DP4/PX4/DP5 Digital Pulse Processor API Reference

Amptek, Inc.
14 De Angelo Dr.
Bedford MA 01730
781-275-2242 **www.amptek.com**

1 DP4/PX4/DP5 DIGITAL PULSE PROCESSOR API (DPPAPI) BASICS

The DP4/PX4/DP5 Digital Pulse Processor API is an Application Programming Interface Library for Amptek DP4, PX4, and DP5 Digital Pulse Processors. **The DP4, PX4, and DP5 will be referred to collectively as DPP (Digital Pulse Processor).** The DPPAPI communicates to the DPP via USB. The DPPAPI can select a DPP from a number of DPP devices attached to a system. The DPPAPI runs on personal computers with Intel® Pentium® or higher equivalent processor , Microsoft Windows® 98 (SE), Me, 2000, or XP and 64 MB RAM (128 MB recommended).

Application programs invoke DPPAPI functions to establish DPP communications, control DPP operations, and retrieve spectrum data. The DPPAPI is distributed with an import library (.LIB), and a dynamic link library (.DLL) and a type library (.TLB). Applications can be developed with the DPPAPI in C++, Visual Basic and many other programming languages.

The C language representation of DPPAPI functions are defined in the DppApi.h header file. These files are provided with the DPPAPI libraries. The USB drivers provided with your Amptek DPP must be installed and the usbdrvd.dll must be accessible to the DPPAPI in order to function properly.

1.1 Programming Guidelines

1.1.1 Accessing the API

Before any interaction with the DPP can take place, the application program must open an instance of the DPPAPI by calling the OpenDppApi function. The OpenDppApi function call creates a handle to the DPPAPI which is then passed to all DPPAPI functions.

After all DPP operations have completed and before the application is terminated the DPPAPI must be closed by calling the CloseDppApi function. The application should always call CloseDppApi before exiting the program.

1.1.2 Establishing Communications

The DPPAPI supports USB communications for one DPP device (called a "connection") from a pool of one or more devices. Before configuring or acquiring data, USB communications must be established. This is done by calling OpenUSBDevice. OpenUSBDevice returns the number of the devices detected. If the number of devices is zero, the DPP cannot be detected. Check if the DPP is ON and the USB cable is plug into the computer. Once a connection is opened, MonitorUSBDppDevices can be used to check the USB connection. When communications are done call CloseUSBDevice to close the USB connection. CloseUSBDevice must be called before CloseDppApi is called.

1.1.3 Selecting a DPP from Multiple Devices

To select a DPP from a pool of DPP devices, the devices first must be enumerated. The enumeration process begins with a call to OpenUSBDevice to count the number of DPP devices attached. The number of devices is saved and the connection is closed (CloseUSBDevice).

Each DPP must then be identified. The device index ranges from 1 to the number of DPP devices. Each device index is matched to the device type and serial number. OpenUSBDeviceEx opens a device connection for a specific index. GetUSBDppDeviceInfo returns the DPP device type and DPP serial number for the opened device. CloseUSBDevice must be called before another device is opened and before CloseDppApi is called.

1.1.4 The DPPAPI Device Index

The DPPAPI stores a copy of the DPP device index to determine which USB device to communicate to. Only two functions can change this value, OpenDppApi and OpenUSBDeviceEx. The DPPAPI device index defaults to one when the DPPAPI is first opened (OpenDppApi). Calls to OpenUSBDeviceEx changes the DPPAPI device index to the selected device.

OpenUSBDevice always opens the first device if only one device is used and opens the last index called by OpenUSBDeviceEx otherwise. The DPPAPI device index can be reset by reopening the DPPAPI or calling OpenUSBDeviceEx with an index of one.

1.1.5 DPP Status

The status can be read in a number of formats. Status formats include *display string*, *byte block*, and *status structure*. DPP status includes identification, configuration, monitor, and acquisition information. GetStatusStruct returns the status in a structure. Status returned in a status structure can tested and displayed. Example functions are provided demonstrating how to check the DPP status.

1.1.6 DPP Configuration

The DPP and the DPPAPI must each have a working configuration to perform data acquisition. The DPP and the DPPAPI can get their configurations from different sources. The simplest example is a PX4 that has been configured at power-up by holding the power button down until the device beeps twice. The PX4 configuration has been loaded at power-up, so the only remaining configuration task is to get the configuration being used from the PX4 by calling GetConfigFromDpp. This copies the configuration stored in the PX4 hardware and loads the configuration into the DPPAPI program. Even though the PX4 is configured, the DPPAPI must have the configuration to properly acquire data and perform configuration management.

1.1.7 DP5 Configuration Notes

The DP5 has additional features that add more functionality. The FPGA Clock (DP5 ONLY) allows for more control over peaking time. The DP5 FPGA 80MHz Clock Mode indicator must be set to correctly interpret DP5 configuration file settings. The FPGA 80MHz Clock indicator is set when a DPP status command is issued and a DPP device is attached or when SetFPGAClockDefault sets this value. **SetFPGAClockDefault is needed when editing DP5 configuration files without a DP5 device.** Get80MHzMode reads the DP5 FPGA 80MHz Clock Mode indicator. The DP5 FPGA Clock setting has 2 speeds, 20MHz (normal speed = 1) and 80MHz (high speed = 1). Always read the status before configuring a DP5.

1.1.8 DP5 Boot Options

DP5 Boot Options allow new configuration options not available with either the DP4 or the PX4. GetBootOptionsStruct reads the DP5 Boot Options. DP5 Boot Options can be changed calling SendDppVendorRequest with corresponding Vendor Request values.

1.1.9 Acquiring Data

Data can be acquired after the DPP and the DPPAPI have been configured. If the DPP is already acquiring data the only call necessary to acquire data is GetDppData. The data is stored in a buffer ready to be plotted. GetDppData returns the number of channels represented in the spectrum data. Data acquisition can be enabled and paused by calling the PauseDppData function. The data channels are cleared by calling ClearDppData. The data returned by GetDppData function is a summary of the DPP histogram memory: The histogram memory operates as in a traditional MCA. When a pulse occurs with a particular peak value, a counter in a corresponding memory location is incremented. The result is a histogram, an array containing, in each cell, the number of events with the corresponding peak value. This is the energy spectrum and is the primary output of the DPP. The DPP uses 3 bytes per channel, which allows up to 16.7M counts per channel. The 3 bytes per channel are combined producing the final the energy spectrum ready to plot.

1.1.10 DPPAPI Programming Summary

The typical tasks required to setup and acquire data can be summarized as follows:

- open the DPPAPI
- open USB communications
- get the device status
- configure the software
- configure the hardware
- pause the DPP data acquisition
- clear the data buffer
- start the data acquisition
- repeat the next 2 steps until data acquisition finished
    a. get the acquired data buffer
    b. plot data
- close usb
- close DPPAPI

Each DPPAPI task can be completed by a function call. A timer is needed for the data acquisition. A method to plot the data must also be added.

1.2 C++ Data Acquisition Example

```
void CvcMinExDlg::OnBnClickedGetdatabutton() {
    GetDlgItem(IDC_GETDATABUTTON)->EnableWindow(false); // disable this button until done
    int numdev;                                         // number of usb devices
    void * objDppApi;                                   // pointer to DPPAPI
    char szStatus[2000];                                // status text
    long DataBuffer[8192];                              // spectrum data
    CString cstrStatus;                                 // status display string
    int NumChan;                                        // number of acquired channels
    DPP_STATUS StatusLst;                               // status structure

    objDppApi = OpenDppApi();                           // Create/Open DPPAPI
    numdev = OpenUSBDevice(objDppApi);                  // Open USB communications
    if (numdev > 0) {
        GetStatusStruct(objDppApi, true, &StatusLst);   // get device status structure
        if (StatusLst.SerialNumber < 1) {
            GetDlgItem(IDC_STATUS)->SetWindowText("Device status error.");
        } else if (! StatusLst.SwConfigRcvd) {
            GetDlgItem(IDC_STATUS)->SetWindowText("Please configure device before taking data.");
        } else {
            if (! StatusLst.StatMcaEnabled) {
                PauseDppData(objDppApi, false);             // enable data acquisition
            }
            GetStatusString(objDppApi, 1, szStatus, 2000);  // get device status display string
            cstrStatus = szStatus;
            GetDlgItem(IDC_STATUS)->SetWindowText(cstrStatus);  // display status
            GetConfigFromDpp(objDppApi);                    // configure from stored configuration
            NumChan = GetDppData(objDppApi, DataBuffer);    // aquire data
            PlotData(DataBuffer, NumChan);                  // plot data
        }
    }
    CloseUSBDevice (objDppApi);                             // close usb
    CloseDppApi (objDppApi);                                // close DPPAPI
    GetDlgItem(IDC_GETDATABUTTON)->EnableWindow(true);
}
```

1.3 VB Data Acquisition Example

```
Private Sub cmdGetData_Click()
    cmdGetData.Enabled = False                          ' disable button until done
    Dim numdev As Integer                               ' number of usb devices
    Dim objDppApi As Long                               ' pointer to dpp api
    Dim szStatus As String * 2000                       ' status text
    Dim DataBuffer(8192) As Long                        ' spectrum data
    Dim NumChan As Integer                              ' number of acquired channels

    objDppApi = OpenDppApi()                            ' create/open DPPAPI
    numdev = OpenUSBDevice(objDppApi)                   ' open USB communications
    If (numdev > 0) Then
        GetStatusString objDppApi, 1, szStatus, 2000    ' get device status
        txtStatus = szStatus
        GetConfigFromDpp objDppApi                      ' configure from hardware
        NumChan = GetDppData(objDppApi, VarPtr(DataBuffer(0)))  ' acquire data
        PlotData DataBuffer, NumChan                    ' plot data
    End If
    CloseUSBDevice (objDppApi)                          ' close usb
    CloseDppApi (objDppApi)                             ' close DPPAPI
    cmdGetData.Enabled = True

End Sub
```

2 DPPAPI PROGRAMMING TOPICS

2.1 Rapid Prototyping Solutions

The DPPAPI was designed for fast and simple development. The source code examples included with the DPPAPI demonstrate most of the functionality. Example code can be cut and pasted into applications to save time and development effort.

- Start with the closest model to the required application.
- Borrow from the other examples.
- Complete and test the basic DPP specific tasks.
- Customize the application.

2.2 USB

The DPPAPI USB interface is a wrapper around the APAUSB interface. No special programming is required. The DPPAPI USB interface includes all the functionality to control and communicate with DP4, PX4 and DP5 type devices. Almost all necessary USB functions are completed automatically by the DPPAPI. The only functions required by the programmer are to open and close the USB device and to monitor if the device has been removed or disconnected. For details of the DPPAPI USB functions see the *USB Manager Functions* section.

NOTE: The usbdrvd.dll APAUSB driver library must be accessible to the DPPAPI.

3 CONFIGURATIONS

3.1 READING AND WRITING DPP DEVICE CONFIGURATIONS

3.1.1 The DP4 Configuration Cycle

The DP4 stores a copy of the current configuration that can be read back **until the unit is powered down.** Each time a DP4 is powered up, it must be sent a valid configuration. Each time a configuration parameter is changed, a copy of the current configuration must be sent to the DP4. It is also recommended that the new configuration be saved to a file for future reference. There are two methods to detect if a DP4 has been configured with a valid configuration. The *Software Configuration Received* boot status bit can be checked, or the configuration can be read back from the DP4 and compared to the configuration sent.

DP4 Start Up Configuration Sequence:

- Load a configuration file (**GetConfigFromFile**)
- Send a configuration to the DP4 (**SendConfigToDPP**)

DP4 Configuration Update Sequence:

- Update the current configuration (**GetTempConfigSettings**, **SetTempConfigSettings**, etc.)
- Send the new configuration to the DP4 (**SendConfigToDPP**)
- Save the current configuration to file (**SaveConfigToFile**)

3.1.2 The PX4 Configuration Cycle

The PX4 stores a copy of the current configuration in EEPROM. It can be recalled at power up by pressing the power button for about three seconds (until two beeps are heard). If the PX4 has been configured by recalling the stored configuration at power up, the *Power Button Configured* boot status bit will be set. Each time a PX4 is powered up, either the current configuration must be recalled or a valid configuration must be sent. Each time a configuration parameter is changed, a copy of the current configuration must be sent to the PX4. It is also recommended that the new configuration be saved to a file for future reference. There are three methods to detect if a PX4 has been configured with a valid configuration. The *Software Configuration Received* and *Power Button Configured* boot status bits can be checked, or the configuration can be read back from the PX4 and compared to the configuration sent.

Px4 Start Up Configuration Sequence, Power Up Configured Method:

- Power up configure PX4
- Read the current configuration from the PX4 (**GetConfigFromDPP**)
- (Optional but recommended) Send a configuration to the PX4 (**SendConfigToDPP**)

Px4 Start Up Configuration Sequence, Configure from File Method:

- Same as the DP4 Start Up Configuration Sequence

Px4 Configuration Update Sequence:

- DP4 Configuration Update Sequence

3.1.3 The DP5 Configuration Cycle

The DP5 stores a copy of the current configuration in EEPROM. It can be recalled at power up by setting the "**Boot configuration state**" boot flag to "**Use nonvolatile configuration**". If the DP5 has been configured by recalling the stored configuration at power up, the *Power Button Configured* boot status bit will be set. Each time a configuration parameter is changed, a copy of the current configuration must be sent to the DP5. It is also recommended that the new configuration be saved to a file for future reference. There are three methods to detect if a DP5 has been configured with a valid configuration. The *Software Configuration Received* and *Power Button Configured* boot status bits can be checked, or the configuration can be read back from the DP5 and compared to the configuration sent. The DP5 configuration sequence is the same as the PX4.

3.1.4 DP5 Configuration Notes

- The FPGA Clock indicator must be set with SetFPGAClockDefault if a configuration file being edited for a DP5 without a DP5 connected.
- Before configuring a DP5 always read the DP5 status. This sets the FPGA Clock indicator.

3.1.5 DPPAPI Configuration Details

The DPPAPI stores a working copy of the current DPP configuration (**active configuration**). After the DPPAPI is opened, a configuration is read from file or transferred from a DPP device. In the case of the DP4, the active configuration is always read from a file. In the case of the PX4, the initial configuration can be read from a file or from PX4 configuration storage memory. If the configuration is changed, it is recommended that a copy be saved to a file for future reference.

The DPPAPI also stores a spare copy of the current DPP configuration (**temporary configuration**). The temporary configuration can be used as a scratchpad for configuration updates. GetTempConfigSettings and SetTempConfigSettings read/write DPP configuration settings from the active configuration and temporary configuration to/from a DPP_CONFIG_SETTINGS structure. The DPP_CONFIG_SETTINGS structure values are designed to be used with dialog controls.

3.1.6 DPPAPI Active and Temporary Configurations

The DPPAPI stores an active and a temporary configuration. The active configuration is the main DPPAPI configuration. The DPPAPI uses the active configuration for configuration transfer operations. Configuration transfer operations include reading/writing configuration files and sending/receiving DPP device configurations. The temporary configuration is used for configuration editing. When ever a DPPAPI configuration is edited, a copy of the temporary is used.

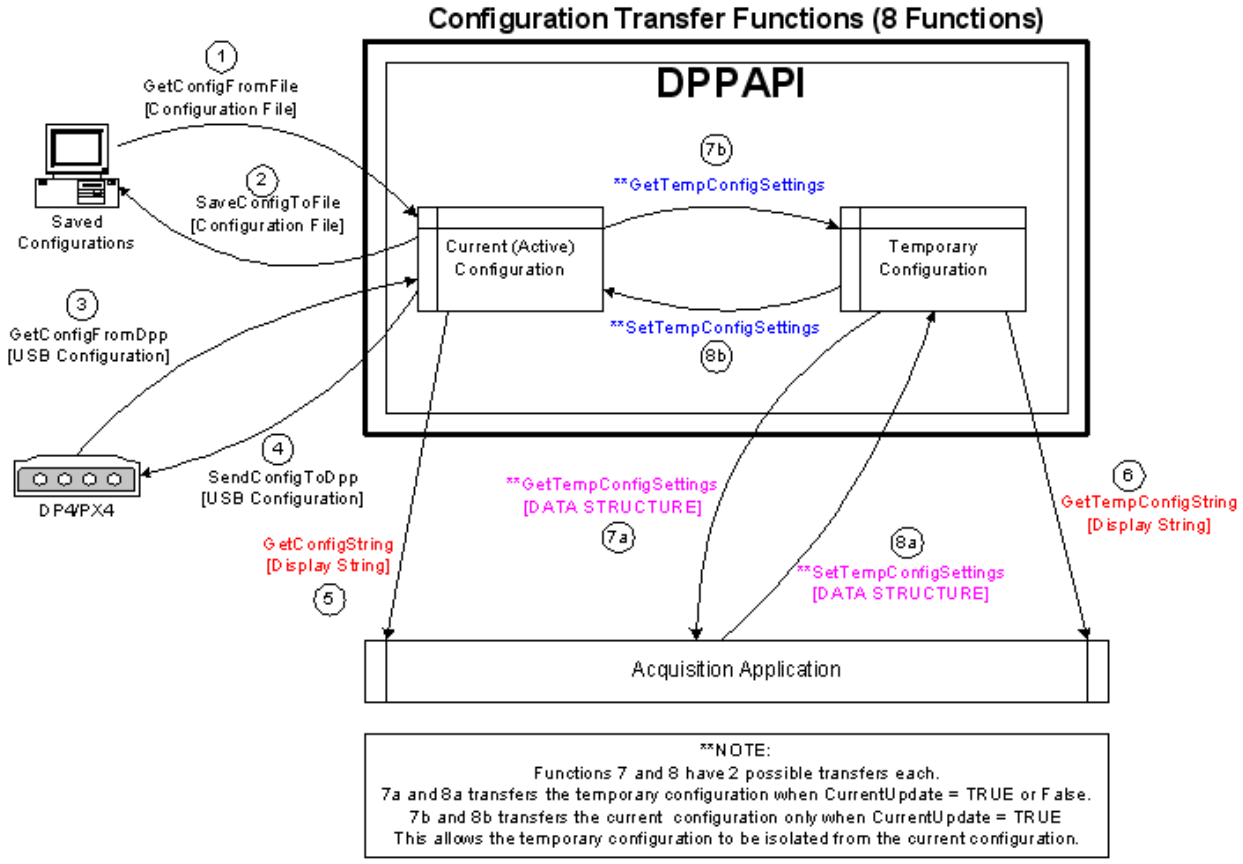
The CurrentUpdate parameter in GetTempConfigSettings and SetTempConfigSettings set to "1 (TRUE)" causes the active configuration settings to be updated along with the temporary settings. This allows the active configuration to be read and directly updated.

The CurrentUpdate parameter in GetTempConfigSettings and SetTempConfigSettings set to "0 (FALSE)" causes the active configuration settings to be isolated from the temporary settings. This allows the temporary configuration to be read and updated independently from the active configuration.

3.1.7 Editing Configurations and the CurrentUpdate Parameter

Set the CurrentUpdate parameter in GetTempConfigSettings and SetTempConfigSettings to "1 (TRUE)" to read and directly update the active configuration.

3.2 Configuration Transfer Functions Diagram



4 DPP STATUS OPERATION

DPP Status provides device information for normal operations. Status information can be read as a block of data, a status structure or a display string. Status information includes device identification, spectrum accumulation status, monitors, and boot status. Also see the DPPAPI *STATUS INDICATOR FUNCTIONS* section.

4.1 DPP STATUS FUNCTION GUIDE

4.1.1 Getting Complete Status with One Function Call

Only one function call is required to get a full snapshot of the current status. Status can be read as a structure with **GetStatusStruct**. In a structure (see DPP_STATUS) the status values are represented in numeric form. This format allows for the values to be easily tested. GetStatusStruct is the preferred method of reading the status. Values in the DPP_STATUS structure can also easily be converted into strings for display.

4.1.2 Returning Display Status with One Function Call

GetStatusString returns a subset of the status for display purposes.

GetStatusString Display

Device Type = PX4
Fast Count = 660
Slow Count = 495
FPGA = 4.00
Firmware = 4.00
Serial Number = 1070
Accumulation Time = 72
StatMcaEnabled = True

4.1.3 Advanced Status Functions

GetStatusBuffer *only* returns the status as a block of bytes (raw data). ***Bytes within the data block must be decoded.*** The data can be manually decoded (see the PX4 Programmer's Guide) or **ProcessStatusBuffer** to decode the status values. If **ProcessStatusBuffer** is called, the decoded status information is stored in the DPPAPI. A call to **DisplayStatusBuffer** converts the status information into a text display string which is the same format as the GetStatusString Display string.

4.1.4 Creating Status Information for Amptek Spectrum Files

CreateMCAFileDPPSettings reads the current status and the current configuration stored in the DPPAPI. The status and configuration is stored in string format. This function is used to create Amptek DPP MCA Spectrum file configuration and status information. (See ADMCA Help - Amptek Spectrum File Format, vbDppApi example application.

The current status and configuration are updated during normal operations. It is important to understand **how** the current information is updated.

The ***current configuration*** is updated when a file is read into the DPPAPI or when the current device settings are read into the DPPAPI.

The ***current status*** is updated when one of the following events occurs:

- **GetStatusStruct** is called
- **GetStatusString** is called.
- **GetStatusBuffer** together with **ProcessStatusBuffer** is called.

4.2 DPP Status Information Types

4.2.1 Device Identification

4.2.1.1 Serial Number

The serial number identifies the individual unit. The DPPAPI also uses the serial number to determine if communications are present between the DPP and the DPPAPI.

4.2.1.2 FPGA Version

The Field Programmable Gate Array (FPGA) version identifies the current hardware capabilities. The FPGA is an array of logic gates that is hardware-programmed to fulfill DPP-specified tasks.

4.2.1.3 Firmware Version

The firmware version identifies the low level software that controls the system hardware. The firmware is a set of instructions and data programmed into the EEPROM responsible for controlling the operation of the DPP device.

4.2.2 Spectrum Accumulation Status

4.2.2.1 Accumulation Time

Accumulation Time is the duration (real time) of the present data acquisition interval.

4.2.2.2 Fast Count

Fast Count is the number of counts that have accumulated in the fast channel in the present data acquisition interval.

4.2.2.3 Slow Count

Slow Count is the number of counts that have accumulated in the slow channel in the present

data acquisition interval.

4.2.3 Monitors

4.2.3.1 Board Temperature

Board Temperature is the current temperature measured on the printed circuit board in Celsius degrees.

4.2.3.2 High Voltage (PX4/DP5 only)

High Voltage is the detector high voltage measured in volts.

4.2.3.3 Thermoelectric Cooler (PX4/DP5 only)

Thermoelectric Cooler is the detector temperature measured in Kelvin.

4.2.4 Boot Status

4.2.4.1 Power Button Configured (PX4 only)

The Power Button Configured bit indicates a PX4 has been configured using the power button.

4.2.4.2 Software Configuration Received

The Software Configuration Received bit indicates a valid configuration has been received since power up.

4.2.4.3 Preset Count Expired (PX4/DP5 only)

The Preset Count Expired bit indicates the preset counts set in the selected range have been reached.

4.2.4.4 Status MCA Enabled

The status MCA enabled bit set indicates the DPP device multichannel analyzer is currently acquiring spectrum data. The DPPAPI makes every effort to preserve data integrity during data acquisition. The MCA acquisition mode is set by the PauseDPPData command.

4.2.4.5 Status Device Indicator (0=DP4, 1=PX4)

The status device indicator identifies the type of DPP device connected. Application features is automatically enabled or disabled depending upon the type device attached.

4.2.5 Tuning Status

4.2.5.1 Set Fast Threshold Done

DPP tune fast threshold has successfully completed.

4.2.5.2 Set Input Offset Done (PX4/DP5 only)

PX4 tune input offset has successfully completed.

5 ACQUISITION MODES

A primary function of the DPPAPI is to transmit the spectrum to the user. The interface also controls data acquisition, by starting and stopping the processing and by clearing the spectrum data. There are two acquisition modes **Multichannel Analyzer (MCA)** and **Multichannel Scaler (MCS)**. For a detailed description of MCA and MCS modes see the DP4 and PX4 Users Manuals. All the configuration settings play an important role in acquiring data in MCA and MCS modes.

5.1.1 Multichannel Analyzer (MCA)

The Multichannel Analyzer (MCA) produces an energy spectrum. The digitized peak amplitude of an input shaped pulse creates a count. Counts are accumulated into channels representing a narrow range of energy. The histogram of the counts is the energy spectrum. This is the normal mode of operation.

5.1.2 Acquiring MCA Data

Before starting an acquisition, all configuration settings must be correctly set. The “Acquisition Mode” configuration setting (AcqMode) must be set to MCA. GetDppData returns a snapshot of the data buffer ready for histogram display or for mca plot file storage. The status indicator StatMcaEnabled set to TRUE indicates the acquisition device MCA is enabled. When MCA is paused StatMcaEnabled is false.

5.1.3 Multichannel Scaling (MCS)

The MultiChannel Scaler (MCS) is similar to the MCA except that each data channel corresponds to time instead of amplitude (energy), with the channel boundaries corresponding to start and stop times. Each value in the histogram array represents the total number of counts above the slow threshold during the appropriate time interval.

An MCS data acquisition is started with a given counting time per channel (MCS Timebase). The system records the number of counts in the first interval and places them in the first channel, and so on. The slow (shaped) channel counts are used for MCS. If the peaking time is set to its fastest (0.8 ms), then the MCS counts more closely resemble the input count rate. The MCS mode uses SCA8, so this gives the user the capability of setting the upper and lower level thresholds

5.1.4 MCS Acquisition Setup

The region of interest (energy range) is set using SCA8. Set SCA8 Low Channel to "***0***" and SCA8 High Channel to "***MCA Channels – 1***". *"**0 to MCA Channels – 1***" is the maximum range and is used for normal operation. ***SCA8 MUST BE ENABLED AND SET FOR MCS TO RUN.***

5.1.5 Acquiring MCS Data

Configure the acquisition device and take a sample acquisition in MCA mode. This ensures the acquisition device and the DPPAPI are functioning and ready. The next step is to setup the MCS related parameters. Set the Acquisition Mode to MCS. Select a MCS Timebase. Set the energy range in SCA8 (From 0 to MCA Channels - 1). Enable SCA8 and apply the configuration. Once the MCS parameters are set, the DPP and DPPAPI are ready for MCS data acquisition. Stop the acquisition, if running. Clear the current data. Start the MCS acquisition. The status indicator MCSDone set to TRUE indicates the MCS acquisition has completed. The spectrum data is read and saved the same as in the MCA mode.

5.2 PRESETS

Presets are set as part of the DPP configuration. There are two types of presets, **Preset Time** and **Preset Counts**. The preset time and preset count values are stored in separate variables. Preset counts also requires channel boundary values and an enable which are stored in SCA8.

5.2.1 Preset Modes

There are four possible preset modes:

- None. – No presets are set.
- Preset Time – A preset time is set, but no preset count is set.
- Preset Count – A preset count is set, but no preset time is set.
- Preset Time and Count – Both a preset time and count is set.

5.2.2 None (No Presets)

To turn off all presets, the Preset Time must be set to zero or none and the preset counts must be disabled. To ensure that the preset counts are disabled, set the preset counts to zero, set the lower and upper preset channels to zero and set the SCA8 enable to disabled.

5.2.3 Preset Time

Preset Time stops the spectrum acquisition at a predetermined time.

- If the preset time value is set before the spectrum acquisition has begun then the spectrum acquisition will stop when the *Accumulation Time* reaches the preset time.
- If the preset time value is set after the spectrum acquisition has begun and the *Accumulation Time* is less than the time set, the spectrum acquisition will stop when the *Accumulation Time* reaches the preset time.
- If the preset time value is set after the spectrum acquisition has begun and the *Accumulation Time* is greater than the time set, the spectrum acquisition will not stop until after a new acquisition has begun and the *Accumulation Time* reaches the preset time.

5.2.4 Preset Counts

Preset Count stops the spectrum acquisition at a predetermined number of counts for a selected group of channels. Select the low and upper count boundary channels and enable SCA8. The preset counts are stored in a variable and the selected channels and the SCA8 enable are stored in SCA8.

5.2.5 Preset Time and Preset Counts

When both **Preset Time** and **Preset Counts** are selected the acquisition stops when the first preset is reached.

5.3 Setting the Slow and Fast Thresholds

Setting the fast and slow thresholds is an important setup consideration. Following is a step-by-step guide to correctly setting the fast and slow thresholds.

1) Configure the DPP and the application software for an acquisition.
2) Start an acquisition.
3) Remove the source.
4) Set to delta mode.
5) Setting the slow threshold:
   A) Look for counts on low end.
   B) Place cursor above low end counts.
   C) Open configuration dialog.
   D) Go to MCA tab.
   E) Press slow threshold cursor button.
   F) Press apply.
6) Setting the fast threshold:
   A) Open configuration dialog.
   B) Go to Shaping Tab
   C) Select a fast threshold.
   D) Press apply.
   E) Check Fast Count value in Info Panel.
   F) Repeat Steps C-E until the Input Counts value is approximately 5.
7) The thresholds are now set. You are ready to begin an acquisition.
8) Replace the source.
9) Start an acquisition.