



QMORPHO

USER'S MANUAL

MULTI-CHANNEL DATA ACQUISITION SYSTEM FOR USE WITH SCINTILLATOR DETECTORS

- ◆ **ACQUIRE ENERGY HISTOGRAMS**
- ◆ **MEASURE COUNT RATES**
- ◆ **DISPLAY PULSE SHAPES**
- ◆ **COMPLEX TRIGGERS**
- ◆ **OPEN-SOURCE BACKEND**



BRIDGEPORT INSTRUMENTS, LLC
6448 E HWY 290, STE D-103
PHONE: 512-533-9933
WWW.BRIDGEPORTINSTRUMENTS.COM

**A
P
I**

Table of Contents

1 Overview.....	4
2 Interface device driver functions.....	5
2.1 LoadDLL.....	5
3 Interface management.....	5
3.1 USB global variables.....	6
3.2 qM_USB_Scan.....	7
3.3 qM_USB_Open and qM_USB_Close.....	7
3.4 Ethernet Global Variables.....	7
3.5 qM_EN_Open.....	7
3.6 qM_EN_Close.....	8
4 Basic IO (qMorpho_Mid.c).....	8
4.1 PacketHeader_0; PH_0.....	9
4.2 PacketHeader 1; PH_1.....	9
4.3 PacketHeader 2; PH_2, PacketHeader 3; PH_3:	9
4.4 Address Data Structure.....	10
4.5 qMorpho_IO.....	10
4.6 qM_Generate_Packet_Header.....	11
4.7 qM_PrepareRead.....	11
4.8 qRD_SP.....	12
4.9 qRD_CC.....	12
4.10 qWR_BCC_Controls.....	13
4.11 qWR_CC_Controls.....	13
4.12 qWR_CC_Actions.....	14
4.13 qWR_SP_Controls.....	14
4.14 qWR_SP_Actions.....	15
5 Midlevel SP-FPGA reads (qMorpho_Mid.c).....	16
5.1 qRD_CC_Version.....	16
5.2 qRD_SP_Statistics.....	16
5.3 qRD_SP_Version.....	17
5.4 qRD_SP_RawData.....	18
5.5 qRD_SP_Status.....	18
5.6 qRD_SP_Histogram.....	19
5.7 qRD_SP_Trace.....	20
5.8 qRD_SP_List_Mode.....	20
6 Register manipulations (qMorpho_Register.c).....	21
6.1 Control summary.....	22
6.2 Energy Controls.....	22
6.3 Gain controls.....	23
6.4 Histogram controls.....	24
6.5 List mode controls.....	25
6.6 Triggered-trace acquisition controls.....	26
6.7 Mode controls.....	26
6.8 Pulser controls.....	27

6.9	qM_SP_Parse_CR.....	27
6.10	qM_SP_Assemble_CR.....	28
6.11	qM_SP_Operation_To_Controls.....	29
6.12	qM_SP_Controls_To_Operation.....	29
6.13	qM_SP_Standard_Config.....	30
6.14	qM_SP_Quick_Configure.....	30
6.15	qM_SP_Parse_AR.....	31
6.16	qM_SP_Assemble_AR.....	32
7	Operations (qMorpho_Operations.c).....	33
7.1	qM_SP_Program_HV.....	33
7.2	qM_SP_Program_HV_Simple.....	34
7.3	qM_SP_Untriggered_Trace.....	34
7.4	qM_SP_UTrace_Simple.....	35
7.5	qM_SP_Start_DAQ.....	35
7.6	qM_SP_Start_DAQ_Simple.....	36
7.7	qM_SP_Stop_DAQ.....	37
7.8	qM_SP_Stop_DAQ_Simple.....	38
7.9	qM_SP_Suspend_DAQ.....	38
7.10	qM_SP_Suspend_DAQ_Simple.....	39
7.11	qM_SP_Resume_DAQ.....	39
7.12	qM_SP_Resume_DAQ_Simple.....	40
7.13	qM_SP_Clear_Stats.....	40
7.14	qM_SP_Clear_Stats_Simple.....	41
7.15	qM_SP_Get_Calibration.....	41
7.16	qM_SP_Get_Rates.....	42
7.17	HV2DAC.....	43
7.18	DAC2HV.....	44
8	Helper functions.....	44
8.1	SetBit, ClearBit, ToggleBit.....	44
9	Revision History.....	44

1 Overview

The API consists of a hierarchical set of six layers. At the very bottom is the code needed to manage the access to the physical interfaces, USB and Ethernet.

The second layer consists of functions for accessing the registers and memory blocks in all involved FPGAs, namely the ones on the qMorpho as well as the one on the backplane.

Layer 3, qMorpho_Register.c, provides translations between the bitfields in the registers and seven functional parameter groups such as those governing energy measurements, gain etc.

The fourth layer is of most use to the application programmer. It contains the functions necessary to operate the qMorphos, ie program the device settings, start/stop data acquisition and read back the data.

A unified command interface provides a convenient, but optional, common entry point for all Morpho commands (qM_Functions.c). Its usefulness lies in the fact that when creating a dll, only a single function, qMorphoCommand(), needs to be exported. Having a single data interface also simplifies connecting this code to a graphical user interface, such as LabView, IGOR or other.

In a typical application, programmers use only the high-level functions and some of the data read functions from the mid-level.

There is a single set of functions, all beginning with qM_, that are used to operate individual qMorphos through their USB or Ethernet interface as well as operating qMorphos plugged into a backplane by addressing the back plane.

The basic variable type is `unsigned short` (or USHORT), consisting of 16-bit words. All functions are of type `int`, meaning signed 16-bit integers.

qM_Functions.c (Command interface) One function to dispatch all commands
qMorpho_Operation.C (User interface) Short scripts to serve typical commands
qMorpho_Register.c Translate between bitfields and functional groups
qMorpho_Mid.c Basic register read and write
qMorpho_IO.c (interface) Interface management

Table 1: Software hierarchy.

2 Interface device driver functions

For the Ethernet interface these function have not yet been defined.

For the USB interface, the function to load the dll library file and all the wrapper functions are located in the file `BPI_FT245RM_API.c`.

2.1 LoadDLL

This function is used to load the dll library file (using `LoadLibrary`) and to create function pointers to all relevant functions within that DLL (using `GetProcAddress`). For each function we provide a wrapper such that the application programmer does not have to use pointers to a function. These functions are described the FTD2XX Programmer's Guide version 3.14 provided by FTDI.

3 Interface management

These are the lowest level functions to communicate with the hardware. They are located in the file `qMorpho_IO.c`.

The API stores minimal information about each qMorpho that is connected to the computer in a few global variables and arrays. Each physical interface requires a different set of data to be stored to describe the interface.

For access through USB this entails pointers to data structures related to the USB driver dll from FTDI (www.ftdi-chip.com) and the serial numbers. For access through Ethernet this includes MAC addresses and pointers to socket-related data structures.

Assuming that the system is controlled by a computer with a 16-bit processor, all variables are of type `unsigned short` (USHORT) or `float` by default. The 16-bit words exchanged with the qMorpho are always `unsigned short` or USHORT for short.

For any physical interface, USB, Ethernet, UART, SPI and others, all I/O functions use an address data structure that conveys routing information to the Morpho system. The same data structure is used in all configurations, whether it is a single qMorpho on the USB bus or a set of backplanes on the Ethernet, each carrying a number of qMorpho cards.

The I/O functions use the content of the address data structure to create the packet header that is part of every data transfer. The FPGA's involved (qM_backplane C&C, qMorpho C&C and qMorpho SP) use the routing information encoded in the packet header to send data to the correct recipient.

Address data structure		
Member type and name		Description
USHORT	iface	interface: USB245 or ETHERNET
USHORT	devNum	If there are n USB or Ethernet cables connecting to the system, devNum will range from 0 to n-1. A value devNum=0xFFFF is used to apply an action to all connected devices.
USHORT	slotNum	Geographic address of a qMorpho slot on a backplane. If a qMorpho is addressed directly via a cable, it will ignore the slot number.
USHORT	module	Number of a functional IO module inside an FPGA.
USHORT	channel	WRITE: Bit mask indicating the targeted channels on a qMorpho. READ: Only a single bit should be set. FPGA uses priority encoder to select a single channel to read from.
USHORT	direction	READ or WRITE
USHORT	numWords	Number of words to be transferred
USHORT	wordsPerDatum	Statistics counters and the histogram modules return 32-bit data, and hence have 2 words per datum. For all others the value is 1.

3.1 USB global variables

The global variables for the USB interface are listed below:

Variable	Description
char SerNums[MAX_DEVICES][SERNUM_LENGTH+1];	The strings containing the serial numbers
char *pSerNums[MAX_USB_DEVICES+1];	Pointers to the above strings
FT_HANDLE USB_QM_HANDLE[MAX_USB_DEVICES];	Device handles
char Manu[100];	Manufacturer name
char ManuID[100];	Manufacturer ID
char Descr[100];	Descriptor string
char SerNo[100];	Serial number
USHORT num_qM_USB;	Number of qMorpho (or backplane) boards found by qM_USB_Scan()

3.2 qM_USB_Scan

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	buffer	First word receives number of eMorphos found

Finds all connected devices and their serial numbers.

The function records the number of Morphos in the global variable num_qM_USB. It stores the serial number strings in SerNums and nulls the USB_QM_HANDLES. qM_USB_Scan limits the number of qMorphos / backplanes to MAX_USB_DEVICES should there be too many devices attached. The extraneous devices will be disregarded.

Returns 0 if the scan was successful and -1 otherwise.

3.3 qM_USB_Open and qM_USB_Close

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT	devNum	Device number

These two functions are used to open and close a particular device. They take one input parameter, namely the device number. Enumeration begins at zero. qM_USB_Open stores the device handle in pUSB_qM_HANDLE[devNum], while qM_USB_Close will NULL that entry. We open the devices using the serial number string as the identifier to the DLL.

Both functions operate on a single device if a legal device number is provided. But if devNum is negative or greater or equal MAX_QM_USB, the functions will operate on all devices found during the last qM_USB_Scan.

Both functions return 0 if successful and -1 or -2 otherwise.

3.4 Ethernet Global Variables

The Ethernet side of the software has not yet been established. Under the MS XP and Vista and using the OS winsock functions, the required globals would be:

<i>Variable</i>	<i>Description</i>
char qM_IPADDR[MAX_QM_ENET][16]	Ethernet IP addresses
SOCKET qM_SOCKET[MAX_QM_ENET]	Winsocket pointers

3.5 qM_EN_Open

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT	devNum	Device number

This function uses the IP address `qM_IPADDR[devNum]` to open a socket and then stores the socket pointer in the `qM_SOCKET[devNum]`. If `devNum` is negative or larger than `MAX_EN_DEVICES` the function will attempt to open all devices at the IP addresses in the `qM_IPADDR` array. It starts at `devNum=0` and proceeds until it fails, or encounters a zero address or reaches the limit of `MAX_EN_DEVICES`. For unconnected devices it fills the `qM_SOCKET` entries with `NULL`. It reports the number of opened devices in the global variable `num_qM_EN`.

It returns 0 if successful and a negative number otherwise.

3.6 `qM_EN_Close`

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT	devNum	Device number

Use this function to close the socket associated with the device number `devNum`. It Nulls the corresponding entry in the `qM_SOCKET` array. Use `devNum<0` or `> MAX_EN_DEVICES` to close all devices.

4 Basic IO (`qMorpho_Mid.c`)

This is a set of functions to address data (read and write) in the registers and memory blocks of the FPGAs on the `qMorpho` card as well as the backplane.

The `qMorpho` and the backplane FPGAs use a common mechanism to exchange data with a host computer.

There are two slightly different forms of write commands. Writing to the action registers requires that the host sends eight 2-byte words---four packet headers and four action register contents. For all other writes, especially the control registers, a write operations consists of sending 32 16-bit words. The first four words are again used as a packet header. Their content determines how data are routed through the system.

The purpose of the `register_IO` functions described below is to set all required bit patterns in the packet header to connect the host with the correct module in the correct FPGA. For completeness, we show the packet header content below, but programmers should not have to concern themselves with the packet header contents.

These are not memory-mapped registers. Rather they receive the 4-word header which is part of each data transfer from the host to any FPGA.

4.1 PacketHeader_0; PH_0

15							8								0
<i>FS</i>		<i>SB</i>					<i>SN</i>								

Bit definitions:

FS[1:0]: FPGA select; 0 → Signal processing FPGA on the qMorpho, 1 → C&C FPGA on the qMorpho; the value 2 is currently unused, but reserved for a trigger FPGA on the backplane, 3 → C&C FPGA on the backplane.

SN[7:0]: Slot number. Used only when addressing a backplane. SN=0 → the backplane is the target; 0<SN<255 → a qMorpho in slot number SN is the target; SN=255 → all qMorpho slots are a target (broadcast). Note that a broadcast is valid only for writes.

SB[5:0]: Number of bytes to send to host in response to a read command from a serial interface. In some implementations (4-wire SPI, UART) the client needs to send a command to the server, telling it how many bytes it wishes to receive during the next read action. Unless instructed otherwise, set this bit-field to zero.

4.2 PacketHeader 1; PH_1

15							8								0
<i>CM</i>			<i>MA</i>				<i>PA</i>				<i>DC</i>	<i>AC</i>			

AC: Address clear; This resets the address counter in the byte interface before the 28-word data payload has been written. This bit self clears.

DC: Delayed address clear; This resets the address counter in the byte interface after the 28-word data payload has been written. This bit self clears.

MA[3:0]: Module address; Used to select up to 16 functional sub units in this FPGA design. The legal values are: 0 → Control registers, 1 → Action registers, 2 → Statistics registers, 3 → Version registers, 4 → Histogram memory, 5 → Trace memory, 6 → List mode data buffer, 7 → User data memory, 8 → Raw data registers and 9 → Status register.

CM: Channel mask. For write operations one or more bits, indicating the selected channel, can be set. In a read operation the C&C FPGA will use a priority encoder to select one channel if more than one bit in the channel mask is set.

PA[5:0]: 6-bit page address. A page is 256 bytes long. After the data payload has been written to the target FPGA, its LocalAddress counter will be set to 0x100 * PA. The next read will then commence from that address. This feature is useful when controlling the qMorpho with an 8-bit controller/processor that can address internal memory only in units of 256-byte pages.

4.3 PacketHeader 2; PH_2, PacketHeader 3; PH_3:

Reserved. Set to zero.

4.4 Address Data Structure

To select a target, the code uses a structure `qM_ADDRESS` to pass all relevant information to the I/O functions. Not every functions reads all address fields and we indicate for each function which fields are relevant. The `qM_ADDRESS` structure is defined in `qMorpho.h` and shown below.

Structure <code>qM_ADDRESS</code> member type and name		Description
unsigned short	<code>iface</code>	USB245 or ETHERNET
unsigned short	<code>devNum</code>	Device number
unsigned short	<code>slotNum</code>	qMorpho slot on a backplane
unsigned short	<code>channel</code>	4-bit mask indicating addressed channel; LSB maps to channel 0, MSB maps to channel 3.
unsigned short	<code>module</code>	Module address inside the FPGA
unsigned short	<code>direction</code>	READ or WRITE
unsigned short	<code>numWords</code>	Number of 16-bit words to be transferred
unsigned short	<code>WordsPerDatum</code>	Histograms and Statistics register contents are 4-byte items, i.e. 2 words per datum; all others are 2-byte (1-word) items.

In the current implementation, which is tailored towards a 16-bit machine, the I/O function `qMorpho_IO` ignores the `WordsPerDatum` member of the address structure.

4.5 `qMorpho_IO`

This is the single I/O function to be used to exchange data with the FPGAs on the qMorpho and the qMorpho backplane.

Input parameter, type and name		Description
<code>qM_ADDRESS *</code>	<code>addr</code>	Target address Uses all members of the address data structure
<code>USHORT *</code>	<code>dataFromHost</code>	16-bit data coming from the host computer
<code>USHORT *</code>	<code>dataToHost</code>	16-bit data going to the host computer

I/O array of type <code>USHORT</code>	Contents when using the unified command interface
<code>address</code>	(<code>address[0]</code> .. <code>address[7]</code>)
<code>dataFromHost</code>	data to be sent
<code>dataToHost</code>	buffer will receive data if this was a read request

Table 2: Usage of I/O arrays when using the unified command interface.

The structure of the data to be sent depends on the target.

<i>Case</i>	<i>dataFromHost</i>
Write data to any set of control registers; Read data from any module in any FPGA	(PH[0] .. PH[3]), (CR[0] .. CR[27])
Write to any set of action registers; i.e. issue a command.	(PH[0] .. PH[3]), (AR[0] .. AR[3])

Table 3: Set up of the *dataFromHost* array when using *qMorpho_IO* through the unified command interface.

(PH[0] .. PH[3]) denotes the four 16-bit packet header words. (CR[0] .. CR[27]) are the 28 control register words, and (AR[0] .. AR[3]) are the four action register words.

4.6 *qM_Generate_Packet_Header*

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses all members of the address data structure
USHORT *	packet	Pointer 4-word packet header

This low-level function will never have to be called explicitly by an application programmer. All I/O functions, except the primitive *qMorpho_IO*, call this function to map the address data structure members into the bit-fields of the 4-word packet header. All characteristics of the firmware that involve the packet header are encapsulated in this function.

The I/O functions calling *qM_Generate_Packet_Header* combine the created packet header with any data they have to send.

4.7 *qM_PrepareRead*

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel, module
USHORT *	controls	Pointer to the data to be written, typically controls and actions.

This function primes a channel and module for a subsequent read. The next read command will pull data from the location requested by this function.

In practice this function does not have to be called by the application programmer. It is embedded in all SP-FPGA read functions, including the general-purpose *qRD_SP* and *qRD_CC* which will read data from any location within an SP-FPGA or CC-FPGA.

4.8 qRD_SP

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, module, channel, numWords
USHORT *	buffer	Pointer to buffer to receive the data

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Buffer will receive data

Use this function to read data from an SP-FPGA on a qMorpho. The function will return numWords entries into the 16-bit array *buffer*.

The function will first prepare the qMorpho for the read by calling qM_PrepRead and then call qMorpho_IO.

In this implementation of the API (geared towards 16-bit embedded processors) the wordsPerDatum member of the address structure is ignored. In a 32-bit implementation, the function would combine two subsequently read words into a 32-bit datum before pushing it into a 32-bit buffer. In this case the function assumes that the low word is read first. Histogram and Statistics counter data are examples of 4-byte (2-word) data items.

4.9 qRD_CC

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, module, channel, numWords
USHORT *	buffer	Pointer to buffer to receive the data

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Buffer will receive data

Use this function to read data from the CC-FPGA on a qMorpho. The function will return

numWords entries into the 16-bit array *buffer*.

The function will first prepare the qMorpho for the read by calling qM_PrepareRead and then call qMorpho_IO.

In this implementation of the API (geared towards 16-bit embedded processors) the wordsPerDatum member of the address structure is ignored. In a 32-bit implementation, the function would combine two subsequently read words into a 32-bit datum before pushing it into a 32-bit buffer. In this case the function assumes that the low word is read first. Histogram and Statistics counter data are examples of 4-byte (2-word) data items.

4.10 qWR_BCC_Controls

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses interface, devNum and module
USHORT *	controls	Pointer to the 28 control register data to be written

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	(CR[0] .. CR[27])
dataToHost	no entries

This function writes 28 control registers to a selected module in the backplane C&C FPGA with the indicated device number. The data to be written are stored in the array pointed to by controls. The function will generate a standard packet header that contains the routing information.

In case this function addresses a qMorpho, instead of a backplane, the qMorpho will receive the data but not overwrite any of its registers.

Returns 0 if successful, and -1 if qM_IO returns an error.

4.11 qWR_CC_Controls

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: interface, devNum, slotNum, module
USHORT *	controls	Pointer to the 28 control register data to be written

This function writes 28 control registers into the C&C FPGA of a qMorpho. For standalone qMorphos it is device number *devNum* that is addressed. For a backplane-based system the function addresses the qMorpho in slot *slotNum* on backplane number *devNum*. Standalone qMorphos ignore the *slotNum* argument. The function will generate a standard packet header that contains the routing information.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	(CR[0] .. CR[27])
dataToHost	no entries

Returns 0 if successful, and -1 if qM_IO returns an error.

4.12 qWR_CC_Actions

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, module
USHORT*	actions	Pointer to the 4 action register data to be written

Writing to the action registers of the CC-FPGA is done via a short write. The qMorpho only expects 8 words to be written (4 packet header + 4 action register words).

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	(AR[0] .. AR[3])
dataToHost	no entries

Returns 0 if successful, and -1 if qM_IO returns an error.

4.13 qWR_SP_Controls

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel, module
USHORT *	controls	Pointer to the data to be written, typically controls and actions.

This function writes data into the control registers of the SP FPGA of a qMorpho. For standalone qMorphos it is device number *devNum* that is addressed. For a backplane-based system the function addresses the qMorpho in slot *slotNum* on backplane number *devNum*. Standalone qMorphos ignore the *slotNum* argument.

If the channel mask *addr.channel* has more than one bit set, data will be written simultaneously to all channels whose bits are set. When writing to the control registers, this feature is rarely used since parameters such as trigger thresholds and high voltage setting will most likely vary between channels.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	(CR[0] .. CR[27])
dataToHost	no entries

Returns 0 if successful, and -1 if qM_IO returns an error.

4.14 qWR_SP_Actions

<i>Input parameter type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel, module
USHORT *	controls	Pointer to the data to be written, typically controls and actions.

This function writes data into the control registers of the SP FPGA of a qMorpho. For standalone qMorphos it is device number *devNum* that is addressed. For a backplane-based system the function addresses the qMorpho in slot *slotNum* on backplane number *devNum*. Standalone qMorphos ignore the *slotNum* argument.

If the channel mask *addr.channel* has more than one bit set, data will be written simultaneously to all channels whose bits are set. When writing to the action registers of one or more SP-FPGAs, this is a very useful feature, as it allows to simultaneously clear statistics registers and start/stop DAQ in many channels at the same time.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	(AR[0] .. AR[3])
dataToHost	no entries

Returns 0 if successful, and -1 if qM_IO returns an error.

5 Midlevel SP-FPGA reads (qMorpho_Mid.c)

These functions are used to read data from the modules inside an SP or CC-FPGA. All data transfer from and to the FPGA follows the Windows/Intel convention of sending the low-order byte first. There are nine different destinations for data exchange within the signal processing FPGAs and fewer for the C&C FPGAs.

5.1 qRD_CC_Version

Use this function to read data from the version registers of the SP-FPGA. Note: There is no function to partially read the version registers. The content of these registers does not change after power up.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	Version	Points to 4-word data array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 4 data words

Returns 0 if successful and -1 if qM_IO returns an error.

See the FPGA_Map document for the contents of the C&C version registers.

5.2 qRD_SP_Statistics

Use this function to read the Statistics registers in the SP FPGA of a qMorpho.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: interface, devNum, slotNum, channel
USHORT *	Statistics	Points to 8-word data array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 8 data words

Returns 0 if successful, and -1 if qM_SP_IO returns an error.

The function returns 8 words of data as shown in the table below.

<i>16-bit words</i>	<i>Description</i>
RunTime_low	Time since the statistics counter was last cleared. 1 LSB = 65536 clock ticks.
RunTime_high	
Events_low	Number of accepted events since the statistics counter was last cleared.
Events_high	
Triggers_low	Number of recognized triggers since the statistics counter was last cleared.
Triggers_high	
DeadTime_low	Measured dead time since the statistics counter was last cleared. 1 LSB = 65536 clock ticks.
DeadTime_high	

Table 4: Contents of the Statistics registers. A clock tick is one period of the ADC sampling clock. The four datums are 32-bit items stored in consecutive 16-bit words, low word first.

Accepted events are those that are not flagged as piled up or went out of the ADC range. Depending on the compression settings, of SP-FPGA control register 12, not all accepted events will be recorded in the histogram. They will, however, always be included in a list mode run.

The correct way to interpret the content of the statistics registers will depend on the application. When operating the channels independently and performing histogram data acquisition, the statistics data can be interpreted as follows.

$$\text{RunTime} = \frac{65536 \cdot (\text{RunTime_low} + \text{RunTime_high} * 0x10000)}{\text{ADC_Sampling_Rate}}$$

$$\text{DeadTime} = \frac{65536 \cdot (\text{DeadTime_low} + \text{DeadTime_high} * 0x10000)}{\text{ADC_Sampling_Rate}}$$

$$\text{EventRate} = \frac{(\text{Events_low} + \text{Events_high} * 0x10000)}{\text{RunTime}}$$

$$\text{TriggerRate} = \frac{(\text{Trigger_low} + \text{Trigger_high} * 0x10000)}{\text{RunTime}}$$

$$\text{TrueTriggerRate} = \frac{\text{TriggerRate}}{1 - \text{DeadTime}/\text{RunTime}}$$

5.3 qRD_SP_Version

Use this function to read data from the version registers of the SP-FPGA. Note: There is no function to partially read the version registers. The content of these registers does not change after power up.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	Version	Points to 4-word data array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 4 data words

Returns 0 if successful and -1 if qM_IO returns an error.

There is only one set of version registers in an SP-FPGA. However, *addr.channel* is required so the C&C-FPGA can select the correct SP-FPGA. Choose *addr.channel* = 1 or 2 to read the version registers of first SP-FPGA and *addr.channel* = 4 or 8 to read those of the second. See the FPGA documentation for the contents of the SP-FPGA version register.

5.4 qRD_SP_RawData

Use this function to read data from the raw data registers of the SP-FPGA. Note: There is no function to partially read the raw data registers.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	Version	Points to 8-word data array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 8 data words

Returns 0 if successful and -1 if qM_IO returns an error.

Consult the FPGA documentation for the contents of the SP-FPGA raw data register.

5.5 qRD_SP_Status

Use this function to read data from the version register of the SP-FPGA.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	Status	Points to 1-word data array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 1 data word

Returns 0 if successful and -1 if qM_IO returns an error.

There is only one version register in an SP-FPGA. However, *addr.channel* is required so the C&C-FPGA can select the correct SP-FPGA. See the FPGA documentation, or section 3.10 of this document, for the contents of the SP-FPGA status register.

To find the status of all four channels on a qMorpho requires two reads, one directed towards ch0 or ch1 and the second read directed towards ch2 or ch3.

Six bits are used in the status word:

<i>Constant</i>	<i>Value</i>	<i>Description</i>
ST_HISTOGRAM_DONE_0	0x1	Pulls high when histogram acquisition is complete.
ST_LISTMODE_DONE_0	0x2	Pulls high when list mode buffer is full.
ST_TRACE_DONE_0	0x4	Pulls high when triggered trace has been acquired.
ST_HISTOGRAM_DONE_1	0x10	Pulls high when histogram acquisition is complete.
ST_LISTMODE_DONE_1	0x20	Pulls high when list mode buffer is full.
ST_TRACE_DONE_1	0x40	Pulls high when triggered trace has been acquired.

Table 5: Status bits in the SP-FPGA

The lower three bits indicate the DAQ status in ch 0 of the SP-FPGA, while bits 4 through 6 indicate the DAQ status in ch 1 of the SP-FPGA. For the qMorpho as a whole ch 0 and ch 1 of SP-FPGA # 0 are called channels 0 and 1, but ch 0 and 1 of SP-FPGA #1 are called channels 2 and 3.

5.6 qRD_SP_Histogram

Use this function to read the energy histogram from a specified target. Note: There is no function to partially read the histogram memory.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	data	Points to 16-bit data stored in an array of type unsigned short. There must be room for 4096 entries in that array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
address	(address[0] .. address[7])
dataFromHost	ignored
dataToHost	Receives 4096 data words

Returns 0 if successful, and -1 if qM_IO returns an error.

Histogram data are 32-bit items. They are returned low-word first, followed by the high word.

5.7 qRD_SP_Trace

Use this function to read 1024 consecutive ADC samples. Note: There is no function to partially read the trace memory. Regardless of the ADC accuracy (10-bit or 12-bit) the data are delivered as 16-bit words with a sign bit in the MSB position and a 15-bit mantissa. Data from a 10-bit ADC are packed into 16-bit words as {0, ADC[9:0], 0,0,0,0,0}. Data from a 12-bit ADC are packed into 16-bit words as {0, ADC[11:0], 0,0,0}.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: interface, devNum, slotNum, channel
USHORT *	data	Points to 16-bit data stored in an array of type unsigned long. There must be room for 1024 entries in that array.

Returns 0 if successful, and -1 if qM_IO returns an error.

5.8 qRD_SP_List_Mode

Use this function to read the list mode memory. Note: There is no function to partially read the list mode memory.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: interface, devNum, slotNum, channel

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT*	data	Points to 16-bit data stored in an array of type unsigned long. There must be room for 1024 entries in that array.

Data will be stored consecutively, 3 long-words per event up to a maximum of 340 events. numEvents = data[0] contains the number of events in the buffer. Note that the list mode memory in the FPGA is never cleared. Hence you have to use numEvents to find the number of valid entries in the data buffer.

There are two selectable formats for the list mode data: Long time stamp and short time stamp. This is controlled via the list mode type in control register CR15. In both cases an event uses three 16-bit words. The finest available time stamp granularity available is $dT = 1 / \text{ADC_Sampling_Rate}$.

The first valid data entry is in data[4].

When using the long time stamp, the time stamp is 32-bits long. The event data format for event number n , with $0 \leq n \leq 339$ is:

<i>Offset</i>	<i>Description</i>
data[4 + 3*n + 0]	16-bit energy, 16 times the value entered into the histogram.
data[4 + 3*n + 1]	16-bit time; lower word. 1 LSB = dT
data[4 + 3*n + 2]	16-bit time; higher word

When using the short time stamp, the event data format is:

<i>Offset</i>	<i>Description</i>
data[4 + 3*n + 0]	16-bit energy, 16 times the value entered into the histogram.
data[4 + 3*n + 1]	16-bit phoswich sum, 16 times the value computed from the original sum and the PIT scaling value.
data[4 + 3*n + 2]	16-bit time in units of $32 * dT$

Returns 0 if successful, and -1 if qM_IO returns an error.

6 Register manipulations (qMorpho_Register.c)

The qMorpho has been designed to be a flexible and expandable DAQ platform. As a result, many control and action registers contain unused bit-fields, which may be used in later editions. User software should not alter unused bit-fields or overwrite with zeroes as this would prevent future software from operating correctly.

To aid the application programmer in this regard, we provide a set of functions, that accept a set of control or action registers, parse their content and reassemble them after the content has

been changed.

For the signal-processing FGPGA (SP-FPGA) we have created seven content groups into which we bundle related parameters and controls.

6.1 Control summary

The table below lists the seven control groups established so far.

<i>Name</i>	<i>Description</i>
Energy_Ctrl	Contains all parameters controlling the digital signal processing
Gain_Ctrl	Parameters associated with gain control, including high voltage
Histo_Ctrl	Histogramming controls
List_Ctrl	List mode controls
Trace_Ctrl	Trace capture controls
Mode_Ctrl	DAQ mode controls
Pulser_Ctrl	Controls to operate the electronic pulser

6.2 Energy Controls

This group comprises six elements all of which govern various aspects of the digital signal processing involved in measuring pulse energies.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
TRIG	0..1023	Trigger threshold for energy measurements.
INTEGRATION	0..65535	Integration time in clock ticks
PILEUP	0..INTEGRATION	Pile up inspection time, in clock ticks
HOLD_OFF	0..65535	Trigger dead time in clock ticks
BASELINE_TRIG	0..1023	Trigger threshold for baseline measurements
PID_TIME	0..65535	Partial-integration time for pulse shape analysis

Table 6: The members of the energy control group

TRIG: Trigger threshold for data acquisition. A value of 1023 always maps to ADC full scale, even on systems that have waveform digitizing ADCs with more or less than 10 bit accuracy. The trigger level is measured with respect to the DC-baseline. If the DC baseline is 100 ADC bins and the trigger level is set to 10, then the channel will trigger when the signal rises above 110 ADC bins.

INTEGRATION: Pulse energies are measured as the sum of pulse values over the integration time. Let $INTEGRATION = N$, $ADC(k) = ADC \text{ data}$, $DC = \text{baseline}$

The energy is computed as

$$\text{Energy} = \sum_{k=0}^N (\text{ADC}(k) - DC)$$

where k=0 is the sample 6 ADC sampling clock cycles before the energy trigger pulse.

PILEUP: This quantity controls a simple pile up inspection mechanism. Similar to before, let $\text{PILEUP} = P$. The energy measuring unit computes a quantity

$$\text{Pile} = \sum_{k=0}^P (\text{ADC}(k) - DC)$$

and make a comparison to establish if there was pile up:

$$\text{Piled_up} = 2 \cdot \text{Pile} > \text{Energy}$$

Setting PILEUP equal to INTEGRATION turns pile up inspection off.

HOLD_OFF: This parameter controls the retriggerable dead time counter. No new energy measurements will be started until the dead time counter has expired. Always set $\text{HOLD_OFF} \geq \text{INTEGRATION}$.

BASELINE_TRIG: This is a trigger threshold to determine if a pulse is present as opposed to just noise on a DC-baseline. Set well above noise level. In almost all cases a value between 3 and 5 is correct.

PID_TIME: This is used to compute a sum over the first part of the PMT pulse. Let $\text{PID_TIME} = PIT$. The FPGA computes:

$$\text{PID_Sum} = \sum_{k=0}^{PIT} (\text{ADC}(k) - DC)$$

This value can be stored in list mode data acquisition and is also available in the raw data registers. In some system it is possible to discriminate between different types of radiation by plotting PID vs Energy where $\text{PID} = (\text{PID_Sum} / \text{Energy})$

6.3 Gain controls

All parameters that affect the gain of the system are combined in this group.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
HV_DAC	0..4095	Value to be programmed into the HV-DAC
ESCALE	0..15	Compression factor
PIDSCALE	0..15	Compression factor
RESIST	0..15	Select gain setting resistors
FACTOR	0..65535	Digital gain

Table 7: The members of the gain control group

HV_DAC: This value will be programmed into the DAC that is used to control high voltage. The full scale of the 12-bit DAC is 3.00 V. Hence, the DAC output voltage is $DV = \text{HV_DAC} / 4095 \cdot 3.00 \text{ V}$. For TwinBase high voltage supplies the output HV will

be 1000 times DV.

ESCALE: The energy sum computed by the FPGA may have a large numerical value. Use ESCALE to fit a full scale pulse into the range accomodated by 16-bit list mode energy words or the 10 to 11-bit histogram range: $E_{out} = \text{floor}(\text{Energy}/2^{\text{ESCALE}})$.

PIDSCALE: This has the function as ESCALE but is applied to the PID_SUM.

RESIST: Chooses the gain-setting resistor in the amplifier stage. Use 0, 1, 2, 4, or 8 to select a transimpedance of 100 Ω, 430 Ω, 1100 Ω, 3400 Ω, or 10,100 Ω, respectively. Do not use intermediate values.

<i>RESIST</i>	<i>gain</i>	<i>RESIST</i>	<i>gain</i>	<i>RESIST</i>	<i>gain</i>	<i>RESIST</i>	<i>gain</i>
0	100Ω	4	3400Ω	8	10100Ω	12	13400Ω
1	430Ω	5	3730Ω	9	10430Ω	13	13730Ω
2	1100Ω	6	4400Ω	10	11100Ω	14	14400Ω
3	1430Ω	7	4730Ω	11	11430Ω	15	14730Ω

Table 8: Effective input amplifier transimpedance (gain) as a function of RESIST.

FACTOR: It is not always possible to adjust the high voltage to achieve a certain target gain, for instance of exactly 1.00 keV/histogram bin. Use factor in this case, to tweak the gain digitally: $E_{final} = E_{out} \cdot \text{FACTOR} / 32768$. Set FACTOR=32768 as a default and adjust only when needed.

6.4 Histogram controls

All parameters controlling histogram acquisition are combined in this group.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
COND	0..3	Stop condition
AMPL	0..1	Histogram amplitudes or energies
SEGEN	0..1	Enable histogram splitting
VAL	0..3	Event validation
CLREN	0..1	Clear enable
REQ_LOW	0..65535	Low and high word of 32-bit value REQUEST.
REQ_HIGH	0..65535	

Table 9: The members of the histogram control group.

COND: Condition used to stop a histogram run.

0-> No histogram

1-> End when RunTime >= REQUEST

2-> End when LiveTime >= REQUEST

3-> End when AcceptedEvents >= REQUEST

See below how REQUEST is constructed.

AMPL: Set to zero to measure energies via the sums shown in section 6.2. Set to 1 to test if the pulses adequately cover the ADC input range. With AMPL=1 the unit measures maximum pulse height above baseline.

SEGEN: Set to 0 for standard operation. Set to 1 to enable histogram memory splitting. This is a non-standard feature and is not present in all qMorpho implementations.

VAL: Select a validation source for pulses:

0-> Local = not out of range & not piled up

1-> Local & TriggerOK from C&C-FPGA

2-> not assigned

3-> not assigned

CLREN: ClearEnable. Must be set to one before a spectrum can be cleared.

REQ_LOW, REQ_HIGH: The FPGA constructs a 32-bit REQUEST from these two 16-bit words: $REQUEST = REQ_LOW + 0x10000 \cdot REQ_HIGH$. If the histogram end condition is a time, 1 LSB of REQUEST equals 65536 ticks of the ADC sampling clock. If the end condition is reaching a target count value for the entire spectrum, REQUEST is that maximum count.

6.5 List mode controls

All parameters controlling list mode data acquisition are combined in this group.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
COND	0..3	Stop condition
VAL	0..3	Event validation
CLREN	0..1	Clear enable

Table 10: The members of the list mode control group.

COND: Condition used to stop a histogram run.

0-> No list mode

1-> End when list mode buffer is full (It holds 340 events)

2-> not assigned

3-> not assigned

VAL: Select a validation source for pulses:

0-> Local = not out of range & not piled up

1-> Local & TriggerOK from C&C-FPGA

2-> not assigned

3-> not assigned

CLREN: ClearEnable. Must be set to 1 before a list mode buffer can be cleared and readied for a new acquisition cycle.

6.6 Triggered-trace acquisition controls

All parameters controlling list mode data acquisition are combined in this group.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
COND	0..3	Stop condition
PRETRIGGER	0..1023	Number of pretrigger data points
VAL	0..3	Event validation
CLREN	0..1	Clear enable

Table 11: The members of the trace acquisition control group.

COND: Condition used to stop a histogram run.

0-> No list mode

1-> End when list mode buffer is full (It holds 340 events)

2-> not assigned

3-> not assigned

PRETRIGGER: In the recorded trace the trigger occurred at sample number PRETRIGGER. This allows to record pre-trigger data.

VAL: Select a validation source for pulses:

0-> Local = not out of range & not piled up

1-> Local & TriggerOK from C&C-FPGA

2-> not assigned

3-> not assigned

CLREN: ClearEnable. Must be set to 1 before a trace buffer can be cleared and readied for a new acquisition cycle.

6.7 Mode controls

This group includes parameters to set data acquisition types and control the statistics counters.

<i>Group member</i>	<i>Range</i>	<i>Description</i>
STATS_CLREN	0..1	Clear-enable for the statistics counters
DAQ_MODE	0..3	Data acquisition mode

Table 12: The members of the mode control group.

STATS_CLREN: The statistics counters can be cleared only if this bit is set to 1.

DAQ_MODE: Select a validation source for pulses:

0-> Idle or scanning

1-> Ready for data taking

2-> not assigned

3-> not assigned

The DAQ mode setting affects the behavior of the statistics registers and data acquisition, cf table below.

DAQ_MODE	Statistics counters	DAQ (Histo, List, Trace)
0	Free running	Inhibited
1	Enabled only during active DAQ. They stop when DAQ stops.	Allowed

Table 13: DAQ modes

6.8 Pulser controls

All parameters controlling the built-in electronic pulser are combined in this group.

Group member	Range	Description
PERIOD	0..4	Set pulser period
WIDTH	0..3	Set pulse width
SEP	0..3	Set double pulse separation
TRIGGER	0..1	Pulser triggers energy measurement
ENABLE	0..1	Output enable

Table 14: The members of the pulser control group.

The pulser operates off a counter running at the ADC sampling clock rate (RATE)

PERIOD: Let $PERIOD = P$ and the pulser period is $Period = 2^{P+1}/RATE$.

WIDTH: Let $WIDTH = W$ and the pulse width is $Width = 2^{W+1}/RATE$.

SEP: Let $SEP = S$ and the double pulse separation is $Separation = 2^{S+1}/RATE$.
A value of SEP=0 turns this feature off, and the pulser produces only single pulses.

TRIGGER: When set to 1, the unit will start an energy measurement as if an input pulse from a PMT had been received. Set to 0, to suppress triggers generated by the pulser.

ENABLE: The pulses are sent to output pins on the qMorpho board. The ENABLE bit acts as an output enable.

6.9 qM_SP_Parse_CR

This function breaks out all bit fields in a given control register, including areas marked as

unused, and stores them sequentially in a pattern array.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Controls	Points to an array of 28 16-bit SP-FPGA control registers
USHORT *	pattern	Points to an array of 16-bit words to receive the bit-field values.
USHORT	numRegister	Identifies the control register to be parsed.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(CR[0] .. CR[27]), numRegister
dataToHost	pattern[0] .. pattern[15]

Table 15: Usage of I/O arrays when using the unified command interface. (CR[0]..CR[27]) denotes the array of 28 control registers.

Calls:	-
Called by:	qM_SP_Controls_To_Operation

Returns 0 if successful and -1 if *numRegister* was illegal.

6.10 qM_SP_Assemble_CR

This function receives a pattern array and constructs the content of a single control register.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Controls	Points to an array of 28 16-bit SP-FPGA control registers
USHORT *	pattern	Points to an array of 16-bit pattern words.
USHORT	numRegister	Identifies the control register to be built.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(pattern[0] .. pattern[15]), numRegister
dataToHost	(CR[0] .. CR[27])

Calls:	-
Called by:	qM_SP_Operation_To_Controls

Multiple calls to qM_SP_Assemble_CR can be used to assemble the 28 control registers one by one.

Returns 0 if successful and -1 if *numRegister* was illegal.

6.11 qM_SP_Operation_To_Controls

This function copies bitfields associated with each group of settings into the control registers. Currently there are seven recognized groups: energy, gains, histogram, listmode, trace, mode, and pulser.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Controls	Points to an array of 28 16-bit SP-FPGA control registers
USHORT *	opCtrl	Points to an array of 16-bit opCtrl words, such as energyCtrl, etc.
USHORT	operation	Identifies the control/operations group.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(CR[0] .. CR[27]), (opCtrl[0] .. opCtrl[15])
dataToHost	(CR[0] .. CR[27]),

The function uses the information in the *opCtrl* array and the *operation* identifier to fill in the appropriate bitfields in the Control registers array pointed to by *Controls*. The function takes care not to alter the contents of unrelated or unused bitfields. It uses *qM_SP_Assemble_CR* to build the individual control registers.

Calls:	qM_SP_Assemble_CR
Called by:	qM_SP_Standard_Config, API top level function

Returns 0 if successful and -1 if the *operation* identifier was illegal.

6.12 qM_SP_Controls_To_Operation

This function parses the control register contents in order to extract operations control values for the requested operations type. The data for a given operations group are usually encoded in several control registers.

The function uses multiple calls to *qM_SP_Parse_CR* to extract the requested information from the different control registers.

Currently there are seven recognized groups: energy, gains, histogram, listmode, trace, mode, and pulser.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Controls	Points to an array of 28 16-bit SP-FPGA control registers
USHORT *	opCtrl	Points to an array of 16-bit opCtrl words, such as energyCtrl, etc.
USHORT	operation	Identifies the control/operations group.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(CR[0] .. CR[27]), operation
dataToHost	(opCtrl[0] .. opCtrl[15])

Calls:	qM_SP_Parse_CR
Called by:	Is API top level function

The function reads the control register data (pointed to by *Controls*) and fills the opCtrl array as directed by the *operation* identifier.

Returns 0 if successful and -1 if the *operation* identifier was illegal.

6.13 qM_SP_Standard_Config

This function is used to create a standard configuration which can then be manipulated using the two functions qM_SP_Controls_To_Operation and qM_SP_Operation_To_Controls.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Controls	Points to an array of 28 16-bit SP-FPGA control registers

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	-
dataToHost	(CR[0] .. CR[27])

Calls:	qM_SP_Operation_To_Controls
Called by:	Is API top level function

This function fills the Controls array with predetermined values. To manipulate the control registers consider the following code fragment, showing the read/modify/write approach:

```
qM_SP_Controls_To_Operation((USHORT *)Controls, (USHORT *)energyCtrl,
                           (USHORT) SP_CTRL_ENERGY );
energyCtrl[SP_ENERGY_TRIG] = 10;
qM_SP_Operation_To_Controls((USHORT *)Controls, (USHORT *)energyCtrl,
                           (USHORT) SP_CTRL_ENERGY );
```

After these three steps the control registers have been correctly updated and there is no need to remember the exact structure of the control register contents.

6.14 qM_SP_Quick_Configure

This function is used to set all the control register values that control the signal processing.

The function will set the integration time, the DC-Wait, the PUT value, start delay and the scaling values appropriately.

The legal scintillator types are listed in qM_Morpho.h. They include: SC_PLASTIC, SC_YAP_CE, SC_LA_CL3, SC_NA_TL, SC_BGO, SC_CSI_CO3, SC_CSI_NA, SC_CSI_TL, SC_CDWO4.

If you operate a phoswich, use the settings for the slower scintillator. Then adjust the phoswich integration time PIT (CR10) to equal the recommended integration time of the faster scintillator. If you perform rise time discrimination, set PIT to equal the longest rise time.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
unsigned short	scintillator	Type of scintillator
unsigned short *	Controls	Report the downloaded Control register values to the calling procedure. There must be room for 28 entries in the Controls array.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(CR[0] .. CR[27]), scintillator ID
dataToHost	(CR[0] .. CR[27])

Control registers affected:

<i>Reg.</i>	<i>Updated parameter</i>
CR3	DC-Wait
CR4	Integration time
CR5	Start delay (for triggered trace capture)
CR10	Phoswich integration time
CR11	Pile up time
CR12	Energy and phoswich scaling values

Returns 0 if successful, -1 if qWR_SP_Controls fails, and -2 if an unsupported scintillator was selected. In the latter case, the function takes no action.

6.15 qM_SP_Parse_AR

This function breaks out all bit fields in a given SP-FPGA action register, including areas marked as unused, and stores them sequentially in a pattern array.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Actions	Points to an array of four 16-bit SP-FPGA action registers
USHORT *	pattern	Points to an array of sixteen 16-bit words to receive the bit-field values.
USHORT	numRegister	Identifies the action register to be parsed.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3]), numRegister
dataToHost	pattern[0] .. pattern[15]

Table 16: Usage of I/O arrays when using the unified command interface. (CR[0]..CR[27]) denotes the array of 28 control registers.

Calls:	-
Called by:	All functions issuing DAQ commands, clearing data and programming the HV. These are found in qMorpho_Operation.c

Returns 0 if successful and -1 if *numRegister* was illegal.

6.16 qM_SP_Assemble_AR

This function receives a pattern array and constructs the content of a single action register.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	Actions	Points to an array of four 16-bit SP-FPGA action registers
USHORT *	pattern	Points to an array of sixteen 16-bit pattern words.
USHORT	numRegister	Identifies the action register to be built.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(pattern[0] .. pattern[15]), numRegister
dataToHost	(AR[0] .. AR[3])

Calls:	-
Called by:	All functions issuing DAQ commands, clearing data and programming the HV. These are found in qMorpho_Operation.c

Multiple calls to qM_SP_Assemble_AR can be used to assemble the four action registers one by one.

Returns 0 if successful and -1 if *numRegister* was illegal.

7 Operations (qMorpho_Operations.c)

This is a collection of functions used to start and stop the data acquisition, to set the controlling parameters and to perform some useful conversion functions.

Most operations are commands calling for some action. There are one-time actions such as reprogramming the HV and clearing counters and there are continuous actions such as starting a particular data acquisition.

Action commands can be directed at a single channel or broadcast to more than one. With the exception of the qM_SP_Program_HV functions, all commands only produce a single write to the action registers. The standard functions directed at the SP_FPGA accept all four sets of action registers as an input and on return will update the action registers according to the broadcast channel mask (*addr.channel*) to accurately reflect the updates as they occurred in the SP_FPGA. The standard function will look at the channel mask and choose the least significant bit that is set to determine which set of action registers to download/broadcast.

The “*_Simple” version of each function only accepts one set of action registers and will download/broadcast it according to the value of the channel mask.

7.1 qM_SP_Program_HV

This function programs up to four high-voltage setting DACs on a qMorpho one-by-one. Unlike the other command functions in this sections, this function does issue separate writes, because a broadcast is not physically possible here. The channel mask (*addr.channel*) determines the channels for which the HV is updated.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	<i>addr</i>	Target address Uses: <i>iface</i> , <i>devNum</i> , <i>slotNum</i> , <i>channel</i>
USHORT *	<i>in_Actions</i>	Points to an array of four sets of four 16-bit SP-FPGA action registers; <i>ch0</i> is the first set.
USHORT *	<i>out_Actions</i>	Points to an array of four sets of four 16-bit SP-FPGA action registers; <i>ch0</i> is the first set.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
<i>dataFromHost</i>	4 x (AR[0] .. AR[3])
<i>dataToHost</i>	4 x (AR[0] .. AR[3])

Control registers affected: none

In the *dataToHost*, or *out_Actions*, array the function reports the values written to the SP_FPGA action registers. Note, however, that the SP_FPGAs do clear the Action register

number 0 (AR[0]) immediately after the command has been received.

It is assumed that in a previous write to the control registers the content DACvalue of CR7 has been updated. This function will program the DACs of the addressed channels with the value stored in CR7.

The output voltage of the DAC is given by $V_{out} = DACvalue/4095 \cdot 3.000 V$.

7.2 qM_SP_Program_HV_Simple

This function reprograms the high-voltage setting DACs for a single channel on a qMorpho.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT *	out_Actions	Points to an array of four 16-bit SP-FPGA action registers.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3])
dataToHost	(AR[0] .. AR[3])

In the dataToHost, or out_Actions, array the function reports the values written to the SP_FPGA action register. Note, however, that the SP_FPGAs will clear the Action register number 0 (AR[0]) immediately after the command has been received.

Only one bit should be set in the addr.channel mask.

It is assumed that in a previous write to the control registers the content DACvalue of CR7 has been updated. This function will program the DACs of the addressed channels with the value stored in CR7.

7.3 qM_SP_Untriggered_Trace

To record an untriggered trace is a one-time action, much like a call for clearing registers, rather than a true data acquisition mode. The function requests the qMorpho to store an untriggered trace in all channels indicated by the channel mask (addr.channel).

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	4 x (AR[0] .. AR[3])
dataToHost	4 x (AR[0] .. AR[3])

In the dataToHost, or out_Actions, array the function reports the values written to the SP_FPGA action registers. Note, however, that the SP_FPGAs do clear the Action register number 0 (AR[0]) immediately after the command has been received.

7.4 qM_SP_UTrace_Simple

To record an untriggered trace is a one-time action, much like a call for clearing registers, rather than a true data acquisition mode. The function requests the qMorpho to store an untriggered trace in one channel as indicated by the channel mask (addr.channel). If more than one bit is set in the mask, the least significant bit set determines the channel.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT *	out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3])
dataToHost	(AR[0] .. AR[3])

In the dataToHost, or out_Actions, array the function reports the values written to the SP_FPGA action registers. Note, however, that the SP_FPGAs will clear the Action register number 0 (AR[0]) immediately after the command has been received.

7.5 qM_SP_Start_DAQ

The basic function to start data acquisition in one or more channels via a single write (broadcast) operation. The value of the channel mask (addr.channel) decides which channels are affected. You can start a new data acquisition in a channel without affecting an ongoing acquisition. For instance, you can call for a trace or list mode acquisition without interrupting an

ongoing histogram acquisition.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT *	out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT	daqType	Specifies the DAQ, namely histogram, list mode, trace.
USHORT	clear	Specifies which data or modules to clear: statistics, histogram, list mode, trace.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	4 x (AR[0] .. AR[3]), daqType, clear
dataToHost	4 x (AR[0] .. AR[3])

daqType: This is a 3-bit wide bit field with one bit for each type of data acquisition: 1 → Histogram, 2 → List mode, 4 → Triggered trace.

Clear: This is a 4-bit wide bit-field to specify which data memory to clear, or which DAQ module to reset so it can start a new data acquisition: 1 → Statistics, 2 → Histogram, 4 → List mode, 8 → Triggered trace.

Note that statistics counters and DAQ modules will only be cleared or reset if their respective clear-enable bits are set, cf the contents of control registers CR15 and CR16.

Given the four sets of action registers, one for each channel, the function returns the updated action registers according to the channel mask as well as daqType and clear.

7.6 qM_SP_Start_DAQ_Simple

The basic function to start data acquisition in one or more channels via a single write (broadcast) operation. The value of the channel mask (addr.channel) decides which channels are affected. You can start a new data acquisition in a channel without affecting an ongoing acquisition. For instance, you can call for a trace or list mode acquisition without interrupting an ongoing histogram acquisition.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four 16-bit SP-FPGA action registers.

<i>Input parameter, type and name</i>		<i>Description</i>
USHORT *	out_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT	daqType	Specifies DAQ, namely histogram, list mode, trace.
USHORT	clear	Specifies which data or modules to clear: statistics, histogram, list mode, trace.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3]), daqType, clear
dataToHost	(AR[0] .. AR[3])

daqType: This is a 3-bit wide bit field with one bit for each type of data acquisition: 1 → Histogram, 2 → List mode, 4 → Triggered trace.

Clear: This is a 4-bit wide bit-field to specify which data memory to clear, or which DAQ module to reset so it can start a new data acquisition: 1 → Statistics, 2 → Histogram, 4 → List mode, 8 → Triggered trace.

Note that statistics counters and DAQ modules will only be cleared or reset if their respective clear-enable bits are set, cf the contents of control registers CR15 and CR16.

Given one set of action registers, the function returns the updated action registers according to daqType and clear.

7.7 qM_SP_Stop_DAQ

The basic function to stop a particular data acquisition in one or more channels via a single write (broadcast) operation. The value of the channel mask (addr.channel) decides which channels are affected. You can stop an ongoing data acquisition in a channel without affecting other ongoing acquisitions. For instance, you can end a trace or list mode acquisition without affecting an ongoing histogram acquisition.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT *	out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT	daqType	Specifies the DAQ, namely histogram, list mode, trace.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
---------------------------------	--

dataFromHost	4 x (AR[0] .. AR[3]), daqType
dataToHost	4 x (AR[0] .. AR[3])

daqType: This is a 3-bit wide bit field with one bit for each type of data acquisition: 1 → Histogram, 2 → List mode, 4 → Triggered trace.

Given the four sets of action registers, one for each channel, the function returns the updated action registers according to the channel mask and daqType.

7.8 qM_SP_Stop_DAQ_Simple

The basic function to stop a particular data acquisition in one or more channels via a single write (broadcast) operation. The value of the channel mask (addr.channel) decides which channels are affected. You can stop an ongoing data acquisition in a channel without affecting other ongoing acquisitions. For instance, you can end a trace or list mode acquisition without affecting an ongoing histogram acquisition.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT *	out_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT	daqType	Specifies the DAQ, namely histogram, list mode, trace.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3]), daqType
dataToHost	(AR[0] .. AR[3])

daqType: This is a 3-bit wide bit field with one bit for each type of data acquisition: 1 → Histogram, 2 → List mode, 4 → Triggered trace.

Given a set of action registers, the function returns the updated action registers according to the channel mask and daqType.

7.9 qM_SP_Suspend_DAQ

Use this function to halt time. Although the onboard clock does not actually stop and the built-in pulser still operates, all other pulse acquisition and counting activity halts.

<i>Input parameter, type and name</i>	<i>Description</i>
qM_ADDRESS * addr	Target address Uses: iface, devNum, slotNum, channel
USHORT * in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT * out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	4 x (AR[0] .. AR[3])
dataToHost	4 x (AR[0] .. AR[3])

Given the four sets of action registers, one for each channel, the function returns the updated action registers according to the channel mask.

7.10 qM_SP_Suspend_DAQ_Simple

Use this function to halt time. Although the onboard clock does not actually stop and the built-in pulser still operates, all other pulse acquisition and counting activity halts.

<i>Input parameter, type and name</i>	<i>Description</i>
qM_ADDRESS * addr	Target address Uses: iface, devNum, slotNum, channel
USHORT * in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT * out_Actions	Points to an array of four 16-bit SP-FPGA action registers.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3])
dataToHost	(AR[0] .. AR[3])

Given one set of action registers, the function returns the updated action registers.

7.11 qM_SP_Resume_DAQ

Use this function to resume a suspended DAQ. In response to a resume command, the statistics counters become enabled again and any active data acquisition continues from where it was halted with the suspend command as if no time had passed in between.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT *	out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	4 x (AR[0] .. AR[3])
dataToHost	4 x (AR[0] .. AR[3])

Given the four sets of action registers, one for each channel, the function returns the updated action registers according to the channel mask.

7.12 qM_SP_Resume_DAQ_Simple

Use this function to resume a suspended DAQ. In response to a resume command, the statistics counters become enabled again and any active data acquisition continues from where it was halted with the suspend command as if no time had passed in between.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
USHORT *	in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT *	out_Actions	Points to an array of four 16-bit SP-FPGA action registers.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3])
dataToHost	(AR[0] .. AR[3])

Given one set of action registers, the function returns the updated action registers.

7.13 qM_SP_Clear_Stats

Use this function to clear the statistics counters and start counting run time, life time, events, triggers and trigger dead time afresh. Note that the statistics counters will only be cleared the clear-enable bits in the target channels have been set, cf the contents of control register CR16.

<i>Input parameter, type and name</i>	<i>Description</i>
qM_ADDRESS * addr	Target address Uses: iface, devNum, slotNum, channel
USHORT * in_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.
USHORT * out_Actions	Points to an array of four sets of four 16-bit SP-FPGA action registers; ch0 is the first set.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	4 x (AR[0] .. AR[3])
dataToHost	4 x (AR[0] .. AR[3])

Given the four sets of action registers, one for each channel, the function returns the updated action registers according to the channel mask.

7.14 qM_SP_Clear_Stats_Simple

Use this function to clear the statistics counters and start counting run time, life time, events, triggers and trigger dead time afresh. Note that the statistics counters will only be cleared the clear-enable bits in the target channels have been set, cf the contents of control register CR16.

<i>Input parameter, type and name</i>	<i>Description</i>
qM_ADDRESS * addr	Target address Uses: iface, devNum, slotNum, channel
USHORT * in_Actions	Points to an array of four 16-bit SP-FPGA action registers.
USHORT * out_Actions	Points to an array of four 16-bit SP-FPGA action registers.

<i>I/O array of type USHORT</i>	<i>Contents when using the unified command interface</i>
dataFromHost	(AR[0] .. AR[3])
dataToHost	(AR[0] .. AR[3])

Given one set of action registers, the function returns the updated action registers.

7.15 qM_SP_Get_Calibration

Use this function to receive the calibration coefficients, namely the conversion from raw ADC numbers into meaningful quantities such as PMT anode current and charge per histogram bin.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: iface, devNum, slotNum, channel
unsigned short *	Controls	Control register array for this device. There must be room for 28 entries in this array, since the function will report back the new control register contents.
unsigned short *	data	Array in which the coefficients will be reported. There must be room for 15 entries.
float	ADC_SPS	ADC sampling rate in samples per second.

Control registers affected: none.

The content of the data array will be:

<i>Item</i>	<i>Description</i>
data[0]	DC-offset in units of ADC_Full_Range / 64, where the full range corresponds to 1023 bins.
data[1]	TwinBase temperature in units of 1/16 th degree C.
data[2] data[3]	PMT anode current per ADC step in units of 1.0e-12 A; low word in data[2], high word in data[3].
data[4] data[5]	Transimpedance (gain), in Ohms, of the analog gain stage; low word in data[4], high word in data[5].
data[6]	Maximum amplitude in ADC steps
data[7] data[8]	Maximum PMT anode current in 1.0e-9 A; low word in data[7], high word in data[8].
data[9] data[10]	PMT anode current resolution in 1.0e-12 A; low word in data[9], high word in data[10].
data[11] data[12]	Incremental charge integral per ADC step and time step in units of 1.0e-21 C; low word in data[11], high word in data[12].
data[13] data[14]	Charge per histogram bin in units of 1.0e-18 C; low word in data[13], high word in data[14].

Returns 0 if successful, and -1 or -2 if one of the data array pointers was NULL.

7.16 qM_SP_Get_Rates

This function converts the data obtained from a call to qM_SP_Read_Statistics into meaningful input/output count rates and data acquisition run times. It receives the raw statistics data (via qRD_SP_Statistics) and adds the evaluated data after that in the *data* array.

<i>Input parameter, type and name</i>		<i>Description</i>
qM_ADDRESS *	addr	Target address Uses: interface, devNum, slotNum, channel
unsigned long *	data	Raw counter data and computed results are stored in this array. There must be room for 18 entries.

<i>Data word</i>	<i>Name</i>	<i>Description</i>
data[0] data[1]	RT_low RT_high	Time since the statistics counter was last cleared. 1 LSB = 65536 clock ticks.
data[2] data[3]	Events_low Events_high	Number of accepted events since the statistics counter was last cleared.
data[4] data[5]	Triggers_low Triggers_high	Number of recognized triggers since the statistics counter was last cleared.
data[6] data[7]	DT_low DT_high	Measured dead time since the statistics counter was last cleared. 1 LSB = 65536 clock ticks.
data[8] data[9]	RunTime_low RunTime_high	Time since the statistics counter was last cleared, in milliseconds.
data[10] data[11]	EventRate_low EventRate_high	Rate of accepted events, in units of 0.001 counts per second (cps)
data[12] data[13]	Trigger_low Trigger_high	Rate of recognized triggers, in units of 0.001 counts per second (cps)
data[14] data[15]	DTF_low DTF_high	Dead time fraction * 1.0e6
data[16] data[17]	PulseRate_low PulseRate_high	Estimated rate of incoming pulses assuming randomness and Poisson statistics, in units of 0.001 counts per second (cps)

7.17 HV2DAC

This function computes the appropriate high-voltage DAC value for a required photomultiplier tube (PMT) high voltage. The function returns an unsigned long value.

The function limits the returned DAC value to 2730 to ensure that the requested PMT high voltage will never exceed 2000 V.

<i>Input parameter, type and name</i>		<i>Description</i>
double	HV	Required high voltage

7.18 DAC2HV

This function computes the photomultiplier tube (PMT) high voltage corresponding to the high-voltage DAC setting. The function returns a double.

<i>Input parameter, type and name</i>		<i>Description</i>
unsigned long	DAC	Value programmed into the high-voltage controlling DAC.

8 Helper functions

8.1 SetBit, ClearBit, ToggleBit

These functions are used to selectively manipulate a single bit in an unsigned long value.

<i>Input parameter, type and name</i>		<i>Description</i>
unsigned long	num	Number of the bit to be manipulated. Enumeration starts at 0.
unsigned long	value	Datum whose content is to be manipulated.

9 Revision History

WP0: Work in progress