage(hwndServer, WM_COMMAND, ID_CONTINUE4, 0L);
    break;
  }
}

VOID APIENTRY SaveSetting()
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_SAVE, 0L);
}

VOID APIENTRY NewSetting(int nDev)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  //if (nDev>=0 && nDev<MAXDSP) bStatus[nDev] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_NEWSETTING, 0L);
}
```

```
VOID APIENTRY NewData()
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_NEWDATA, 0L);
}

int APIENTRY GetStatus(int nDev)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  if (bStatus[nDev]) {
      SendMessage(hwndServer, WM_COMMAND, ID_GETSTATUS, 0L);
  }
  return bStatus[nDev];
}

UINT APIENTRY ServExec(HWND ClientWnd)
{
  bRemote = 1;
  hwndClient = ClientWnd;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  if (hwndServer) {
    ShowWindow(hwndServer, SW_MINIMIZE);
    return 32;
  }
  else
    return WinExec("mcs6a.exe", SW_SHOW);
}

UINT APIENTRY ClientExec(HWND ServerWnd)
{
  if (ServerWnd) hwndServer = ServerWnd;
  return WinExec((LPSTR)"MPANT /device=MCS6A", SW_SHOW);
}

VOID APIENTRY UnregisterClient()
{
  hwndClient = 0;
  bRemote = 0;
}

VOID APIENTRY DestroyClient()
{
  bRemote = 0;
  if (hwndClient) SendMessage(hwndClient, WM_CLOSE, 0, 0L);
  hwndClient = 0;
}

VOID APIENTRY Erase(int nSystem)
{
  if (nSystem < 0 || nSystem > 3) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  switch (nSystem) {
    case 0:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE, 0x10000L);
     break;
    case 1:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE2, 0L);
     break;
    case 2:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE3, 0L);
     break;
```

```
    case 3:
     PostMessage(hwndServer, WM_COMMAND, ID_ERASE4, 0L);
     break;
  }
}

VOID APIENTRY SaveData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  PostMessage(hwndServer, WM_COMMAND, ID_SAVEFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY LoadData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_LOADFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY AddData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SUMFILE,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY SubData(int nDisplay, int all)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
//  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SUBTRACT,
                        MAKELPARAM((WORD)nDisplay, (WORD)all));
}

VOID APIENTRY Smooth(int nDisplay)
{
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  bStatus[nDisplay] = FALSE;
  PostMessage(hwndServer, WM_COMMAND, ID_SMOOTH,
                        MAKELPARAM((WORD)nDisplay, (WORD)0));
}

VOID FAR PASCAL HardwareDlg(int item)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server", NULL);
  switch (item) {
    case 0:
      PostMessage(hwndServer, WM_COMMAND, ID_HARDWDLG, 0L);
      break;
    case 1:
      PostMessage(hwndServer, WM_COMMAND, ID_DATADLG, 0L);
      break;
    case 2:
      PostMessage(hwndServer, WM_COMMAND, ID_COMBDLG, 0L);
```

```
      break;
    case 3:
      PostMessage(hwndServer, WM_COMMAND, ID_MAPLSTDLG, 0L);
      break;
    case 4:
      PostMessage(hwndServer, WM_COMMAND, ID_REPLDLG, 0L);
      break;
  }
}

VOID APIENTRY RunCmd(int nDisplay, LPSTR Cmd)
{
#ifdef WINDOWS31
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  if (Cmd != NULL) {
    _fstrcpy(&DLLData[0].comment0[800], Cmd);
  }
#else
  char sz[40];
  HANDLE hcm;
  char *comment0;
  if (nDisplay < 0 || nDisplay >= MAXDSP) return;
  if (!bSetting[nDisplay]) return;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  sprintf(sz,"MCS6A_CM_%d",nDisplay);
  if (!(hcm = OpenFileMapping(FILE_MAP_WRITE, FALSE, sz)))
            return;
  if (!(comment0 = (char *)MapViewOfFile(hcm,
        FILE_MAP_WRITE, 0, 0, 0))) {
    CloseHandle(hcm);
    return;
  }
  strcpy(&comment0[800], Cmd);
#endif
  SendMessage(hwndServer, MM_NEARCONTROL, (WPARAM)ID_RUNCMD, (LONG)(LPSTR)Cmd);
#ifndef WINDOWS31
  strcpy(Cmd, &comment0[1024]);
  UnmapViewOfFile(comment0);
  CloseHandle(hcm);
#endif
}

long APIENTRY GetSVal(int DspID, long xval)
{
  long val=0;
  if (xval == -2) {
              hwndMPANT = FIND_WINDOW("mpwframe",NULL);
              return (long)hwndMCDWIN;
              // should be called first to be sure that MCDWIN is started
  }
  if (!hwndMPANT) hwndMPANT = FIND_WINDOW("mpwframe",NULL);
  if (!MM_GETVAL) MM_GETVAL = RegisterWindowMessage((LPSTR)"MCS6AGetval");
  val = SendMessage(hwndMPANT, MM_GETVAL, (WPARAM)DspID, (LPARAM)xval);
    // for xval == -1 returns Display size
  return val;
}
```

```
int APIENTRY BytearrayToShortarray(short *Shortarray, char *Bytearray, int length)
{
  int i;
  char c;
  for (i=0; i<length; i++) {
            c = Bytearray[i];
            Shortarray[i] = c;
            if (!c) {
                      i++; break;
            }
  }
  return i;
}

int APIENTRY LedBlink(int nDev)
{
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  switch(nDev) {
  case 0:
            PostMessage(hwndServer, WM_COMMAND, ID_LEDBLINK0, 0); break;
  case 1:
            PostMessage(hwndServer, WM_COMMAND, ID_LEDBLINK1, 0); break;
  case 2:
            PostMessage(hwndServer, WM_COMMAND, ID_LEDBLINK2, 0); break;
  }
  return 0;
}

int APIENTRY DigInOut(int value, int enable)  // controls Dig I/0 ,
                                                                                    // returns digin
{
  int val=0;
  long lval;
  if (!hwndServer) hwndServer = FIND_WINDOW("MCS6A Server",NULL);
  if (!MM_NEARCONTROL) MM_NEARCONTROL =
RegisterWindowMessage((LPSTR)"MPANEARCONTROL");
  lval = ((long)value & 0xFF) | ((enable & 0xFF)<<8);
  val = SendMessage(hwndServer, MM_NEARCONTROL, ID_DIGINOUT, (LONG)lval);
  return val;
}
```

A.5 How to compile the DLL

The 32 bit DLL can be compiled with the Microsoft Visual C/C++ compiler version 4.2 or higher. To recompile the DLL under VC 4.2, use the makefile DMCS6.mak. For higher version create a new DLL project and include the files DMCS6.c and DMCS6.def.