Command And Data Interface For Morpho-Based MCAs

1 Operating the instrument

The Morpho instruments are programmed for autonomous data acquisition, receiving occasional commands from a remote controller. The controller communicates its requests via macro-commands that carry enough information to let the instrument proceed with the data acquisition and send data to the controller, or a data aggregator, when required. To eliminate the need for polling, the instrument is able to send data to the controller when they become available, without the controller having to issue a read request. Data sent upstreams are preceded with a data header that tells the controller exactly what type of data are being sent, what their format is, and which instrument they originate from.

1.1 Data exchange

To facilitate communication between different computer systems, multi-byte datums have to have their bytes arranged in proper order for the target computer to make sense. Internet traffic uses big-endian ordering and two-byte, four-byte and eight-byte datums have to be treated separately when converting from or to little-endian environments. Hence, some care is required when sending mixed-data type arrays.

Instruments based on ARM-9 processors will send and receive Ethernet data in big-endian convention. MSP430-based instruments are little-endian machines with limited computing resources. To save time, bulk data are sent and received in little-endian convention (LSB first), leaving the required conversions to the upstreams computer.

Commands sent from the controller to the instrument consist of a fixed-format header and command-specific data. Data sent by the instrument, consist of a fixed-format header with identification and formatting information for the data block that follows. Note that the instrument may send data to the controller either in response to a read command or autonomously in periodic intervals or in response to an event.

2 Command structure

2.1 Command header

The command header is an array of eight 16-bit words. The first three are used to prepare the next read operation. Entries at index 3 and 4 are used for reads and writes. The entries at 5 – 7 describe a command.

Index	Name / offset	Description
0	CH_CTRL	16-bit bit-field: First byte, bit 0: endianess of remainder of command header; bit 1: endianess of data following the header. Bit-2: Set to program instrument to adopt the device number given in field CH_RW_INSTRNUM (only in setup command).
1	CH_READ_SEL	Select data group (upper byte) and the data array (lower byte) for a subsequent read; only used in a read command, ignored otherwise.
2	CH_READ_FIRST	Offset (in bytes) where the read begins.
3	CH_RW_DEVNUM	Either 16-bit device number or 8-bit device number in lower byte and 8-bit channel number in upper byte (in multi-channel systems, such as qMorpho). A value of 255 in either byte indicates a broadcast.
4	CH_RW_INSTRNUM	Instrument number
5	CH_CMD_GROUP	Command group (DAQ, peripheral, CPU); see the known groups listed below
6	CH_CMD_COMMAND	The command number
7	CH_CMD_ACTIONS	Reserved
8	CH_CMD_DATA	Command data in indices 8 .. INSTRUMENT_COMMAND_SIZE-1, typically 255.

The CH_CTRL bit-field. This field is always in little endian format.

15							**8**	**7**							**0**
													AN	ED	EH

EH: 0 → command header is little endian; 1 → command header is big endian;

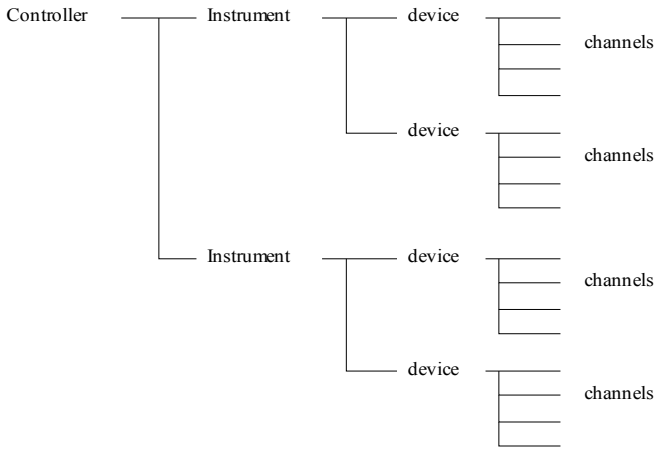
ED: 0 → command data are little endian; 1 → command data are big endian;

AN: 0 → no action; 1 → instrument adopts ID as given in field CH_RW_INSTRNUM as myDeviceNumber (used in setup commands only).

Known command groups:

Index	Name	Description
0	CMD_DONE	Ready; no command pending
1	CMD_DAQ	Data acqusition command
2	CMD_SPI	Command related to a device on the SPI bus (high voltage DAC, temperature sensor, battery monitor, NVRAM, RTC, etc)
3	CMD_CPU	Command affecting CPU registers (timers, clock speed, etc).
4	CMD_ICOM	Instrument communication commands (reset)
5	CMD_CCOM	Computer communication commands (informative for the instrument; for instance: intent to close connection)
6	CMD_API	GUI/User to controller API command

2.2 Communication model



The software recognizes individual instruments that are connected to a remote controller via some communication bus on which they each occupy a distinct address.

An instrument may consist of more than one device and each device may have more than one MCA & DAQ channel.

In the case of an Ethernet MCA base, each instrument includes one ARM-9 processor (instrument) and one 1-channel eMorpho (device). In the case of a qDAQ-4 each instrument includes one Arm-9 processor (instrument), 2 qMorpho boards (devices) with 4 MCA channels each.

In all the above cases, the controller is a computer connecting to the instruments via Ethernet/Internet. For the ultra low-power RDR, the instrument consists of an MSP430 MCU and a single-channel tinyMorpho. In this case, the controller may be an embedded ARM-9 processor connecting via an 8-bit handshake bus or a computer connecting via USB.

Moving data and commands between the components is achieved by defining command and data structures that carry sufficient information for routing commands to the target and associating data that have been sent back to the controller, or other data aggregator, with the correct source instrument/device and channel. In addition data characteristics such as formatting (little or big endian, number of bytes per datum, integer or float) are indicated in the data headers.

2.3 Communication method

The method of communication depends on the physical link. In a connection-oriented protocol (TCP/IP) the instruments employs a never-ending main loop to listen for a command from the controller and execute it. It will only be ready to accept the next command when the previous one has been processed. When data become available, the instrument initiates a connection and sends data to the controller.

The very low-power MSP430-based instruments are meant to operate mostly autonomously, with all operating software being alive after power on. Data can be stored in on board NV-RAM and be retrieved via USB or an 8-bit parallel hand shake bus. In either case the communication is not connection oriented and the controller has to read status information first to ensure the unit is ready for the next command.

To prepare a read, the controller writes at least one 16-bit word to the command array, which contains the index of where the read will begin,. Subsequent reads will always commence from the same address unless the controller first changes the value of the first entry into the command array.

The controller should not write more than four words to avoid overwriting the currently executed command and its arguments. The write pointer is always reset to the start of the write area.

To write a command to the instrument, the controller writes at least 8 words, and more if the command requires any accompanying data or arguments.

The instrument's main polling loop checks the command group and command once every turn. All commands are blocking commands in the sense that the function does not return until the command has been executed. For commands that call for any type of data acquisition, the function will return when the DAQ has started. Upon completion, the index CH_CMD_COMMAND in the command header is cleared.

The interrupt routine that receives the command from the controller will check if index CH_CMD_COMMAND in the command header has been cleared. If it is not zero, it will not accept the incoming command. If possible (not for USB), the instrument will send an error message back in this case. Alternatively the controller can read the instrument status data before sending a new command. However, it should be noted that high-frequency polling will slow down the instrument. It is best for the controller to await the arrival of the command-status message.

3 DAQ commands

DAQ commands are used to control the Morpho MCA&DAQ module as well as instrument data acquisition actions.

3.1 The DAQ_CMD_SETUP command

This command is used to generate a predefined setting and download it to the Morpho MCA unit. The settings parameters can be read back SI (and human readable) units . There are are up to 7 predefined settings to choose from. If this option is chosen, the controller need only send the first word of the command's data array, plus an optional high-voltage value

Alternatively, the controller can send data to the instrument to be converted into the correct bit pattern (set of 16 control registers) and to be downloaded into the Morpho MCA.

3.1.1 Data and arguments for the SETUP command

Data and arguments for this command start at location CH_CMD_DATA [8] into the command array. The four arguments are 16-bit words:

Index	Name	Description
0	DSETUP_SELECT	Bit field [2:0]: if ≠ 0 it is the index of a standard setting to be loaded; if 0 setup data will follow below. Bit [4:3]: type of setup data: 0 → float data, 1 → discrete data, 2 → bit patterns, Bit 5: 1 → HV alone follows as float data
1	DSETUP_ACTIONS	bit 0: if 1 download new setup to Morpho; bit 1: 0 → MCA OFF, 1 → MCA ON bit 2: 0 → HV OFF, 1 → HV ON;
2	DSETUP_R1	Reserved
3	DSETUP_R2	Reserved
4	DSETUP_DATA	Begin of setup data

There are different options to configure the MCA. The most simple is to choose one of the up to 7 predefined settings. This is encoded in bits 0-2 in DSETUP_SELECT. If the field is zero, the instrument expects the setup data to follow beginning at location 4 After the fact, these values can be read back from the data space using one of the selectors: DH_TYPE_SETUP_F, DH_TYPE_SETUP_I, DH_TYPE_SETUP_C. Alternatively, the controller can download the required setup data.

The DSETUP_SELECT bit-field

15							**8**	**7**							**0**
										XV	FMT		SEL		

SEL: 0 → don't use a predefined setup; actual setup data will follow; 1 through 7 → use the indicated predefined setup.

FMT: type of setup data: 0 → float data, 1 → discrete data, 2 → bit patterns (MCA control registers),

There are three choices for the data format of the setup data, encoded in bits 3-4 of DSETUP_SELECT. Data of the type float express all setup parameters for the MCA in SI-units for physical quantities and pure numbers for all others. The data type used throughout is float. Discrete data use 16-bit numbers as the MCA uses them (e.g. times are measured in sampling clock ticks, etc). Method no. 2 (bit-patterns) uses an array of 16 tightly-packed control registers. This is the most hardware-dependent method, while method no. 0 (float data) is the most hardware independent method. Clearly, method 0 is preferable in most cases.

When the controller sends a setup command, the setup data are is copied into the three setup data arrays, carrying the same information in three increasingly machine-dependent formats.

XV: 1 → high voltage follows in a data field; used with SEL > 0 to override the preset voltage. The high voltage format is determined by the contents of FMT field: 0 → float32 datum, 1 → uint16 datum.

3.1.2 Setup data in float format, long form:

When interpreting this data field as an array of floating point numbers, create a (float) pointer to its beginning and read the data as floats, which means an index increment of 1 is worth 4 bytes.

Index	Name DSETUP_xx	Description
0	GAIN_FINE	Fine gain; 2.0> x ≥ 0
1	GAIN_ECOMPRESS	MCA bin compression; $2^{15} \geq x \geq 1$
2	GAIN_ANALOG	1100 (tinyMorpho); {100, 430, 1100, 3400, 10100} (eMorpho, qMorpho)
3	GAIN_HV	High voltage; $1800 \geq x \geq 1$
4	GAIN_PCOMPRESS	PID bin compression; $2^{15} \geq x \geq 1$
5	E_TRIGGER	Energy trigger; $100 \geq x \geq 0.1$ (in % of FSR)
6	E_IT	Integration time; in seconds
7	E_PUT	Pileup threshold; $65535 \geq x \geq 0$; pure number; x=0 turns it off.
8	E_PIT	Partial integration time; in seconds (for n/γ discrimination)
9	E_NTRIGGER	Noise trigger $100 \geq x \geq 0.1$ (in % of FSR)
10	P_ENABLE	Pulser enable; $x = 0,1$
11	P_TRIGGER	Trigger on pulser; $x = 0,1$
12	P_PERIOD	Pulse period; in seconds
13	P_WIDTH	Pulse width in seconds
14	P_SEP	Double pulse separation

3.1.3 Setup data in discrete format, long form:

In this format, the entries are indeed 16-bit unsigned numbers and an index increment of 1 corresponds to 2 bytes.

Index	Name DSETUP_xx	Description
0	TYPE	Bit field [2:0]: if ≠ 0 it is the index of a standard setting to be loaded; if 0 setup data will follow below. Bit [4:3]: type of setup data: 0 → float data, 1 → discrete data, 2 → bit patterns; Bit [5]: 0 → short form, 1 → long form
1	ACTIONS	bit0: if 1 load new setup; bit 1: 0 → MCA OFF, 1 → MCA ON bit 2: 0 → HV OFF, 1 → HV ON;

2	GAIN_FINE	Fine gain; $65535 \geq x \geq 0$
3	GAIN_ECOMPRESS	MCA bin compression; $2^{15} \geq x \geq 1$
4	GAIN_ANALOG	0 (tinyMorpho); {0,1,2,4,8} (eMorpho, qMorpho)
5	GAIN_HV	High voltage; $2450 \geq x \geq 0$
6	GAIN_PCOMPRESS	PID bin compression; $2^{15} \geq x \geq 1$
7	E_TRIGGER	Energy trigger; $1023 \geq x \geq 1$
8	E_IT	Integration time; $65535 \geq x \geq 5$ (clock cycles)
9	E_PUT	Pileup threshold; $65535 \geq x \geq 0$; pure number; x=0 turns it off.
10	E_PIT	Partial integration time; $65535 \geq x \geq 5$ (clock cycles)
11	E_NTRIGGER	Noise trigger $1023 \geq x \geq 1$
12	P_ENABLE	Pulser enable; $x = 0,1$
13	P_TRIGGER	Trigger on pulser; $x = 0,1$
14	P_PERIOD	Pulse period; $31 \geq x \geq 1$, $p = 2^{(x+1)}$ clock cycles
15	P_WIDTH	Pulse width, $15 \geq x \geq 1$, $p = 2^{(x+1)}$ clock cycles
16	P_SEP	Double pulse, $15 \geq x \geq 1$, $p = 2^{(x+1)}$ clock cycles

3.1.4 Setup data as bit-patterns:

This is the content of the control registers, cf the FPGA-Map document for a detailed description.

3.2 The DAQ_CMD_SCAN command

This command is used to program the instrument to collect count rate data and histogram data with preset periods. The two collection periods need not be the same. A mode argument allows the controller to control the details of the operation, including histogram memory splitting, auto-reporting and turning the MCA on and off. Note that the RDR1 can turn the MCA off, but keep the PMT powered and use only the MCU and the CPLD to measure count rates in the detector crystal – at less than 2mA power consumption.

Entries in this data array are all floats. Hence the instrument will aim a float pointer at the beginning of this data array.

Index	**Name DSCAN_xx**	**Description**
0	STATS_PERIOD	Period at which statistics are updated in instrument memory
1	HISTO_PERIOD	Period at which histograms are updated in instrument memory

2	MODE	Bit-0: Enable histogram bank switching (histo and stats period must be equal) Bit 1: Enable auto reporting (Instrument initiates communication) for statistics Bit 2: Enable auto reporting for histograms Bit-3: Clear statistics after update Bit-4: Clear histogram after update Bit-5: Set to stop DAQ Bit-6: 0 → histogram energies, 1 → histogram pulse heights (diagnostic only);
3	CH_PATTERN	Bit field; Pattern of channels to activate in this radiation scan. Ignored in systems with only one channel total.

This command allows for scanning for radiation, with the ability to react more quickly to a change in count rate than to finding a significant peak in the histogram.

Alternatively, this command can be used to collect histogram and count rate data (without clearing) cumulatively over a long period of time while saving the data ever so often on the controller side. This can be used to monitor slow changes in the environment or monitor gain changes as the data are being accumulated.

Set STATS_PERIOD to zero to suppress acquiring and reporting of statistics data by the instrument. The same applies to the HISTO_PERIOD.

3.3 The DAQ_CMD_HISTO command

This is a macro-command for acquiring a sequence of histograms while varying one parameter. It can also be used to acquire a single histogram. Its argument are in float32 format. Below follows a list of its arguments:

Index	Name DHISTO_xx	Description
0	TYPE	Bit 0: 0 → energy, 1→ amplitude
1	TIME	Acquisition time for each histogram
2	FIRST_VALUE	Start-value for the parameter that is to be varied
3	INCR_VALUE	Parameter increment
4	LAST_VALUE	Last parameter value
5	FIRST_FUNC	Index of a user hook function to be called before the first histogram acquisition
6	MIDDLE_FUNC	Index of a user hook function to be called after each histogram acquisition, except the last

7	LAST_FUNC	Index of a user hook function to be called after the last histogram acquisition
8	CH_PATTERN	Bit field; Pattern of channels to activate in this radiation scan. Ignored in systems with only one channel total.

The first argument, TYPE is a 16-bit wide bit-field:

15							**8**	**7**							**0**
								PAR			DA	CL	STOP		A

A: Histogram 0 → energy, 1→ amplitude

STOP: Stop condition 0 → infinite, 1 → RT, 2 → LT, 3 → counts

CL: 1 → Clear histogram memory and the statistics counters in the MCA.

DA: 0 → DAQ idle, 1 → DAQ active. When active, the MCA updates the statistics registers only when a histogram acquisition is in progress, and freezes the counters when done. This way count-rates can be accurately read, even by a slow micro-controller.

PAR: Select the parameter to vary. 0 → none, 1 → high voltage, 2 → integration time, 3 → pileup recognition parameter

3.4 The DAQ_CMD_LM command

This command is used to acquire list mode data.

Index	**Name DLM_xx**	**Description**
0	TYPE	Bit 0: type switch Bit 2 – 3: Choice of event validation source (for instance, external triggers or coincidences). Usually set to 0.
1	NUM_BUFFERS	Number of list mode buffers requested. Writing a 0 will end an ongoing macro-command for list mode acquisition
2	FIRST_FUNC	Index of a user hook function to be called before the first list mode acquisition
3	MIDDLE_FUNC	Index of a user hook function to be called after each list mode acquisition, except the last
4	LAST_FUNC	Index of a user hook function to be called after the last list mode acquisition
5	CH_PATTERN	Bit field; Pattern of channels to activate in this acquisition. Ignored in systems with only one channel total.

Bit 0 in the TYPE field selects one of the two list mode acquisition modes: 0 → acquire 32-bit time and 16-bit energy; 1 → acquire 16-bit time, 16-bit PID value, and 16-bit energy.

The first, middle and last function calls are calls to user hook functions to be exercised at the proper time. These functions may for instance implement a simple analysis and store only data summaries.

3.5 The DAQ_CMD_TRACE command

This command is used to acquire trace data. The entries are of type uint16.

Index	Name DTRACE_xx	Description
0	TYPE	Bit 0-1: 0 → untriggered trace, 1 → triggered trace, 2 → validated trace. Bit 2 – 3: Choice of validation source for validated traces (for instance, external triggers or coincidences). Usually set to 0.
1	NUM_BUFFERS	Number of traces requested. Writing a 0 will end an ongoing macro-command for trace acquisition
2	FIRST_FUNC	Index of a user hook function to be called before the first list mode acquisition
3	MIDDLE_FUNC	Index of a user hook function to be called after each list mode acquisition, except the last
4	LAST_FUNC	Index of a user hook function to be called after the last list mode acquisition
5	CH_PATTERN	Bit field; Pattern of channels to activate in this acquisition. Ignored in systems with only one channel total.

The type field determines the type of traces to be acquired.

The first, middle and last function calls are calls to user hook functions to be exercised at the proper time. The functions may implement data analysis to select only certain traces for storage and sending back to the controller.

4 Data structures and fields

Within the instrument's memory space, data are organized into a number of arrays of different data types, mostly unsigned integers and floats. All data sent to the controller are preceded by a typically 12-byte long header, which identifies data types and formatting, so that the controller can tell what the data are and how to decode them. The data header is of type **uint8** (unsigned char). The data header has to be examined byte-by-byte with the exception of bytes 6 – 9. These form a four-byte unsigned long in network byte order, i.e. big endian.

4.1 The data header

The data header is a character array, where most information is contained in single bytes. Only the instrument ID and the number-of-bytes entry have to be interpreted as a multi-byte datum.

Index	Name	Description
0	DH_LEN	Length of header in bytes;
1	DH_FORMAT	Format of data to follow (int, float, etc)
2	DH_GRP	Data group, same as the command group that created the data
3	DH_TYPE	Data type (rates, histograms, list mode, etc)
4	DH_DEVNUM	Device number
5	DH_CHNUM	Channel number
6 – 7	DH_INSTRNUM	16-bit item: Instrument number (big-endian)
8 - 11	DH_NUM_BYTES	32-bit item: number of bytes in the data block, following the header (in network byte order; ie big endian)

The first six bytes need to be examined by the controller in the order received, and not be interpreted as integers. Their byte ordering must not be changed.

Byte 0 contains the length of the data header in bytes.

Byte 1 is a bit field with the following content:

7	6	5	4	3	2	1	0
Endianess		Sequence		Format			

Format: Describes the data format of the data block following the data header. 0 → char; 1 → unsigned char, 2 → short (16-bit), 3 → unsigned short, 4 → long (32-bit), 5 → unsigned long, 6 → xlong (64-bit), 7 → unsigned xlong, 8 → float (32-bit), 9 → double (64-bit).

Sequence: Set to zero when the controller initiated the read or when the data are not part of a sequence. When the instrument is programmed, for instance, to send a sequence of histograms in predetermined time intervals, the sequence field will be 1 for the first histogram, 2 for any in the middle and 3 for the last histogram. The controller can use this information to take appropriate action, e.g. open a data file,

adding to it, and then close the data file after the last transmission.

Endianess: 0 → little-endian: LSB is sent and received first; 1 → big-endian: MSB is sent and received first. The endianess applies to the data following the header.

Byte 2 directs the data read to a particular data array within a data group, e.g. version data, calibration data, bulk data, etc. See list and content descriptions below.

Byte 3 in the data header selects the data group. This group number is the same as the group number of the command that created the data that are being sent. Currently there are three groups: CMD_DAQ → for MCA data acquisition and data analysis, CMD_SPI → for commands communicating with peripherals on the SPI bus (e.g HV-DAC, temperature sensor, NVRAM, RTC, battery voltage monitor); CMD_CPU → for commands reprogramming CPU registers (e.g to change clock speed).

Byte 4 has the device number for systems with more than one instrument. The device number has previously been assigned to the instrument by the controller.

Byte 5 is used as either the higher byte of the controller-assigned device ID or is used as a channel number for instruments with more than one channel, such as qDAQ systems.

Byte 6,7 are reserved for future use.

The data header concludes with a 32-bit item that indicates how many bytes there are in the data block to follow.

4.2 Group CMD_DAQ: Overview

The following data arrays belong to the DAQ group:

Value	Name	Data type	Description
0	DH_TYPE_STATUS	uint16	Instrument status
1	DH_TYPE_VERSION	uint16	Hardware / software / firmware versions and builds
2	DH_TYPE_CALIBRATION	float	Calibration; clock speeds, gains, ADC FSR etc.
3	DH_TYPE_SETUP_F	float	Setup data in floating point format for MCA
4	DH_TYPE_SETUP_I	uint16	Setup data in uint16 format for MCA
5	DH_TYPE_SETUP_C	uint16	Setup data in uint16 control registers for MCA
6	DH_TYPE_RATES	float	raw statistics and evaluated count rate data
7	DH_TYPE_HISTO	uint16 uint32	Histogram data; the FORMAT field in the data header tells the format
8	DH_TYPE_HISTO1	uint16 uint32	Histogram data, second bank
9	DH_TYPE_LM	uint16	List mode data;
10	DH_TYPE_TRACE	uint16	Trace data;

4.3 Group CMD_DAQ: Instrument status

This uint16 daq_ds_instrument data block consists of three entries:

Index	Name	Description
0	IN_STATUS	Status register
1	IN_TEMP	Temperature; instrument dependent format
2	IN_BATTVOLT	Battery voltage; instrument dependent format

The status register is a 16-bit bitfield; content to be determined

15							**8**	**7**							**0**

4.4 Group CMD_DAQ: Version data

This is a uint6 data array:

Index	Name	Description
0	SYS_HW_VERSION	Hardware version
1	SYS_HW_BUILD	Standard or non-standard build
2	SYS_SW_VERSION	Software version
3	SYS_SW_BUILD	Standard or non-standard build
4	SYS_CPLD_VERSION	CPLD version
5	SYS_CPLD_BUILD	CPLD build
6	SYS_MCA_DEVICE	MCA device
7	SYS_MCA_VERSION	MCA firmware version
8	SYS_MCA_BUILD	MCA firmware build

SYS_MCA_DEVICE: 0 → eMorpho square; 1 → eMorpho round, industrial temperature range, 2 → qMorpho (4-channel), 3 → tinyMorpho

4.5 Group CMD_DAQ: Calibration data

This float32 array contains mostly MCA calibration and size data

Index	Name	Description
0	MCA_ADC_SPEED	ADC digitizing speed in MSPS

1	MCA_ADC_BITS	Number of ADC bits
2	MCA_ADC_FSR	ADC full-scale range, in volts
3	MCA_GAIN	Transimpedance value in Ω
4	MCA_CUR_FSR	Anode current full scale range, in A
5	MCA_CUR_MAX	Anode current full scale range, in A corrected for DC-offset
6	MCA_CHARGE_UNIT	Charge per MCA bin
7	MCA_DC_OFF	DC offset, in ADC bins (0-1023)
8	MCA_TEMP	Temperature in deg. C; populated when MCA read temperature from HV unit.
9	MCA_ANODE_CUR	Average anode current, in A
10	MCA_ROI_AVG	MCA_ROI_AVG
11	MCA_HISTO_BYTES	Histogram memory size in bytes
12	MCA_LM_BYTES	List mode memory size in bytes
13	MCA_TRACE_BYTES	Trace memory size in bytes
14	MCA_MA6_BYTES	Module 6 memory size in bytes
15	MCA_MA7_BYTES	Module 7 memory size in bytes

4.6 Group CMD_DAQ: Setup data

The read-back MCA setup data have the same format as the corresponding command inputs described in the command section.

4.7 Group CMD_DAQ: Count rates

This float32 array contains the raw statistics counter values as well as the evaluated count rate data:

Index	Name	Description
0	CR_REAL_TICKS	Real time since last clear_stats command in ticks (65536/ADC_Sampling_Rate)
1	CR_EVENTS	Number of accepted events since last clear_stats command
2	CR_TRIGGERS	Number of recognized triggers since last clear_stats command
3	CR_DEAD_TICKS	Trigger dead time since last clear_stats command in ticks (65536/ADC_Sampling_Rate)

4	CR_REAL_TIME	Real time since last clear_stats command in units of seconds
5	CR_EVENT_RATE	Rate of accepted events; 1/s
6	CR_TRIGGER_RATE	Rate of recognized triggers;1/s
7	CR_DEAD_TIME_FRACTION	Trigger dead time fraction; pure number
8	CR_INPUT_RATE	Estimated true input rate using Poisson statistics; 1/s
9-15	CR_R1 - CR_R7	Reserved

The above structure repeats for a second set of statistics registers (CR_RAW_REAL_TIME2, and so on) for systems that use histogram bank switching.

4.8 Pseudo group CMD_DAQ: Bulk data

On an MSP430-based instrument there is only one bulk data array to hold either histogram, list mode or trace data. The instrument fills the first 12 bytes with an appropriate data header to indicate the type of data (and their characteristics) stored in the bulk data array. As a result only one type of bulk data (histogram, list mode or trace) can be stored at one time.

In Arm-9 processor-based instruments, there is room to store each type of bulk data in a separate chunk of memory, and the respective data acquisitions can occur concurrently.

The Morpho will produce all data in little-endian format. If the instrument does not process the bulk data locally, it may send the data upstreams without converting the byte order, but indicating the endianess in the data header.

4.9 Group CMD_DAQ: Histogram data

The selector DH_TYPE_HISTO indicates histogram data. The format (uint16 or uint32) is described in the data header. The selector DH_TYPE_HISTO1 indicates the second memory bank in systems with memory bank switching,

4.10 Group CMD_DAQ: List mode data

List mode data consist of 16-bit uint16 entries. There are 340 events per list mode buffer. Each event uses three 16-bit entries. Their contents depend on the “List mode data switch”, cf below.

LMdataSwitch = 0		**LMdataSwitch = 1**	
0	16-bit energy	0	16-bit energy
1	32-bit time value, lower value word first	1	16-bit PID value
2		2	16-bit time

List mode data are sent as blocks of 1024 x uint16. The last word is a bit-field with the following information:

15							8	7							0
SW							num_events								

num_events: Number of events in the buffer; 6 bytes/event.

SW: List mode data switch: 0 → {32-bit time, energy}, 0 → {16-bit time, PID, energy}

The selector DH_TYPE_LM in the data header indicates list mode data.

4.11 Group CMD_DAQ: Trace data

Trace (pulse) data are stored as uint16. ADC samples are simply stored consecutively in memory. The selector DH_TYPE_TRACE in the data header indicates trace data.

5 User Input to Controller

The user input to the controller makes use of the same command header structure but allows input data to reside in appropriate data arrays (uint16, uint32, float32) as is useful and required for a graphical user interface. Most command input data are kept in a uint16 array dataUI16.

Currently, only index 5 and 6 of the 8-word command header are used for command addressed at the controller. Set the word at index CH_CMD_GROUP to CMD_API and choose from the list of commands:

Controller commands:

Index	Name	Description
0	CMD_DONE	Ready; no command pending
1	CTRL_AUTO_SETUP	Load settings into the instruments
2	CTRL_SETUP_F	Setup with user-defined float32 parameters
3	CTRL_SETUP_I	Setup with user-defined uint32 parameters
4	CTRL_SETUP_C	Setup with user-defined MCA control register contents
5	CTRL_RAD_SCAN	Send a radiation scan command to the instruments
6	CTRL_TRACE	Acquire a number of traces from instrument
7	CTRL_LISTMODE	Acquire list mode buffers from instrument
8	CTRL_PAR_SCAN	Acquire histograms while varying a parameter, such as integration time (not yet implemented)

5.1 Auto-Setup command

Use the setup command to load the MCA control register on the instrument. Supporting command data are stored in a 16-bit data array cmdDataUI16.

cmdDataUI16 entries:

Index	Name	Description
0	FIRST_ID	First instrument's number; Set 0 if only one instrument
1	LAST_ID	Last instrument's number; Set 0 if only one instrument
2	SETUP_NO	Choose one of 7 predefined setups (1 – 7)
3	CTRL_HV	Bit 0: 0 → Turn HV off; 1 → Turn HV on Bit 1: 0 → Use predefined HV; 1 → Use user-provided HV; Bit 2: 0 → Turn MCA off; 1 → Turn MCA on; (RDR only)

The instrument high voltages should be stored in a float32 dataFP32 array. For Ethernet-connected instruments the IP-addresses should be stored as 32-bit numbers in a uint32 cmdDataUI32 array.

5.2 Setup_C command

Use this command to download a user-populated control register set to a single instrument. See the FPGAmap document of the MCA for the contents of the control registers.

cmdDataUI16 entries:

Index	Name	Description
0	INSTRUMENT_ID	Selected instrument's number; Set 0 if only one instrument
1	CTRL_HV	Bit 0: 0 → Turn HV off; 1 → Turn HV on Bit 1: ignored Bit 2: 0 → Turn MCA off; 1 → Turn MCA on; (RDR only)
2 – 17	CTRL_REGS	Control register contents, 16 words

5.3 Radiation scan command

cmdDataUI16 entries:

Index	Name	Description
0	FIRST_ID	First instrument's number; Set 0 if only one instrument
1	LAST_ID	Last instrument's number; Set 0 if only one instrument
2	MODE	Bit-0: Enable histogram bank switching (histo and stats period must be equal) Bit 1: Enable auto reporting (Instrument initiates communication)

		for statistics Bit 2: Enable auto reporting for histograms Bit-3: Clear statistics after update Bit-4: Clear histogram after update Bit-5: Set to stop DAQ Bit-6: 0 → histogram energies, 1 → histogram pulse heights (diagnostic only);

Load the float32 dataFP32 array as follows:

Index	Name	Description
0	RATES_PERIOD	Period, in seconds, for the instrument to read count rates from the MCA and send them to the controller.
1	HISTO_PERIOD	Period, in seconds, for the instrument to read a histogram from the MCA and send it to the controller.

Setting any of the scan periods to zero will disable the corresponding scan, but will not end the data acquisition.

5.4 Trace acquisition command

cmdDataUI16 entries:

Index	Name	Description
0	FIRST_ID	First instrument's number; Set 0 if only one instrument
1	LAST_ID	Last instrument's number; Set 0 if only one instrument
2	TYPE	Bit 0-1: 0 → untriggered trace, 1 → triggered trace, 2 → validated trace.
3	NUM_BUFFERS	Number of traces requested. Writing a 0 will end an ongoing macro-command for trace acquisition

5.5 List mode acquisition command

cmdDataUI16 entries:

Index	Name	Description

0	FIRST_ID	First instrument's number; Set 0 if only one instrument
1	LAST_ID	Last instrument's number; Set 0 if only one instrument
2	TYPE	Bit 0: type switch
3	NUM_BUFFERS	Number of list mode buffers requested. Writing a 0 will end an ongoing macro-command for list mode acquisition

Bit 0 in the TYPE field selects one of the two list mode acquisition modes: 0 → acquire 32-bit time and 16-bit energy; 1 → acquire 16-bit time, 16-bit PID value, and 16-bit energy.