SpcM MATLAB Driver

Driver for all M2i, M3i, M4i, M4x, M2p cards,
digitizerNETBOX products
generatorNETBOX products

Installation, Libraries,
Data sorting, Examples,
Standard mode, FIFO mode

English version **May 24, 2018**

SPECTRUM INSTRUMENTATION GMBH · AHRENSFELDER WEG 13-17 · 22927 GROSSHANSDORF · GERMANY
PHONE: +49 (0)4102-6956-0 · FAX: +49 (0)4102-6956-66 · E-MAIL: info@spec.de · INTERNET: www.spectrum-instrumentation.com

General Information

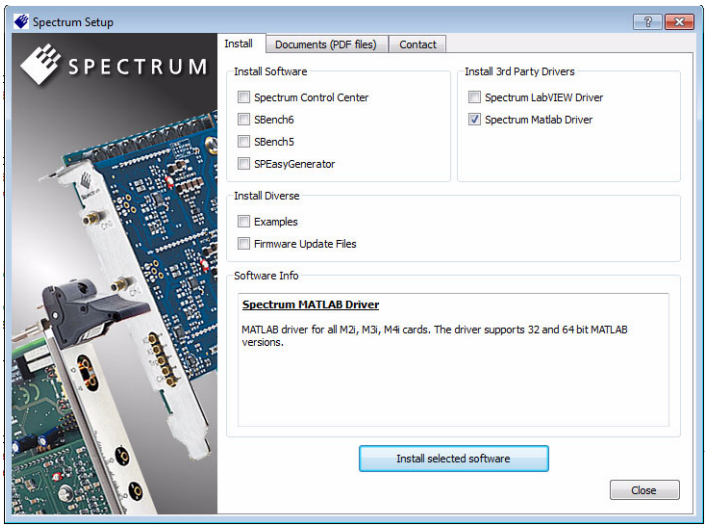
This driver is suitable for all cards of the M2i, M3i, M4i, M4x and M2p series as well as the digitizerNETBOX and generatorNETBOX products from Spectrum. The driver supports all MATLAB versions starting with version 7.7 (R2008b). The Spectrum MATLAB driver supports Windows (32bit and 64bit) and Linux (64bit only) operating systems. Please follow the install instructions to have the drivers properly installed in your system.

Windows Installation

MATLAB Driver Installation

Please follow these steps when installing the MATLAB driver:

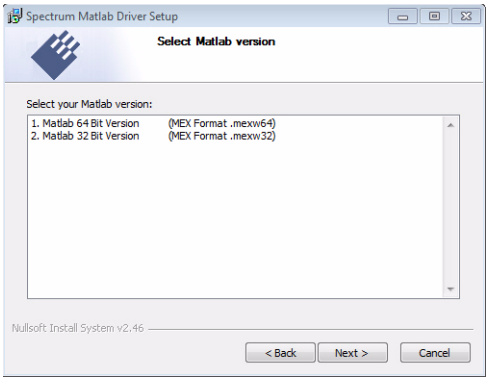
- Install the card(s) into the system as shown in the hardware manual
- Install the standard Windows driver as shown in the hardware manual
- Install the MATLAB driver as explained below



The MATLAB driver is delivered as a self extracting archive. You'll find the MATLAB driver together with your card's current driver on the CD delivered with the card. Please follow the CD menu to „Install" -> „MATLAB Drivers" as shown on the right side.

It is also possible to install the MATLAB driver manually selecting the install file with the windows explorer. Please select the path CD:\Install\win\spcm_drv_matlab_install.exe and execute the exe file. The installer will guide you through the installation routing step by step.

When doing the installation the installer needs to know whether either a 32bit or a 64bit version of MATLAB is installed to select the correct libraries. The dialog shown on right appears. Please select the correct MATLAB version that matches type installed on your system.

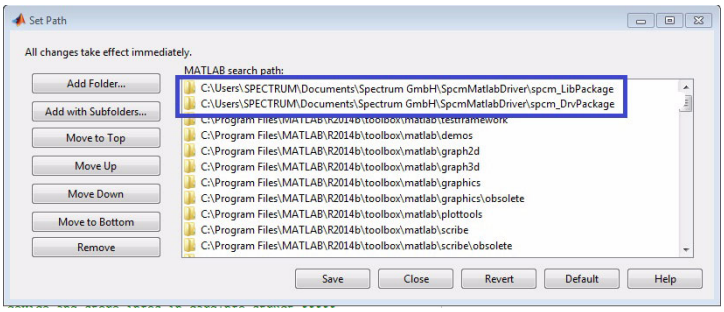


Please note that the installer and the licensing has been updated January 2013. A separate license for the MATLAB driver is no longer needed.

MATLAB environment path setup

The Matlab driver files are installed as default under the current USER directory in the following folder: [USER]\Documents\Spectrum GmbH\SpcmMatlabDriver.

To run the Spectrum MATLAB libraries and examples it is necessary to add the path information to the MATLAB environment. The path setup is found under the MATLAB program surface in the menu „File" -> „Set Path…". The path is added using the „Add Folder" button.



Please add the both directories [INSTALLPATH]\spcm_DrvPackage and [INSTALLPATH]\spcm_LibPackage as shown in the screenshot.

MATLAB Driver Update

As the MATLAB driver also uses the standard Windows drivers as a base, any updates on these drivers will improve the system and any changes are available under MATLAB immediately. Updating the MATLAB driver can simply be done by installation of the new MATLAB driver archive.

Linux Installation

The MATLAB driver is available as an RPM and a DEB archive for easy installation with the package management tool of the distribution.

To install the driver on RPM based distributions like Fedora or openSUSE use the following command as root (assuming that /mnt/CD is the path where the CD drive is mounted):

```
rpm -Uhv /mnt/CD/Install/linux/spcm_matlab_driver-1.0.0-1.x86_64.rpm
```

For DEB based distributions like Debian or Ubuntu use the following command as root:

```
dpkg -i /mnt/CD/Install/linux/spcm-matlab-driver-1.0.0-2.amd64.deb
```

General Information

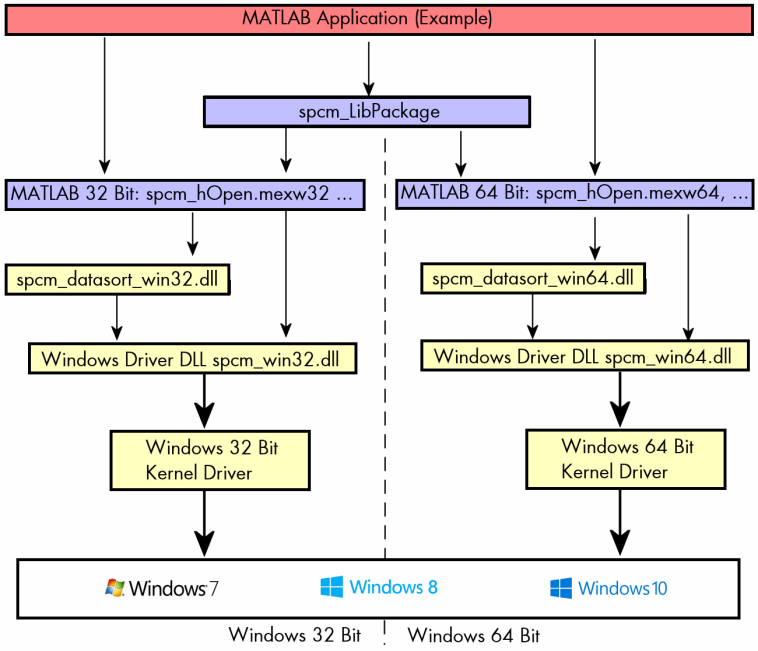
Demo mode

The MATLAB driver runs fine with demo cards installed in your system. Please follow the steps in the hardware manual to see how you insert a simulated demo card into your system. Please keep in mind that the generated data is only simulated. The simulation and calculation of demo data takes more time than just transferring data from hardware to the PC. Therefore the performance of the system is worse when using demo cards.

Driver Structure

The driver itself consists of a couple of MATLAB MEX files like spcm_dwSetParam_i32.mexw64, spcm_hOpen.mexw64, ... (shown in blue), a couple of library m-files in the spcm_LibPackage (also shown in blue) and one additional DLL (depending on the driver type (32bit or 64bit) either spcm_datasort_win32.dll or spcm_datasort_win64.dll) (shown in yellow).

All hardware access is routed through the standard Windows drivers and using the standard Windows kernel driver.
Access of the cards can be solely done by using the direct driver interface consisting of the MATLAB DLLs but using the more comfortable spcm_LibPackage as shown in the examples is the much easier way.



The components of the MATLAB driver:

spcm_win32.dll / spcm_win64.dll

This is the standard Windows driver as it is installed together with the kernel driver when the new hardware is first time recognized in the system.This driver is used by all software that will access the cards.

The driver library is available as 32 bit version (spcm_win32.dll) and 64 bit version (spcm_win64.dll). On 64 bit systems both libraries are installed to allow access for 32 bit programs to the hardware. MATLAB driver uses the 64 bit library. Both drivers can be updated from the internet at any time under www.spectrum-instrumentation.com.

spcm_datasort_win32.dll / spcm_datasort_win64.dll

This is a special helper dll that is used by several Spectrum drivers for third-party products like LabVIEW, MATLAB or VEE. It handles the data access and offers some additional functions to sort data and to re-calculate data to true voltage values. This library also handles the FIFO mode and holds the application data buffer when FIFO mode is used. The DLL is updated with the regular driver updates.

MATLAB MEX files

These MEX files implement the complete interface between MATLAB and the driver. For each driver function there is one MATLAB MEX file that allows the access to this driver function from within MATLAB. The driver functions itself are documented in great detail inside the hardware manual.

spcm_LibPackage

These are additional m-files that form user libraries allowing easy access to certain function groups. Feel free to use these user libraries or to only access the driver directly using the MATLAB MEX files. Our examples make extended use of these user libraries to make them easier to understand and modify.

MATLAB MEX files

This chapter gives you an overview of the available MATLAB MEX files. These files give you the possibility to access all driver functions directly. All registers described in the hardware manual can therefore be accessed and all functions of the hardware can be used.

Throughout our examples we use the more advanced user libraries located in spcm_LibPackage. Please have a look into the next chapter to see a description of these functions.

MATLAB MEX files interfacing with Spectrum driver

These MEX files give you an interface to the standard Spectrum card driver for your operating system allowing any register access. Please see the hardware manual for further information on these functions.

Throughout this manual the MEX files are shown with the simplified file extension *.MEX. Depending on your operating system, these are either *.mexw32 (Windows 32bit), *.mexw64 (Windows 64bit) or *.mexa64 (Linux 64bit).

spcm_hOpen.MEX

```
hDrv = spcm_hOpen (drvName);
```

The function tries to open a card by name. Under windows the drvName can be any valid string with a numeric index at the end. The driver function only checks the number and tries to open the card with that index. This MEX file is similar to the spcm_hOpen function of the standard driver described in the hardware manual.

spcm_vClose.MEX

```
spcm_vOpen (hDrv);
```

Closes the driver handle again. Be sure to include this function at the end of each MATLAB program to allow access to the hardware by other programs. It is not possible to open a card handle by two programs at the same time. This MEX file is similar to the spcm_vClose function of the standard driver described in the hardware manual.

spcm_dwSetParam_i32.MEX

```
dwErrorCode = spcm_dwSetParam_i32 (hDrv, dwRegister, lValue);
```

Programs one of the driver software registers to a 32 bit integer value. The function can be used for all software registers that do not exceed the 32 bit integer range. Please use the _i64 or _i64m function for larger software registers. This MEX file is similar to the spcm_dwSetParam_i32 function of the standard driver described in the hardware manual.

spcm_dwSetParam_i64.MEX

```
dwErrorCode = spcm_dwSetParam_i64 (hDrv, dwRegister, llValue);
```

Programs one of the software registers using a 64 bit signed integer value. The function can be used for all software registers. This MEX file is similar to the spcm_dwSetParam_i64 function of the standard driver described in the hardware manual.

spcm_dwSetParam_i64m.MEX

```
dwErrorCode = spcm_dwSetParam_i64m (hDrv, dwRegister, lValueHigh, dwValueLow);
```

Programs one of the software registers using a 64 bit signed integer value that is split into two parts of 32 bit each. This MEX file is similar to the spcm_dwSetParam_i64m function of the standard driver described in the hardware manual.

spcm_dwGetParam_i32.MEX

```
[dwErrorCode, lValue] = spcm_dwGetParam_i32 (hDrv, dwRegister);
```

Reads out a software register limited to 32 bit integer range. If the register has a larger value the function will return an error. Please use the _i64 or _i64m function to access larger software registers. This MEX file is similar to the spcm_dwGetParam_i32 function of the standard driver described in the hardware manual.

spcm_dwGetParam_i64.MEX

```
[dwErrorCode, llValue] = spcm_dwGetParam_i64 (hDrv, dwRegister);
```

Reads out a software register to a 64 bit integer value. This MEX file is similar to the spcm_dwGetParam_i64 function of the standard driver described in the hardware manual.

spcm_dwGetParam_i64m.MEX

```
[dwErrorCode, lValueHigh, dwValueLow] = spcm_dwGetParam_i64m (hDrv, dwRegister);
```

Reads out a software register to a 64 bit integer value splitted in two 32 bit parts. This MEX file is similar to the spcm_dwGetParam_64m function of the standard driver described in the hardware manual.

spcm_dwGetErrorInfo_i32.MEX

```
[dwErrorCode, lErrorReg, lErrorVal, szErrorText] = spcm_dwGetErrorInfo_i32 (hDrv);
```

Reads out the current error information and returns an error description text that can be printed. If no error has occurred this function just returns zero in all other cases it returns the current error code that is locked inside the driver. This MEX file is similar to the spcm_dwGetErrorInfo_i32 function of the standard driver described in the hardware manual.

MATLAB MEX files interfacing with spcm_datasort_win32.dll/_win64.dll

These MEX files give you an interface to the additional data sorting DLL allowing easy access to the data and maintaining the FIFO mode and the FIFO buffers.

spcm_dwGetData.MEX

```
[dwErrorCode, Dat_Ch0, Dat_Ch1, ..., Dat_Ch15] = spcm_dwGetData (hDrv, dwOffs, dwLen, dwChannels, dwDataType);
```

This MEX file returns channel data sorted in the correct order. The function can be used to get up to 16 analog channels of data. The parameter dwChannels needs to be programmed to the number of analog channels that are passed to the function.

The dwDataType parameter selects whether the function returns the data as 16 bit integer values (dwDataType = 0) or as float values representing the real voltage values at the input (dwDataType = 1). If the re-calculation to float values is used, the driver function takes the programmed input range as well as the programmed user offset into account.

Using dwOffs and dwLen (both given in samples per channel) allows to access certain chunks of the on-board memory. The value dwOffs defines the offset start address and dwLen the length of the data chunk to be read. These two parameters allow the read-out of complete on-board memory which can be up to 4 GBytes without the need of having the same memory size installed in the PC.

If using this function together with the FIFO mode the dwOffs and dwLen parameters are ignored. Instead the programmed notify size of data is returned as it was defined by the spcm_dwSetupFIFOBuffer function.

spcm_dwGetSplittedData.MEX

```
[dwErrorCode, DigitalData, AnalogData] = spcm_dwGetSplittedData (hDrv, dwOffs, dwLen, dwBitsAnalogValue);
```

This function is only to be used if the option -dig (Digital Inputs) is installed on the card and the option is enabled by the MATLAB script before. The function splits analog and digital input data and returns the data in an unsorted and multiplexed way. It is necessary to tell the driver the number of bits that are used for one analog sample (12 or 14). Please have a look at the hardware manual to see how data is multiplexed.

spcm_dwSetData.MEX

```
[dwErrorCode] = spcm_dwSetData (hDrv, dwOffs, dwLen, dwChannels, dwDataType, Dat_Ch0, Dat_Ch1, Dat_Ch2,
Dat_Ch3);
```

This MEX file writes channel data sorted in the correct order. The function can be used to write up to 4 analog channels of data. The parameter dwChannels needs to be programmed to the number of analog channels that are passed to the function.

The dwDataType parameter selects the data type. At the moment only (dwDataType = 0) for int16 values is defined.

Using dwOffs and dwLen (both given in samples per channel) allows to access certain chunks of the on-board memory. These two parameters allow the write of complete on-board memory which can be up to 4 GBytes without the need of having the same memory installed in the PC.

If using this function together with the FIFO mode the dwOffs and dwLen parameters are ignored. Instead the programmed notify size of data is used as it was defined by the spcm_dwSetupFIFOBuffer function.

spcm_dwSetSplittedData.MEX

```
dwErrorCode = spcm_dwSetSplittedData (hDrv, dwOffs, dwLen, DataDigital, DataAnalog, dwBitsAnalogVal);
```

This function is only to be used if the option -dig (Additional Digital Outputs) is installed on the card and the option is enabled by the MATLAB script before.The function writes analog and digital output data in an unsorted and multiplexed way. The function combines analog and digital data and stores them into the samples as described in the hardware manual. It is necessary to tell the driver the number of bits that are used for one analog sample (only 14 bit at the moment). Please have a look at the hardware manual to see how data is multiplexed.

spcm_dwGetABAData.MEX

```
[dwErrorCode, Dat_Ch0, ..., Dat_Ch15] = spcm_dwGetABAData (hDrv, dwLen, dwChannels, dwDataType);
```

Reads out the slow ABA data from the programmed buffer. The parameters are similar to the ones mentioned under the spcm_dwGetData function.

spcm_dwGetTimestampData.MEX

```
[dwErrorCode, Dat_Timestamp] = spcm_dwGetTimestampData (hDrv, dwNoOfTimestamps);
```

The function reads out the timestamps of the card. Timestamps are returned as 64 bit integer values. Before using this function it is necessary to set up a FIFO buffer for the timestamps as timestamps are always stored inside a FIFO buffer. Please have a look at the Multiple Recording example to see the timestamp buffer handling.

spcm_dwGetTimeRefclockData.MEX

```
[dwErrorCode, Dat_Timestamp] = spcm_dwGetTimeRefclockData (hDrv, dwNoOfTimestamps);
```

The function reads out the timestamps of the card and is used if timestamp reference clock mode is activated. Timestamps are returned as two dimensional array with 2 times 32 bit integer values. Index 1 is the timestamp and index 2 is the reference clock count. Before using this function it is necessary to setup a FIFO buffer for the timestamps as timestamps are always stored inside a FIFO buffer. Please have a look at the Multiple Recording Timestamp Reference Clock example to see the timestamp buffer handling.

spcm_dwGetRAWData.MEX

```
[dwErrorCode, RAWData] = spcm_dwGetRAWData (hDrv, dwOffs, dwLen, dwBytesPerSample);
```

The function returns the acquired data in a RAW unsorted and multiplexed way. This is the fastest function to get data from the card to MATLAB. It is necessary to tell the driver the number of bytes that are used for one sample (1 byte for 8 bit cards and 2 bytes for 12, 14 or 16 bit cards). Please have a look at the hardware manual to see how data is multiplexed.

spcm_dwSetRAWData.MEX

```
[dwErrorCode] = spcm_dwSetRAWData (hDrv, dwOffs, dwLen, RAWData, dwBytesPerSample);
```

The function writes the data in a RAW unsorted and multiplexed way. This is the fastest function to get data from MATLAB to the card. It is necessary to tell the driver the number of bytes that are used for one sample (1 byte for 8 bit cards and 2 bytes for 12, 14 or 16 bit cards). Please have a look at the hardware manual to see how data is multiplexed.

spcm_dwSetupFIFOBuffer.MEX

```
dwErrorCode = spcm_dwSetupFIFOBuffer (hDrv, dwBufType, bAllocate, bRead, dwBufferInBytes, dwNotifyInBytes);
```

The MEX file sets up a FIFO buffer for a following FIFO transfer. This is an universal function also allowing to set up FIFO buffers for timestamp or ABA data and for any direction. Please have a look at one of the FIFO examples to see how this function works in detail.

The dwBufType needs to be 0 for sample data, 1 for timestamp data, 2 for slow ABA data. The setup for each buffer type is totally independent.

The bAllocate flag needs to be set to 1 for allocation of the FIFO buffer and 0 to set the FIFO buffer free again. Please be sure that the FIFO transfer has been stopped before setting the buffer free again.

The bRead flags defines the direction for the following FIFO transfer. If set to 1 the transfer is reading data from the card to the PC memory, if set to 0 the transfer is writing data from the PC memory to the card.

The value dwBufferInBytes defines the length of the buffer to allocate in bytes.

The value dwNotifyInBytes defines the notify size in bytes. This is the block size that can be read out from the FIFO buffer using the GetData and the GetRAWData functions.

spcm_dwGetSegmentStatisticData.MEX (M4i/M4x cards or digitizerNETBOX only)

```
[dwErrorCode, Dat_Ch0, …, DatCh3] = spcm_dwGetSegmentStatisticData (hDrv, dwNumOfSegments, dwChannels);
```

This function returns the statistic data for up to 4 channels. The parameter dwNumOfSegments needs to be programmed to the number of acquired segments. The parameter dwChannels needs to be programmed to the number of programmed analog channels. The statistic data is a 64 bit array that contains the statistic data in the following order:

Dat_Ch0[1] = Average value for segment 1
Dat_Ch0[2] = Minimum value for segment 1
Dat_Ch0[3] = Maximum value for segment 1
Dat_Ch0[4] = Minimum position for segment 1
Dat_Ch0[5] = Maximum position for segment 1
Dat_Ch0[6] = Timestamp value for segment 1

Dat_Ch0[7] = Average value for segment 2
Dat_Ch0[8] = Minimum value for segment 2
Dat_Ch0[9] = Maximum value for segment 2
Dat_Ch0[10] = Minimum position for segment 2
Dat_Ch0[11] = Maximum position for segment 2
Dat_Ch0[12] = Timestamp value for segment 2

Dat_Ch0[13] = Average value for segment 3
…

This requires the M4i/M4x signal processing firmware option „Block Statistics/Peak Detect“ to be installed on the card. If you do not have this option installed, you can also purchase and install it later by a firmware upgrade.

MATLAB LibPackage

The library package offers a couple of functions that are used to easily program the driver. The functions group driver calls to make the programming easier and better understandable. All library functions are included in source code in m-language to allow detailed look or the use for own projects.

It is not recommended to modify the library functions itself as they're updated with a new MATLAB driver install. Instead it is recommended to make a copy of this function and modify this copy.

The user library functions are provided „as is" to help programming and to make the examples better readable. They are not intended to be a complete driver and therefore some functions can only be accessed by using the direct driver calls described in the last chapter.

General Information

All functions of the LibPackage use a common card information structure that holds the driver handle, some card information as well as the current error information. The card info structure is passed through all functions allowing an easy internal error check and providing additional information for the functions.

The spcm_LibPackage library doesn't cover some special modes of single cards. It is also possible that some functionality is added to the standard driver later on. As changing the interface would mean that none of the examples or customer applications would work any more after an update, these interfaces are not planned to be updated in the future. Any further or later added content can be directly accessed using the driver functions that are located in the MATLAB MEX files.

Acquisition Example

The following example just initializes card zero, reads out some information, prints them, does a simple card setup, starts the card and reads out all data of the acquired channel.

```
[success, cardInfo] = spcMInitCardByIdx (0);

if (success == true)
    fprintf (spcMPrintCardInfo (cardInfo));
else
    spcMErrorMessageStdOut (cardInfo, 'Error: Could not open card\n', true);
    return;
end

% ***** do card setup *****
[success, cardInfo] = spcMSetupModeRecStdSingle (cardInfo, 0, 1, 16 * 1024, 8 * 1024);
[success, cardInfo] = spcMSetupClockPLL (cardInfo, 10000000, 0);
[success, cardInfo] = spcMSetupTrigSoftware (cardInfo, 0);  % trigger output : enable = 1, disable = 0

% ----- set command start and wait ready -----
errorCode = spcm_dwSetParam_i32 (cardInfo.hDrv, 100, 12);

% ----- read the one channel that was acquired as 16 bit integer values -----
[errorCode, Dat_Ch0] = spcm_dwGetData (cardInfo.hDrv, 0, cardInfo.setMemsize, cardInfo.setChannels, 0);

spcMCloseCard (cardInfo);
```

Error handling

The error information is put into the card information structure and can be read out at the end of all setup. It is normally not necessary to check the error code in between as all driver functions are error safe.

```
[success, cardInfo] = spcMSetupModeRecStdSingle (cardInfo, 0, 1, 16 * 1024, 8 * 1024);
[success, cardInfo] = spcMSetupClockPLL (cardInfo, 10000000, 0);
[success, cardInfo] = spcMSetupTrigSoftware (cardInfo, 0);  % trigger output : enable = 1, disable = 0
if (cardInfo.setError)
    fprintf (cardInfo.errorText);
```

Example of LibPackage Function

This example shows how the LibPackage function looks internally. As one can see it's simple a multiple call of the common driver functions paired with some error checking.

```
function [success, cardInfo] = spcMSetupModeRecStdSingle (cardInfo, chEnableH, chEnableL, memSamples, postSam-
ples)

    % ----- setup the standard single acquisition mode
    error =         spcm_dwSetParam_i32  (cardInfo.hDrv, 9500,  1);
    error = error + spcm_dwSetParam_i64m (cardInfo.hDrv, 11000, chEnableH, chEnableL);
    error = error + spcm_dwSetParam_i64  (cardInfo.hDrv, 10000, memSamples);
    error = error + spcm_dwSetParam_i64  (cardInfo.hDrv, 10100, postSamples);

    % ----- store some information in the structure for later use-----
    cardInfo.setMemsize = memSamples;
    cardInfo.setChEnableMapH = chEnableH;
    cardInfo.setChEnableMapL = chEnableL;
    [errorCode, cardInfo.setChannels] = spcm_dwGetParam_i32 (cardInfo.hDrv, 11001);  % 11001 = SPC_CHCOUNT

    [success, cardInfo] = spcMCheckSetError (error, cardInfo);
```

LibPackage Functions

This chapter gives you an overview of the available LibPackage functions that have been implemented at the time of printing the manual. As these library functions are extended from time to time, please have a look at the directory [INSTALLPATH]\Spcm_LibPackage to find additional functions that are not described in the manual so far.

If you find any essential library functions missing please contact our support team by email and suggest new functions.

Using Mnemonics for registers and values

To simplify the access to the Spectrum driver registers and error codes there is the possibility to create two MATLAB maps that contain the register/ErrorCode names as map-key and the register/ErrorCode numerical number as map-values.
So as an example the register „SPC_M2CMD" has the number „100" and the „M2CMD_CARD_START" has the number „4". You can call the driver with the numerical values:

```
spcm_dwSetParam_i32 (cardInfo.hDrv, 100, 4);
```

or you can use the map to make the code more readable:

```
spcm_dwSetParam_i32 (cradInfo.hDrv, mRegs('SPC_M2CMD'), mRegs('M2CMD_CARD_START'));
```

To initialize the maps there are two functions in the spcm_LibPackage folder::

```
mRegs = spcMCreateErrorMap.m
mErrors = spcMCreateRegMap.m
```

Initialisation functions

These functions are needed for all kind of cards to open and close card handles, to do error handling and message display and to print general information on the card.

spcMInitCardByIdx.m

```
[success, cardInfo] = spcMInitCardByIdx (cardIdx);
```

The function opens the driver with the given index, reads out card information and returns a filled card information structure. The success flag indicates whether the open function was successful or not.

spcMInitDevice.m

```
[success, cardInfo] = spcMInitDevice  (deviceString);
```

This function opens and initializes an installed card or a digitizerNETBOX by using a valid device name or VISA string as a parameter.

- Card: deviceString = '/dev/spcm0'
- digitizerNETBOX: deviceString = 'TCPIP::[IP ADDRESS]::inst0::INSTR'

spcMPrintCardInfo.m

```
[cardInfoText] = spcMPrintCardInfo (cardInfo);
```

Prints some of the card information to a string that can be used for further display. The printed card information contains the card type, the serial number, the installed memory in bytes, the maximum sampling rate, the number of channels and the detailed driver information including kernel and library driver version.

spcMCheckSetError.m

```
[success, cardInfo] = spcMCheckSetError (error, cardInfo);
```

This function is used internally to check for an error code and to read out the current error information. It is called after each block of setup to have the current error information in the card information structure.

spcMErrorMessageStdOut.m

```
spcMErrorMessageStdOut (cardInfo, message, printCardErr);
```

Prints the error message to std out and ends the driver if it's active. The program can be left with this function showing a final error message and closing the driver.

spcMCloseCard.m

```
spcMCloseCard (cardInfo);
```

Closes the card handle, should be called on every end of the program.

Clock Setup Functions

These functions allow to setup any of the clock modes the card can handle. It is only possible to use one of the clock modes at a time. Please be sure to select one of the clock modes for your program and to program a clock that is valid for your card type.

spcMSetupClockPLL.m

```
[success, cardInfo] = spcMSetupClockPLL (cardInfo, samplerate, clockOut);
```

Sets the clock to internal clocking using the PLL. The sampling rate is set to the given „samplerate“ parameter and the optional clock output can be enabled using the „clockOut“ flag. After the call the card information structure contains the sampling rate set by the driver which is the nearest matching sampling rate that is possible.

spcMSetupClockExternal.m

```
[success, cardInfo] = spcMSetupClockExternal (cardInfo, extRange, clockTerm, divider)
```

Sets the clock to external clock using either directly the external clock or using the external clock with an additional divider. Please specify the matching clock range („extRange“) as described in the hardware manual. The „clockTerm“ parameter defines whether the clock input termination should be set or not. The optional „divider“ parameter defines the clock divider. Set this to 1 if the clock is directly used.

spcMSetupClockQuartz.m

```
[success, cardInfo] = spcMSetupClockQuartz (cardInfo, samplerate, clockOut)
```

Sets the clock to internal clocking using the Quartz and a divider. The sampling rate is set to the given „samplerate“ parameter and the optional clock output can be enabled using the „clockOut“ flag. After the call the card information structure contains the sampling rate set by the driver which is the nearest matching sampling rate that is possible.

spcMSetupClockRefClock.m

```
[success, cardInfo] = spcMSetupClockRefClock (cardInfo, refClock, samplerate, clockTerm)
```

The internal feed in clock is used as a reference clock for the PLL. For calculation of the correct sampling rate („samplerate“) the driver needs to know the frequency of the fed in reference clock („refClock“). Optional one can terminate the clock input.

Input related functions

These functions are only used with acquisition cards or I/O cards. If using an I/O card the card must be switched to a recording mode to allow input channel setup.

spcMSetupAnalogInputChannel.m

```
[success, cardInfo] = spcMSetupAnalogInputChannel (cardInfo, channel, inputRange, term, inputOffset,
diffInput);
```

Does a setup for one analog input channel which must be specified in the „channel“ parameter. Channel numbering starts from 0!

The „inputRange“ parameter defines the input range to choose for this channel, the „term“ parameter switches the programmable input termination, the „inputOffset“ programs the offset of the input and the „diffInput“ switches between single-ended and differential inputs. As this is an universal function not all cards support all of these parameters. If the card type does not support all of the features these settings will simply be ignored.

spcMSetupAnalogPathInputCh.m (analog cards with different input paths)

```
[success, cardInfo] = spcMSetupAnalogPathInputCh (cardInfo, channel, path, inputRange, term, ACCoupning,
BWLimit, diffInput);
```

Does a setup for one analog input channel which must be specified in the „channel“ parameter. Channel numbering starts from 0! This function is used for all analog data acquisition cards using input path information (like M3i and M3i-Express series)

The „path“ parameter defines the path to be selected. The „inputRange“ parameter defines the input range to choose for this channel, the „term“ parameter switches the programmable input termination, the „ACCoupling“ switches between DC (0) and AC (1) coupling, the „BWLimit“ parameter activates the bandwidth limiting filter (1) and the „diffInput“ switches between single-ended (0) and differential inputs (1). As this is an universal function not all cards support all of these parameters. If the card type does not support all of the features these settings will simply be ignored.

spcMSetupDigitalInput.m

```
[success, cardInfo] = spcMSetupDigitalInput (cardInfo, group, term);
```

Does a setup for one digital input channel group which must be specified in the „group“ parameter. Group numbering starts from 0! Each digital input or i/o card can have different setup for a group of channels. Please refer to the hardware manual to see the input setup grouping.

The „term“ parameter switches the programmable input termination for the specified digital input channel group. As this is an universal function not all cards support all of these parameters. If the card type does not support all of the features these settings will simply be ignored.

spcMDemuxDigitalData.m

```
[digData] = spcMDemuxDigitalData (RAWData, RAWDataLen, channels);
```

Splits up (demultiplexes) digital data from a RAW data file containing standard integer values to a matrix containing single values for each digital channel. Be aware that the resulting array has 8 times the size of the RAW data file as each 1 bit sample is stored in one 8 bit integer value. The „RAWData“ gets the data as directly read from the card. The „DataLen“ and „channels“ info describe how the data is organized.

Output related functions

These functions are only used with generation cards or I/O cards. If using an I/O card the card must be switched to a generation mode to allow input channel setup.

spcMSetupAnalogOutputChannel.m

```
[success, cardInfo] = spcMSetupAnalogOutputChannel (cardInfo, channel, amplitude, outputOffset, filter,
stopMode, doubleOut, differential);
```

Does a setup for one analog output channel which must be specified in the „channel“ parameter. Channel numbering start from 0! the „amplitude“ parameter defines the output amplitude in mV, the „outputOffset“ defines the output offset of the channel, the filter defines the code

of the output filter to be used. „stopMode", „doubleOut" and „differential" are special functions of the generator card and are defined in the manual in detail. As this is an universal function not all cards support all of these parameters. If the card type does not support all of the features these settings will simply be ignored.

spcMSetupDigitalOutput.m

```
[success, cardInfo] = spcMSetupDigitalOutput (cardInfo, group, stopMode, lowLevel, highLevel, diffMode);
```

Does a setup for one digital output channel group which must be specified in the „group" parameter. Channel numbering start from 0! Each digital input or i/o card can have different setup for a group of channels. Please refer to the hardware manual to see the input setup grouping.

The „stopMode" defines the behaviour between outputs, details are described in the hardware manual. The „lowLevel" and „highLevel" define the output low and high levels in mV. The „diffMode" flag enables a special differential output mode which is described in more detail in the hardware manual. As this is an universal function not all cards support all of these parameters. If the card type does not support all of the features these settings will simply be ignored.

Acquisition Mode Setup

These functions define the acquisition mode for the next recording. The functions are only used with acquisition or I/O cards. It is only possible to use one of the acquisition modes at a time. Please be sure to select one of the modes for your program. Some of the acquisition modes like RecStdMulti require an option to be installed on your card. If that option isn't installed you will get an error message.

spcMSetupModeRecStdSingle.m

```
[success, cardInfo] = spcMSetupModeRecStdSingle (cardInfo, chEnableH, chEnableL, memSamples, postSamples);
```

Sets the recording mode to standard single and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples" and „postSamples" gives memory size and post trigger in samples per channel.

spcMSetupModeRecStdMulti.m

```
[success, cardInfo] = spcMSetupModeRecStdMulti (cardInfo, chEnableH, chEnableL, memSamples, segmentSize,
postSamples);
```

Sets the recording mode to standard Multiple Recording and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples", „segmentSize" and „postSamples" gives memory size, segment size and posttrigger in samples per channel.

spcMSetupModeRecStdGate.m

```
[success, cardInfo] = spcMSetupModeRecStdGate (cardInfo, chEnableH, chEnableL, memSamples, preSamples,
postSamples);
```

Sets the recording mode to standard Gated Sampling and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples", „preSamples" and „postSamples" gives memory size, pre-samples (before gate start) and post-samples (after gate end) in samples per channel.

spcMSetupModeRecStdABA.m

```
[success, cardInfo] = spcMSetupModeRecStdABA (cardInfo, chEnableH, chEnableL, memSamples, segmentSize, postSam-
ples, ABADivider);
```

Sets the recording mode to standard ABA mode and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples", „segmentSize" and „postSamples" gives memory size, segment size and posstrigger in samples per channel. The „ABADivider" value defines a divider value for the slow „A" sampling rate. The „A" data will be samples with [sampling rate] / [ABADivider].

spcMSetupModeRecStdAverage.m (M4i, M4x cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupModeRecStdAverage (cardInfo, chEnableH, chEnableL, memSize, segmentSize, post-
trigger, averagesPerSegment);
```

Sets the recording mode to standard Averaging and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSize", „SegmentSize" and „posttrigger" gives memory size, segment size and posttrigger in samples per channel. The „averagesPerSegment" parameter sets the number of accumulated cycles for each segment.

This requires the M4i/M4x signal processing firmware option „Block Average“ to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

spcMSetupModeRecStdAverage16Bit.m (M4i.22xx, M4x.22xx cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupModeRecStdAverage16Bit (cardInfo, chEnableH, chEnableL, memSize, segmentSize,
posttrigger, averagesPerSegment);
```

This function uses the 16 bit average mode of the M4i/M4x.22xx series cards and based upon digitizerNETBOX products. The parameters are the same as the „spcMSetupModeRecStdAverage.m“ function. For more information about the average 16 bit mode please take a look at the M4i/M4x.22xx hardware manual.

This requires the M4i/M4x signal processing firmware option „Block Average“ to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

spcMSetupModeRecStdSegmStat.m (M4i, M4x cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupModeRecStdSegmStat.m (cardInfo, chEnableH, chEnableL, memSize, segmentSize,
posttrigger);
```

Sets the recording mode to standard Block Statistics and programs all necessary settings. The „chEnableH“ (upper 32 channels) and „chEnableL“ (lower 32 channels) parameter must form together a valid channel enable mask while „memSize“, „SegmentSize“ and „posttrigger“ gives memory size, segment size and posttrigger in samples per channel.

This requires the M4i/M4x signal processing firmware option „Block Statistics/Peak Detect“ to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

spcMSetupModeRecFIFOSingle.m

```
[success, cardInfo] = spcMSetupModeRecFIFOSingle (cardInfo, chEnableH, chEnableL, preSamples, blockToRec,
loopToRec);
```

Sets the recording mode to FIFO single and programs all necessary settings. The „chEnableH“ (upper 32 channels) and „chEnableL“ (lower 32 channels) parameter must form together a valid channel enable mask while „preSamples“, „blocktoRec“ and „loopToRec“ gives pretrigger samples and the block size and loops to record. „loopToRec“ can be left zero if the acquisition should run continuously without any stop.

spcMSetupModeRecFIFOMulti.m

```
[success, cardInfo] = spcMSetupModeRecFIFOMulti (cardInfo, chEnableH, chEnableL, segmentSize, postSamples,
segmentsToRec);
```

Sets the recording mode to FIFO Multiple Recording and programs all necessary settings. The „chEnableH“ (upper 32 channels) and „chEnableL“ (lower 32 channels) parameter must form together a valid channel enable mask while „segmentSize“ and „postSamples“ gives segment size and posstrigger in samples per channel. „segmentsToRec“ defines a number of segments after which the recording should stop automatically. If left zero the acquisition will run until stopped by user.

spcMSetupModeRecFIFOGate.m

```
[success, cardInfo] = spcMSetupModeRecFIFOGate (cardInfo, chEnableH, chEnableL, preSamples, postSamples,
gatesToRec);
```

Sets the recording mode to FIFO Gated Sampling and programs all necessary settings. The „chEnableH“ (upper 32 channels) and „chEnableL“ (lower 32 channels) parameter must form together a valid channel enable mask while „preSamples“ and „postSamples“ gives presamples (before gate start) and post-samples (after gate end) in samples per channel. „gatesToRec“ defines a number of gate-segments after which the recording should stop automatically. If left zero the acquisition will run until stopped by user.

spcMSetupModeRecFIFOABA.m

```
[success, cardInfo] = spcMSetupModeRecFIFOABA (cardInfo, chEnableH, chEnableL, segmentSize, postSamples,
ABADivider, segmentsToRec);
```

Sets the recording mode to FIFO ABA mode and programs all necessary settings. The „chEnableH“ (upper 32 channels) and „chEnableL“ (lower 32 channels) parameter must form together a valid channel enable mask while „segmentSize“ and „postSamples“ gives segment size and posttrigger in samples per channel. The „ABADivider“ value defines a divider value for the slow „A“ sampling rate. The „A“ data will

be samples with [sampling rate] / [ABADivider]. „segmentsToRec" defines a number of segments after which the recording should stop automatically. If left zero the acquisition will run until stopped by user.

spcMSetupRecFifoAverage.m (M4i, M4x cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupRecFifoAverage (cardInfo, chEnableH, chEnableL, segmentSize, posttrigger,
averagesPerSegment, loops);
```

Sets the recording mode to FIFO Averaging and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „segmentSize" and „posttrigger" gives segment size and posttrigger in samples per channel. The „averagesPerSegment" parameter sets the number of accumulated cycles for each segment and „loops" defines the number of segments after which the recording should stop automatically. If „loops" is left zero the acquisition will run until stopped by user.

This requires the M4i/M4x signal processing firmware option „Block Average" to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

spcMSetupRecFifoAverage16Bit.m (M4i.22xx, M4x.22xx cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupRecFifoAverage16Bit (cardInfo, chEnableH, chEnableL, segmentSize, posttrigger,
averagesPerSegment, loops);
```

This function uses the 16 bit Fifo average mode of the M4i.22xx series. The parameters are the same as the „spcMSetupModeFifoStdAverage.m" function. For more information about the average 16 bit mode please take a look at the M4i.22xx hardware manual.

This requires the M4i/M4x signal processing firmware option „Block Average" to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

spcMSetupModeRecFifoSegmStat.m (M4i, M4x cards and digitizerNETBOX only)

```
[success, cardInfo] = spcMSetupModeRecFifoSegmStat.m (cardInfo, chEnableH, chEnableL, segmentSize, posttrigger,
loops);
```

Sets the recording mode to standard Block Statistics and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „SegmentSize" and „posttrigger" gives memory size, segment size and posttrigger in samples per channel. The „loops" parameter defines the number of segments after which the recording should stop automatically. If „loops" is left zero the acquisition will run until stopped by user.

This requires the M4i/M4x signal processing firmware option „Block Statistics/Peak Detect" to be installed on the card. If you do not have this option installed, you can also purchased and install it later by a firmware upgrade.

Generation Mode Setup

These functions define the generation mode for the next run. The functions are only used with generator or I/O cards. It is only possible to use one of the generation modes at a time. Please be sure to select one of the modes for your program.

spcMSetupModeRepStdSingle.m

```
[success, cardInfo] = spcMSetupModeRepStdSingle (cardInfo, chEnableH, chEnableL, memSamples);
```

Sets the generation mode to standard single and programs all necessary settings. The memory is replayed after receiving the trigger event once. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples" gives memory size per channel.

spcMSetupModeRepStdLoops.m

```
[success, cardInfo] = spcMSetupModeRepStdLoops (cardInfo, chEnableH, chEnableL, memSamples, loops);
```

Sets the generation mode to standard continuous mode and programs all necessary settings. The memory is replayed in a loop after receiving a trigger event. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples" gives memory size per channel. The „loop" parameter programs the number of repetitions after receiving the trigger event. A zero will program the card to continue looping until it is stopped.

spcMSetupModeRepStdSingleRestart.m

```
[success, cardInfo] = spcMSetupModeRepStdSingleRestart (cardInfo, chEnableH, chEnableL, memSamples, loops);
```

Sets the generation mode to single restart mode and programs all necessary settings. The memory is replayed once on each received trigger event. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask while „memSamples" gives memory size per channel. The „loop" parameter programs the number of single shots to be done. A zero will program the card to continue singleshots on trigger until it is stopped.

spcMSetupModeRepSequence.m

```
[success, cardInfo] = SpcMSetupModeRepSequence (cardInfo, chEnableH, chEnableL, numSegments, startSegment);
```

Sets the generation mode to standard Sequence mode and programs all necessary settings. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask. The „numSegments" parameter programs the number of segments the on-board memory should be divided into and „startSegment" defines which of all defined steps in the sequence memory will be used first directly after the card start.

TThis requires the feature „Sequence Replay Mode" to be installed on the card.
For further details on the Sequence Replay Mode please refer to the card's hardware manual.

spcMSetupModeRepFIFOSingle.m

```
[success, cardInfo] = spcMSetupModeRepFIFOSingle (cardInfo, chEnableH, chEnableL, blockToRep, loopToRep);
```

Sets the generation mode to single FIFO mode and programs all necessary settings. After receiving the trigger event the card starts to output data continuously in FIFO mode and loads data continuously from PC memory. The „chEnableH" (upper 32 channels) and „chEnableL" (lower 32 channels) parameter must form together a valid channel enable mask. „blocktoRep" and „loopToRep" gives the block size and loops to replay. „loopToRep" can be left zero if the generation should run continuously without any stop.

Timestamp Setup

The timestamp functions are only available on acquisition cards or I/O cards and need the option timestamp to be installed on the card.

spcMSetupTimestamp.m

```
[success, cardInfo] = spcMSetupTimestamp (cardInfo, mode, refTimeoutMS);
```

Sets up the timestamp mode and performs a synchronization with reference clock if that mode is activated. Checks for BASEXIO option if one wants to use reference clock mode. The different available modes are explained in the hardware manual.

Trigger Setup

These functions define the trigger mode for the next card run. Depending on the card type not all of these functions are available with every card. The MSetupTrigChannel for example is only available for acquisition cards.

spcMSetupTrigSoftware.m

```
[success, cardInfo] = spcMSetupTrigSoftware (cardInfo, trigOut);
```

Programs the trigger event to software. Card is started immediately after the start command without waiting for any external signals.

spcMSetupTrigExternal.m (external TTL trigger)

```
[success, cardInfo] = spcMSetupTrigExternal (cardInfo, extMode, trigTerm, pulsewidth, singleSrc, extLine);
```

Programs the external trigger mode. The „extMode" value must contain a valid external mode as described in the hardware manual. The „trigTerm" flag defines whether to terminate the trigger input while the „pulsewidth" value programs the pulsewidth for any external trigger mode that uses a pulsewidth counter. If the „singleSrc" flag is set the external trigger is the only trigger source and all other trigger sources are disabled. When not programming the „singleSrc" flag it is necessary to program the OR and AND mask manually allowing to combine several trigger sources. The „extLine" parameter allows to select different possible TTL sources, if available on the particular card.

spcMSetupTrigExternalLevel.m (analog external trigger)

```
[success, cardInfo] = spcMSetupTrigExternalLevel (cardInfo, extMode, level0, level1, trigTerm, ACCoupling,
pulsewidth, singleSrc);
```

Programs the external trigger mode. The „extMode" value must contain a valid external mode as described in the hardware manual. „Level0" and „Level1" program the corresponding trigger levels in mV. The „trigTerm" flag defines whether to terminate the trigger input while the „ACCoupling" flag swicthes between DC (0) and AC (1) coupling. If supported by the hardware the „pulsewidth" value programs the pulsewidth for any external trigger mode that uses a pulsewidth counter. If the „singleSrc" flag is set the external trigger is the only trigger source and all other trigger sources are disabled. When not programming the „singleSrc" flag it is necessary to program the OR and AND mask manually allowing to combine several trigger sources.

spcMSetupTrigChannel.m

```
[success, cardInfo] = spcMSetupTrigChannel (cardInfo, channel, trigMode, trigLevel0, trigLevel1, pulsewidth,
trigOut, singleSrc)
```

Programs the channel trigger mode of one channel. The „channel" parameter defines which channel trigger to be programmed. Channel counting starts with 0. The „trigMode" value must contain a valid channel trigger mode as described in the hardware manual. The „trigLevel0", „trigLevel1" and „pulsewidth" values define the setup of the trigger mode depending on the additional parameters the selected trigger mode needs. If the „singleSrc" flag is set the inhere defined channel trigger is the only trigger source and all other trigger sources are disabled. When not programming the „singleSrc" flag it is necessary to program the OR and AND mask manually allowing to combine several trigger sources.

spcMSetupTrigMask.m

```
[success, cardInfo] = spcMSetupTrigMask (cardInfo, channelOrMask0, channelOrMask1, channelAndMask0,
channelAndMask1, trigOrMask, trigAndMask)
```

Defines the trigger masks for more complex triggers. Please note that the external and channel triggers need to be defined with the „singleSrc" = 0 flag to allow combined triggers. If using a combined trigger one needs to define at least the trigger mask and one trigger source. The „channelOrMask" and the „channelAndMask" parameters define the masks, each bit corresponds to the related trigger channel. The main „trigOrMask" and „trigAndMask" define the masks for external trigger and additional trigger.

The following example shows you how to program a trigger conjunction of channel 0 rising edge OR channel 1 rising edge:

```
[success, cardInfo] = spcMSetupTrigChannel (cardInfo, 0, 1, 0, 0, 0, 0, 0);
[success, cardInfo] = spcMSetupTrigChannel (cardInfo, 1, 1, 0, 0, 0, 0, 0);
[success, cardInfo] = spcMSetupTrigMask (cardInfo, 3, 0, 0, 0, 0, 0);
```

Miscellaneous Settings

These settings do not fit under one of the other topics and may be card type dependant.

spcMSetupMultiIO.m (M3i, M4i/M4x and M2p only)

```
[success, cardInfo] = spcMSetupMultiIO (cardInfo, modeX0, modeX1)
```

This library function defines the behavior ot the two multi purpose I/O lines of the M3i series, the three lines of the M4i/M4x cards or the four lines of the M2p cards or related digitizerNETBOX/generatorNETBOX products. Please refer to the hardware manual to see which modes are available for your card.

spcMSetupSequenceStep.m

```
[success, cardInfo] = spcMSetupSequenceStep (cardInfo, step, nextStep, segment, loops, condition);
```

This functions defines the order of the sequence steps. The „step" paramter is the index of the current step to programm. It should start with „0" and should be increased by „1" with every following step. The „nextStep" defines the step index of the next step that should be executed after the currect step has finished. The „segment" parameter sets the segment index to replay and the „loops" parameter set the number of replay iterations. The „condition" parameter sets the condition when to switch to the next step or to end the complete sequence.

Values for „condition":

- 0 –> End loop always
- 1 –> End loop on trigger
- 2 –> End sequence

TThis requires the feature „Sequence Replay Mode" to be installed on the card. For further details on the Sequence Replay Mode please refer to the card's hardware manual.

MATLAB Examples

This chapter gives you a brief overview over the available MATLAB examples. As the MATLAB driver is an universal driver for all Spectrum cards there are examples for all kinds of cards and features explained here.

General Information

All examples are provided „as is" without any warranty. Feel free to modify these examples in any kind and to use them as a base for your own programs.

All examples only show how to program the card and how to set up a basic setup. Data is retrieved from the card (or written to the card) with only a very few of MATLAB functions.

Examples

rec_std_single.m

Example for all SpcMDrv based (M2i, M3i, M4i/M4x and M2p) acquisition cards and digitizerNETBOX products. Shows standard data acquisition using single mode (one shot). All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card.

rec_std_multi.m

Example for all SpcMDrv based (M2i, M3i, M4i/M4x and M2p) acquisition cards and digitizerNETBOX products with the feature Multiple Recording installed. Shows Multiple Recording data acquisition using single mode (one shot). If the timestamp option is installed the corresponding timestamp values are also read out and displayed on the console. All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card.

m2i_m3i_rec_std_multi_tsrefclock.m

Example for all M2i and M3i acquisition cards and digitizerNETBOX with the feature Multiple Recording, Timestamp and the option BaseXIO (M2i and M3i) installed. Shows Multiple Recording data acquisition using standard mode with timestamp reference clock mode. The timestamp and reference clock values are also read out and displayed on the console. All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card.

rec_std_gate.m

Example for all SpcMDrv based (M2i, M4i and M4x) acquisition cards and digitizerNETBOX products with the feature Gated Sampling installed. Shows Gated Sampling data acquisition using single mode (one shot). If the timestamp option is installed the corresponding timestamp values are also read out and displayed on the console. All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card.

rec_fifo_single.m

Example for all SpcMDrv based (M2i, M3i, M4i and M4x) acquisition cards and digitizerNETBOX products. All channels are enabled for the acquisition. Does a continuous FIFO transfer and writes a part of that data to binary files. Afterwards the first four channels will be read out from the files and displayed using the MATLAB plot window.

rec_fifo_multi.m

Example for all SpcMDrv based (M2i, M3i, M4i and M4x) acquisition cards and digitizerNETBOX products with the feature Multiple Recording installed. All channels are enabled for the acquisition. Does a continuous FIFO transfer with Multiple Recording and writes a part of that data to binary files. Afterwards the first four channels will be read out from the files and displayed using the MATLAB plot window.

m2i_m3i_rec_fifo_multi_ts_poll.m

Example for all SpcMDrv based (M2i, M3i) acquisition cards and digitizerNETBOX products with the feature Multiple Recording and Timestamp installed. All channels are enabled for the acquisition. Does a continuous FIFO transfer with Multiple Recording and writes a part of that data to binary files. Afterwards the first four channels will be read out from the files and displayed using the MATLAB plot window. The timestamps are also polled from the and stored to an separate binary file.

m2i_m3i_rec_fifo_ABA_ts_poll.m

Example for all M2i or M3i acquisition cards and digitizerNETBOX products with the feature ABA and Timestamp installed. Similar to the „m2i_m3i_rec_fifo_multi_ts_poll.m" example, but not using Multiple Recording but ABA mode instead.

rec_fifo_gate.m

Example for all SpcMDrv based (M2i, M4i and M4x) acquisition cards and digitizerNETBOX products with the feature Gated Sampling installed. All channels are enabled for the acquisition. Does a continuous FIFO transfer with Gated Sampling and writes a part of that data to binary files. Afterwards the first four channels will be read out from the files and displayed using the MATLAB plot window.

m2i_m3i_rec_std_aba.m

Example for all M2i and M3i acquisition cards and digitizerNETBOX products with the feature ABA installed. Shows ABA data acquisition using single mode (one shot). The corresponding timestamp values are also read out and displayed on the console. All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card. The example shows how to read out the slow A-data as well as how to read out the fast B-data.

rep_std_single.m

Example for all SpcMDrv based (M2i, M4i and M4x) generator cards and generatorNETBOX products. Supports all three single modes: singleshot, continuous and single restart. Data is generated using library helper functions. The example uses the above mentioned LibPackage functions to set up the card.

rep_fifo_single.m

Example for all SpcMDrv based (M2i, M4i and M4x) generator cards and generatorNETBOX products. Shows the use of the FIFO output mode. Data is generated continuously and loaded into card memory. The example uses the above mentioned LibPackage functions to set up the card.

rep_sequence.m

Example for all SpcMDrv based (M2i, M4i and M4x) generator cards and generatorNETBOX products with the feature „Sequence Replay Mode“ installed.

m2i_io_std_sync.m

Shows the use of the synchronization option with two or more M2i mixed acquisition and generation cards. The cards are used in standard single acquisition mode and are started synchronously with software trigger. After the acquisition has finished channel 0 of each card is displayed on screen.

m2i_m3i_rec_std_sync.m

Shows the use of the synchronization option with two or more M2i or M3i acquisition cards or digitizerNETBOX products, that use more than one acquisition module internally. The cards are used in standard single acquisition mode and are started synchronously with software trigger. After the acquisition has finished channel 0 of each card is displayed on screen.

m4i_m2p_rec_std_multi_ts.m

Example for all M4i, M4x and M2p acquisition cards and digitizerNETBOX. Shows Multiple Recording data acquisition using standard mode with timestamp reference clock mode. The timestamp and reference clock values are also read out and displayed on the console. All channels are enabled for the acquisition and up to the first four acquired channels are displayed in a MATLAB plot window. The example uses the above mentioned LibPackage functions to set up the card.

m4i_rec_std_average.m

Standard example for all M4i and M4x acquisition cards and digitizerNETBOX products with the firmware option „Block Average“ installed.

m4i_rec_std_segmstat.m

Standard example for all M4i and M4x acquisition cards and digitizerNETBOX products with the firmware option „Block Statistics/Peak Detect“ installed.

m4i_m2p_rec_fifo_multi_ts.m

Example for all M4i, M4x and M2p acquisition cards and digitizerNETBOX. Shows Multiple Recording data acquisition using FIFO mode with timestamp reference clock mode, similar as the „m4i_m2p_rec_std_multi_ts.m“.

m4i_rep_fifo_single.m

Fifo example for all M4i and M4x based arbitrary waveform generator cards and generatorNETBOX products.

sb6_binary_import.m

Example showing how to import pure binary data that has been previously recorded with SBench6.

Error Codes

The following error codes could occur when a driver function has been called. Please check carefully the allowed setup for the register and change the settings to run the program.

error name	value (hex)	value (dec.)	error description
ERR_OK	0h	0	Execution OK, no error.
ERR_INIT	1h	1	An error occurred when initializing the given card. Either the card has already been opened by another process or an hardware error occurred.
ERR_TYP	3h	3	Initialization only: The type of board is unknown. This is a critical error. Please check whether the board is correctly plugged in the slot and whether you have the latest driver version.
ERR_FNCNOTSUPPORTED	4h	4	This function is not supported by the hardware version.
ERR_BRDREMAP	5h	5	The board index re map table in the registry is wrong. Either delete this table or check it carefully for double values.
ERR_KERNELVERSION	6h	6	The version of the kernel driver is not matching the version of the DLL. Please do a complete re-installation of the hardware driver. This error normally only occurs if someone copies the driver library and the kernel driver manually.
ERR_HWDRVVERSION	7h	7	The hardware needs a newer driver version to run properly. Please install the driver that was delivered together with the card.
ERR_ADRRANGE	8h	8	One of the address ranges is disabled (fatal error), can only occur under Linux.
ERR_INVALIDHANDLE	9h	9	The used handle is not valid.
ERR_BOARDNOTFOUND	Ah	10	A card with the given name has not been found.
ERR_BOARDINUSE	Bh	11	A card with given name is already in use by another application.
ERR_EXPHW64BITADR	Ch	12	Express hardware version not able to handle 64 bit addressing -> update needed.
ERR_FWVERSION	Dh	13	Firmware versions of synchronized cards or for this driver do not match -> update needed.
ERR_LASTERR	10h	16	Old error waiting to be read. Please read the full error information before proceeding. The driver is locked until the error information has been read.
ERR_BOARDINUSE	11h	17	Board is already used by another application. It is not possible to use one hardware from two different programs at the same time.
ERR_ABORT	20h	32	Abort of wait function. This return value just tells that the function has been aborted from another thread. The driver library is not locked if this error occurs.
ERR_BOARDLOCKED	30h	48	The card is already in access and therefore locked by another process. It is not possible to access one card through multiple processes. Only one process can access a specific card at the time.
ERR_DEVICE_MAPPING	32h	50	The device is mapped to an invalid device. The device mapping can be accessed via the Control Center.
ERR_NETWORKSETUP	40h	64	The network setup of a digitizerNETBOX has failed.
ERR_NETWORKTRANSFER	41h	65	The network data transfer from/to a digitizerNETBOX has failed.
ERR_FWPOWERCYCLE	42h	66	Power cycle (PC off/on) is needed to update the card's firmware (a simple OS reboot is not sufficient !)
ERR_NETWORKTIMEOUT	43h	67	A network timeout has occurred.
ERR_BUFFERSIZE	44h	68	The buffer size is not sufficient (too small).
ERR_RESTRICTEDACCESS	45h	69	The access to the card has been intentionally restricted.
ERR_INVALIDPARAM	46h	70	An invalid parameter has been used for a certain function.
ERR_REG	100h	256	The register is not valid for this type of board.
ERR_VALUE	101h	257	The value for this register is not in a valid range. The allowed values and ranges are listed in the board specific documentation.
ERR_FEATURE	102h	258	Feature (option) is not installed on this board. It's not possible to access this feature if it's not installed.
ERR_SEQUENCE	103h	259	Command sequence is not allowed. Please check the manual carefully to see which command sequences are possible.
ERR_READABORT	104h	260	Data read is not allowed after aborting the data acquisition.
ERR_NOACCESS	105h	261	Access to this register is denied. This register is not accessible for users.
ERR_TIMEOUT	107h	263	A timeout occurred while waiting for an interrupt. This error does not lock the driver.
ERR_CALLTYPE	108h	264	The access to the register is only allowed with one 64 bit access but not with the multiplexed 32 bit (high and low double word) version.
ERR_EXCEEDSINT32	109h	265	The return value is int32 but the software register exceeds the 32 bit integer range. Use double int32 or int64 accesses instead, to get correct return values.
ERR_NOWRITEALLOWED	10Ah	266	The register that should be written is a read-only register. No write accesses are allowed.
ERR_SETUP	10Bh	267	The programmed setup for the card is not valid. The error register will show you which setting generates the error message. This error is returned if the card is started or the setup is written.
ERR_CLOCKNOTLOCKED	10Ch	268	Synchronization to external clock failed: no signal connected or signal not stable. Please check external clock or try to use a different sampling clock to make the PLL locking easier.
ERR_CHANNEL	110h	272	The channel number may not be accessed on the board: Either it is not a valid channel number or the channel is not accessible due to the current setup (e.g. Only channel 0 is accessible in interlace mode)
ERR_NOTIFYSIZE	111h	273	The notify size of the last spcm_dwDefTransfer call is not valid. The notify size must be a multiple of the page size of 4096. For data transfer it may also be a fraction of 4k in the range of 16, 32, 64, 128, 256, 512, 1k or 2k. For ABA and timestamp the notify size can be 2k as a minimum.
ERR_RUNNING	120h	288	The board is still running, this function is not available now or this register is not accessible now.
ERR_ADJUST	130h	304	Automatic card calibration has reported an error. Please check the card inputs.
ERR_PRETRIGGERLEN	140h	320	The calculated pretrigger size (resulting from the user defined posttrigger values) exceeds the allowed limit.
ERR_DIRMISMATCH	141h	321	The direction of card and memory transfer mismatch. In normal operation mode it is not possible to transfer data from PC memory to card if the card is an acquisition card nor it is possible to transfer data from card to PC memory if the card is a generation card.
ERR_POSTEXCDSEGMENT	142h	322	The posttrigger value exceeds the programmed segment size in multiple recording/ABA mode. A delay of the multiple recording segments is only possible by using the delay trigger!
ERR_SEGMENTINMEM	143h	323	Memsize is not a multiple of segment size when using Multiple Recording/Replay or ABA mode. The programmed segment size must match the programmed memory size.
ERR_MULTIPLEPW	144h	324	Multiple pulsewidth counters used but card only supports one at the time.
ERR_NOCHANNELPWOR	145h	325	The channel pulsewidth on this card can't be used together with the OR conjunction. Please use the AND conjunction of the channel trigger sources.
ERR_ANDORMASKOVRLAP	146h	326	Trigger AND mask and OR mask overlap in at least one channel. Each trigger source can only be used either in the AND mask or in the OR mask, no source can be used for both.
ERR_ANDMASKEDGE	147h	327	One channel is activated for trigger detection in the AND mask but has been programmed to a trigger mode using an edge trigger. The AND mask can only work with level trigger modes.
ERR_ORMASKLEVEL	148h	328	One channel is activated for trigger detection in the OR mask but has been programmed to a trigger mode using a level trigger. The OR mask can only work together with edge trigger modes.
ERR_EDGEPERMOD	149h	329	This card is only capable to have one programmed trigger edge for each module that is installed. It is not possible to mix different trigger edges on one module.
ERR_DOLEVELMINDIFF	14Ah	330	The minimum difference between low output level and high output level is not reached.
ERR_STARHUBENABLE	14Bh	331	The card holding the star-hub must be enabled when doing synchronization.

error name	value (hex)	value (dec.)	error description
ERR_PATPWSMALLEDGE	14Ch	332	Combination of pattern with pulsewidth smaller and edge is not allowed.
ERR_PCICHECKSUM	203h	515	The check sum of the card information has failed. This could be a critical hardware failure. Restart the system and check the connection of the card in the slot.
ERR_MEMALLOC	205h	517	Internal memory allocation failed. Please restart the system and be sure that there is enough free memory.
ERR_EEPROMLOAD	206h	518	Timeout occurred while loading information from the on-board EEProm. This could be a critical hardware failure. Please restart the system and check the PCI connector.
ERR_CARDNOSUPPORT	207h	519	The card that has been found in the system seems to be a valid Spectrum card of a type that is supported by the driver but the driver did not find this special type internally. Please get the latest driver from www.spectrum-instrumentation.com and install this one.
ERR_FIFOHWOVERRUN	301h	769	Hardware buffer overrun in FIFO mode. The complete on-board memory has been filled with data and data wasn't transferred fast enough to PC memory. If acquisition speed is smaller than the theoretical bus transfer speed please check the application buffer and try to improve the handling of this one.
ERR_FIFOFINISHED	302h	770	FIFO transfer has been finished, programmed data length has been transferred completely.
ERR_TIMESTAMP_SYNC	310h	784	Synchronization to timestamp reference clock failed. Please check the connection and the signal levels of the reference clock input.
ERR_STARHUB	320h	800	The auto routing function of the Star-Hub initialization has failed. Please check whether all cables are mounted correctly.
ERR_INTERNAL_ERROR	FFFFh	65535	Internal hardware error detected. Please check for driver and firmware update of the card.