Bridgeport Instruments, LLC
6448 E HWY 290, STE D-103
www.BridgeportInstruments.com

Instrument Controller Software 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.

The command and data structures in use are described in the Command and Data Interface document.

Details of the communication with the embedded MCA are described in its respective API document: The eMorpho API documentation covers all single-channel eMorpho's. The API for the 4-channel qMorpho is covered in the qMorpho API document.

This document focuses on the code used on the controller, which communicates directly only with the instrument's micro-controller.

2 Architecture

The controller-side software is provided as a library. No main executable is provided. Instead it consists of three sets of functions: hardware I/O to the instruments, formatting and sending command to the instrument, and receiving data from the instrument and parsing the data headers.

2.1 Hardware I/O

The library can be used on a PC or on an ARM-9 processor. when running on a PC, the interface to the instrument will be an Ethernet interface. When running on an ARM-9 that is connected to an instrument with an MSP430 MCU we have a hybrid system. To the instrument the ARM-9 is a controller and it will use an 8-bit handshake data bus to communicate with it. However, to an upstreams PC the ARM-9 will look like an instrument and will need the instrument-side part of the Ethernet code.

2.2 Command generator

The library provides an interface between a user data area with different types of data arrays (uint16, uint32, float32) as they may be required for, for instance, a graphical user interface. On the output side, these user data need to be packed into a byte stream with a header that tells the command, data formats and byte ordering. The associated structures are described in the command and data interface document.

2.3 Data parser

Similar to above, data streams arriving through the serial link need to be examined to discover their origin, content, format and byte ordering so the data can be converted into user-friendly formats for processing and displaying.

3 Hardware I/O

3.1 Ethernet IO

On a PC, the controller needs to first call `WSAStartup(0x0202, &wsa_data)` and do so only once for the duration of the session.

The controller then opens a listening socket for data on port 9932 using `tcp_listen(..)`.

To discover the IP-addresses of any available instruments situated on the same subnet, the controller will open a second, temporary, listen socket on port 9931. It will then issue a broadcast via UDP to all units on the same subnet, and begin to poll the listening socket for the answers coming back from the instruments. Their messages contain their MAC addresses and IP-addresses, which the controller stores. Once all expected messages have arrived, the controller may close the temporary listening socket.

The controller will send commands to port 9933 on the instrument.

3.1.1 Open listening port

This function is used to open a listening port when the controller is running on a PC.

```
uint16 tcp_listen(uint32 myIPaddress, uint16 port)
```

Type	Format	Name	Description
input	uint32	myIPaddress	The IP address of the controller
input	uint16	port	Port number on which it is going to listen

The function reserves a large buffer for the socket (32k to 64k) so that data arriving from the controller can get queued without holding up the data transfer.

Returns the socket number if no errors are encountered.

3.1.2 Scanning for instruments

This is the function used to discover any attached instruments and receive their MAC and IP addresses.

```
int tcp_probe(uint32 myIPaddress, uint32 *Detector_IP_Addresses, uint8
*Detector_IP_Addresses ,  uint61 NumInstrExpected)
```

Type	Format	Name	Description
input	uint32	myIPaddress	The IP address of the controller
output	uint32 *	Detector_IP_Addresses	Address of a data array to hold the detector IP addresses
output	uint8 *	Detector_MAC_Addresses	Address of a data array to hold the detector MAC addresses, 6 bytes per detector
input	uint16	NumInstrExpected	Number of instruments expected to answer the UDP broadcast.

Returns number of instruments that did answer the UDP broadcast.

3.1.3 UDP broadcast

This function is called from tcp_probe and is used to issue the UDP broadcast to which the instruments are supposed to respond by sending their MAC addresses and IP-addresses.

```
udp_brdcst(uint32 myIPaddress, uint16 udp_port)
```

Type	Format	Name	Description
input	uint32	myIPaddress	The IP address of the controller
input	uint16	udp_port	Port number to which it is sending the broadcast message.

The function opens a UDP socket, sends the broadcast message, closes the socket and exits.

Returns 0 if no errors are encountered.

3.1.4 ARM-9 parallel I/O

When the ARM-9 is a controller it may connect to an instrument via an 8-bit wide parallel handshake bus. The associated write function sends commands in the standard format. On a write, the receiving MSP430-based instrument will be forced into an interrupt in order to minimize the response time.

When the MSP430-based instrument sends data back to the ARM-9 it uses the data formats described in the command and data interface document. The ARM-9 processor does not enter an interrupt on receiving the data transfer request but picks up the request when repeating its main loop.

4 Command Formatter

The controller provides a convenient interface between data organization that is practical for a user interface (a GUI, for instance) and the strict formatting requirements necessary for a data connection between different types of computers.

Commands consist of a command header (unsigned short) and command data which may be in unsigned short, unsigned long or floating point format arrays. The command generator uses input from these four data arrays to assemble the commands to be sent to the instrument. The details of the command header and data fields are described in the "Command and data Interface" document. Here we only describe in which data arrays the data are expected.

The input data arrays are:

Type	Name	Description
uint16	cmdHeader	Eight-word command header
uint16	cmdDataUI16	16-bit unsigned integer command data; content depends on command
uint32	cmdDataUI32	32-bit unsigned integer command data; content depends on command
float32	cmdDataFP32	32-bit floating point command data; content depends on command

The complete command is assembled into the output data array:

Type	Name	Description
uint16	cmd2instr	Command, complete with header and following data.

Currently, the command formatter recognizes one command group: `CMD_API`. cf 5.1

4.1 sendComand

Once a command has been assembled into `cmd2instr`, the controller sends the command to the instrument. When the controller is a PC the command is sent via Ethernet and the function takes the following parameters:

```
sendCommand(uint32 ip_addr,uint8 *cmd2instr, int numBytes)
```

Type	Name	Description
uint32	ipAddress	32-bit ip-address in the controller's native byte order.
uint8 *	cmd2instr	Pointer to the start of the command buffer
int	numBytes	Number of bytes to be sent

If the controller is implemented on an ARM-9 processor connected to one or more instruments via a shared 8-bit parallel handshake bus, the first argument of the send_command function will be used as a device address. Where the hardware connectivity supports this, devices can be selected via physical "chip-select" lines (geographical addressing).

5 The COMMAND_API function

This function manages all requests belonging to the CMD_API command group. It accepts data arrays that can conveniently be filled in a user interface, for instance a GUI.

```
int CAPI_command(uint16 *cmdHeader, float32 *cmdDataFP32, uint32 *cmdDataUI32,
uint16 *cmdDataUI16, uint16 *dataUI16, uint32 *dataUI32, float32 *dataFP32,
uint16 *diagnostics)
```

Type	Name	Description
uint16	cmdHeader	Eight-word command header
float32	cmdDataFP32	32-bit floating point command data; content depends on command
uint32	cmdDataUI32	32-bit unsigned integer command data; content depends on command
uint16	cmdDataUI16	16-bit unsigned integer command data; content depends on command
uint16	dataUI16	16-bit unsigned integer data returned by API or instrument
uint32	dataUI32	16-bit unsigned integer data returned by API or instrument
float32	dataFP32	16-bit unsigned integer data returned by API or instrument
uint16	diagnostics	16-bit unsigned integer diagnostics data returned by the API; for debugging purposes.

5.1 Setup commands

Setup commands are used to load the Morpho control registers to steer the MCA's operation. Setup commands may rely on predefined settings, settings parameters provided in SI physical units or as user-provided sets of 16-word control register arrays. Currently, only the AustoSetup command has been implemented.

5.1.1 AutoSetup

Selected when `cmdHeader[CH_CMD_COMMAND] == CTRL_AUTO_SETUP`

Generate a setup command to configure the Morpho control registers with one of of 7 predefined settings. A user defined high-voltage may be included as a floating point constant in cmdDataFP32[0].

The input data are:

Item	Description
cmdDataUI16[0]	nFirst = index of first instrument; set to 0 if only one
cmdDataUI16[1]	nLast = Index of last instrument; set to 0 if only one
cmdDataUI16[2]	Bit-field: Bit-0: Set to 1 to turn HV on; Bit-1: set to 1 when providing user-defined HV
cmdDataUI32[0..n-1]	IP-addresses or other device addresses for n = nLast – nFirst + 1 connected and targeted instruments
cmdDataFP32[0]	User-provided high-voltage value.

5.2 Radiation Scan

This command `CTRL_RAD_SCAN` instructs the instrument to periodically acquire and send count rate and histogram data to the controller. The period for sending count rates can be different from the period for sending histograms.

The arguments for this command are:

Item	Description
cmdDataUI16[0]	nFirst = index of first instrument; set to 0 if only one
cmdDataUI16[1]	nLast = Index of last instrument; set to 0 if only one
cmdDataUI32[0..n-1]	IP-addresses or other device addresses for n = nLast – nFirst + 1 connected and targeted instruments
cmdDataFP32[0]	Reporting period for count rate data (rates_period)
cmdDataFP32[1]	Reporting period for histogram data (histogram_period)
cmdDataFP32[3]	Bit-field governing data acquisition; index = DSCAN_MODE

Content of the DSCAN_MODE bit-field located in cmdDataFP32[DSCAN_MODE]:

15							**8**	**7**							**0**
									HA	STOP	CH	CS	HIAR	STAR	SE

SE: Segment enable; Count rate and histogram reporting periods must be equal (advanced).

STAR: Enable count rate auto reporting. If the count-rate reporting period is > 0 but STAR==0, the instrument will periodically read out the MCA count-rates but not report them to the host. Presumably, the instrument's embedded processor will periodically acquire and analyze the count-rates and either record them to non-volatile media or alert the controller when the rate exceeds a preset threshold.

HIAR: Enable histogram auto reporting. If the histogram reporting period is > 0 but HIAR==0, the instrument will periodically read out the MCA histogram but not report it to the host. Presumably, the instrument's embedded processor will periodically acquire and analyze a histogram and report to the controller only if the histogram no longer fits the previous classification; for instance background radiation or industrial isotope.

CS: Set to force a clear of the statistics registers with every rates_period. This resets the count-rates measurements on every period boundary.

CH: Set to force a clear of the histogram memory with every histogram_period.

STOP: Call an end to all ongoing or pending DAQ by forcing the RunControl register or RunAction register to zero.

HA: Set to histogram pulse amplitudes rather than pulse energies. Use 'histogram pulse amplitudes' to make sure the signals fit into the ADC dynamic range. Use histogram energies (HA=0) for best energy resolution.

5.3 Trace acquisition

The command `CTRL_TRACE` is a bulk-data command. It instructs the instrument to acquire a given number of traces (digitized waveforms) and send them to the controller. Traces can be untriggered traces, simple triggered or validated (according to any validation algorithm that may hev been implemented in the instrument).

The arguments for this command are:

Item	Description
cmdDataUI16[0]	nFirst = index of first instrument; set to 0 if only one
cmdDataUI16[1]	nLast = Index of last instrument; set to 0 if only one
cmdDataUI16[2]	Type of trace to be acquired: 0 → untriggered, 1 → triggered, 2 → validated.
cmdDataUI16[3]	Number of traces to be acquired
cmdDataUI16[4 .. 6]	reserved

5.4 List mode acquisition

The command CTRL_LISTMODE is a bulk-data command. It instructs the instrument to acquire a given number of list mode buffers and send them to the controller.

The arguments for this command are:

Item	Description
cmdDataUI16[0]	nFirst = index of first instrument; set to 0 if only one
cmdDataUI16[1]	nLast = Index of last instrument; set to 0 if only one
cmdDataUI16[2]	List mode buffer format: 0 → 16-bit energy + 32-bit time stamp, 1 → 16-bit energy, 16-bit pulse shape word, and 16-bit time stamp
cmdDataUI16[3]	Number of list mode buffers to be acquired
cmdDataUI16[4 .. 6]	reserved

5.5 Instrument search

The command CTRL_ENET_SCAN instructs the controller to scan for the presence of Ethernet connected instruments on the same subnet. It uses the method described in chapter 3.1.

The arguments for this command are:

Item	Description
cmdDataUI32[0]	The IP address of the controller in the controller's native byte order.
cmdDataUI32[1]	The number of instruments expected to be found.
dataUI32[0]	Number of instruments found.
dataUI32[1 .. n]	IP-addresses of instruments no. 1 through n. (in controller's native byte order)

5.6 Receive data via Ethernet

The command CTRL_ENET_RECV polls the controller's listening socket and receives any data that are available. Received data are sent to the parse_data function.

The arguments for this command are:

Item	Description
dataUI16[0]	16-bit unsigned integer array; filled if data are of type uint16
dataUI32[1]	32-bit unsigned integer array; filled if data are of type uint32
dataFP32[0]	32-bit float array; filled if data are of type float32
diagnostics	The data header is copied into the diagnostics array.

5.7 Receive data via parallel bus

The command CTRL_P8_RECV polls the ARM-9 controller's parallel bus to find out if a send request from the instrument has arrived. If so, the ARM-9 will receive the data and send them to the parse_data function.

The arguments for this command are:

Item	Description
dataUI16[0]	16-bit unsigned integer array; filled if data are of type uint16
dataUI32[1]	32-bit unsigned integer array; filled if data are of type uint32
dataFP32[0]	32-bit float array; filled if data are of type float32
diagnostics	The data header is copied into the uint16 diagnostics array.

The user interface will examine the content of the 12-byte data header in the diagnostics array to discover the length, and nature of the received data. The data byte order is guaranteed to be in the controller's native byte order.

5.8 Parse data

Within the receive commands there are two components: The first is the function that physically receives the byte stream and is hardware deependent. The second is the data parser (hardware independent) that ensures all data are converted into the controller's native byte order. To save time and computer resources at the instrument side, bulk data such as traces, list mode and histograms are kept in the byte order they were generated (typically `MORPHO_BYTE_ORDER = LITTLE_ENDIAN`) and sent in that order. When the data arrive at the controller, however, the data byte order may have to corrected to make sense to the local software.