Application Note

AN2471/D
3/2003

PC Master Software Communication Protocol Specification

By **Pavel Kania and Michal Hanak**
S^{3}L Applications Engineerings
MCSL Roznov pod Radhostem

Introduction

The purpose of this application note is to detail the PC master software communication protocol specification, which is used during debugging with the PC master Windows® application. The information in this document should be sufficient to develop further PC master software host applications.

The PC master software RS-232 protocol is a set of simple binary structures and conventions, enabling a data/code exchange between a personal computer (PC) and a target controller board. It uses raw 8 bits, no parity, and serial transfer at a standard speed (9600 kbps by default).

The communication model is based on a master-slave relationship, where the PC sends a message with a **command** and its arguments, and the target responds immediately (within a specified time) with the operation status code and return data. The target never initiates communication; its responses are specified and always of a fixed (known) length. This is true on a logical level; however, on a link level, there is a replication of special start-of-block bytes. The following paragraphs will detail this replication.

Windows is a registered trademarks of Microsoft Corporation in the U.S. and/or other countries.

PC Master Software Protocol Messages Structure

Command Message (PC → Target)

Because the communication protocol and all its message have been optimized to achieve minimum data flow on the line, the messages are divided into fast commands and standard commands.

Fast Command Message Format

Fast commands do not contain information about length of the data part, as the length is known. Fast commands are all commands with a a command number greater or equal to 0xC0. See **PC Master Software Command Codes** for more details.

start-of-block (1 BYTE)	command (1 BYTE)	data part (known length)	checksum (1 BYTE)

Standard Command Message Format

In the fast command message format, command numbers must be ≥0xc0. Standard commands contain one byte of data length in the message. See **PC Master Software Command Codes** for more details.

start-of-block (1 BYTE)	command (1 BYTE)	data length (1BYTE)	data part (variable length)	checksum (1 BYTE)

start of block (SOB)
Special character defined as ASCII '+' code (0x2B) is defined as start-of-block: MCB_SOB

command
A one byte command code (see **PC Master Software Command Codes**)

data length
Length of the data part of the message.

data part
Variable length data

checksum
Two's complement checksum; computed by taking the two's complement of the sum of all bytes of a message after the SOB.

The following code example calculates the checksum for a message in standard format before it is transmitted.

```
typedef unsigned char BYTE;

struct {
       BYTE cmd;
       BYTE len;
       BYTE data[N];
       BYTE _space_for_checksum;
} message;

// prepare message
// ...

// calculate checksum
BYTE *p = &message;
BYTE sum = *p++;          // cmd field
for(int i=0; i<=message.len; i++)
       sum += *p++;       // add len and data
// store checksum after last valid data byte
*p = 0x100 - sum;

// transmit SOB byte
// ....

// transmit message and checksum
// (replicating each occurrence of SOB byte)
// ....
```

The start-of-block (SOB) character receives special treatment from the link protocol layer. When received, it should reset the receiver's state machine and initialize it for reception of a new message. Since the data being transferred across an RS-232 line is in binary format, a byte with a value equal to SOB may be contained in the message body which could cause an undesirable re-initialization of the receiver. This is why each occurrence of a SOB byte in the length, data, or checksum part of a message is signalled by duplicating this byte. On the other hand, the receiver resets its state machine only when the SOB byte it receives is followed by a non-SOB byte. If the receiver receives two consecutive SOB bytes, it merges them into a single one.

The command is any enumerated value from the header file. If the special (fast) commands carry either no fixed length data or short fixed length data, the data part length is encoded directly into the value.

Fast Command Code

The fast command code has the following format:

1	1	L	L	C	C	C	C

1-field
The fast command values are identified by the two most significant bits (MSB) set to 1 (command values are ≥0xc0).

L-field
This field indicates the length of the data part in 2-byte increments (0, 2, 4, and 6 bytes can be specified).

C-field
This field specifies the fast command code.

Response Message (Target → PC)

A response message is always sent from the target to the PC. The format of response messages follows:

start-of-block (1 BYTE)	status code (1 BYTE)	data part (known length)	checksum (1 BYTE)

start of block
The special character MCB_SOB is defined as ASCII '+' code (0x2B).

status code
The one byte operation status code (see **PC Master Software Status Codes**)

data part
Variable length data, the length depends on the status code value. An error response message (status MSB set) carries no data, a successful response message carries known data to the PC (the data length is predetermined).

checksum
Two's complement checksum computed by taking the two's complement of the sum of all bytes of a message after the SOB.

PC Master Software Command Codes

A list of fast and standard commands which are implemented in the protocol can be found in this section. Some commands were added to spread out the family of commands and to better satisfy customer's needs. For more detail, see **PC Master Software Command Specifications**.

Fast Commands

Command Code[1]	Value	Description
MCB_CMD_GETINFO	0xC0	Retrieve board information structure
MCB_CMD_GETINFOBRIEF — **V2**	0xC8	Retrieve a subset of board information structure
MCB_CMD_READVAR8(EX — **V3**)	0xD0 (0xE0)	Read BYTE variable
MCB_CMD_READVAR16(EX — **V3**)	0xD1 (0xE1)	Read WORD variable
MCB_CMD_READVAR32(EX — **V3**)	0xD2 (0xE2)	Read DWORD variable
MCB_CMD_WRITEVAR8 — **V2**	0xE3	Write BYTE variable
MCB_CMD_WRITEVAR16 — **V2**	0xE4	Write WORD variable
MCB_CMD_WRITEVAR32 — **V2**	0xF0	Write DWORD variable
MCB_CMD_WRITEVAR8MASK — **V2**	0xE5	Write specified bits in BYTE variable
MCB_CMD_WRITEVAR16MASK — **V2**	0xF1	Write specified bits in WORD variable
MCB_CMD_STARTREC	0xC1	Start data recorder
MCB_CMD_STOPREC	0xC2	Stop data recorder
MCB_CMD_GETRECSTS	0xC3	Get the recorder status
MCB_CMD_GETRECBUFF(EX — **V3**)	0xC4 (0xC9)	Get the recorder data
MCB_CMD_READSCOPE	0xC5	Get the scope data
MCB_CMD_GETAPPCMDSTS	0xC6	Get the application command status

1. **V2** — The command was added to the PC master software communication protocol version 2.
   V3 — The command was added to the PC master software communication protocol version 3.

Standard Commands

Command Code[1]	Value	Description
MCB_CMD_READMEM(EX — **V3**)	0x01 (0x04)	Read a block of memory
MCB_CMD_WRITEMEM(EX — **V3**)	0x02 (0x05)	Write a block of memory
MCB_CMD_WRITEMEMMASK(EX — **V3**)	0x03 (0x06)	Write a block of memory with bit mask
MCB_CMD_SETUPSCOPE(EX — **V3**)	0x08 (0x0A)	Setup the oscilloscope
MCB_CMD_SETUPREC(EX — **V3**)	0x09 (0x0B)	Setup the recorder
MCB_CMD_SENDAPPCMD	0x10	Send the application command

1. **V2** — The command was added to the PC master software communication protocol version 2.
   V3 — The command was added to the PC master software communication protocol version 3.

PC Master Software Status Codes

Success Codes

There is a list of the success codes included in the response message as shown in the following table. Success codes are designated by the most significant bit (MSB) set to 0.

Status Code	Value	Description
MCB_STS_OK	0x00	The operation finished successfully, return data is valid (if any)
MCB_STS_RECRUN	0x01	Data recorder is running (see data recorder description)
MCB_STS_RECDONE	0x02	Data recorder is stopped (see data recorder description)

Error codes

Error codes are designated by the MSB set to 1.

Status Code	Value	Description
MCB_STC_INVCMD	0x81	Unknown command code (unsupported operation).
MCB_STC_CMDCSERR	0x82	Command checksum error
MCB_STC_CMDTOOLONG	0x83	Command to long, the receive buffer to small to accept it
MCB_STC_RSPBUFFOVF	0x84	Response would not fit into the transmit buffer
MCB_STC_INVBUFF	0x85	Invalid buffer length or operation
MCB_STC_INVSIZE	0x86	Invalid size specified
MCB_STC_SERVBUSY	0x87	Service is busy
MCB_STC_NOTINIT	0x88	Service is not initialized

PC Master Software Command Specifications

MCB_CMD_GETINFO (0xC0)

This command directs the target to determine the internal board configuration.

If this command is implemented it means that all the new commands of PC master software protocol V1, V2, or V3 (dependent on the *protVer* variable) are implemented unless the `MCB_CFGFLAG_NOFASTREAD`, `MCB_CFGFLAG_NOFASTWRITE`, or the `MCB_CFGFLAG_EXACCESSONLY` flag in the response message is set.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains the structure with board configuration information. Refer to the following code example.

```
#define MCB_DESCR_SIZE 25
#define MCB_CFGFLAG_BIGENDIAN    0x01    // board uses bigEndian
                                         // byte order
#define MCB_CFGFLAG_NOFASTREAD   0x02    // do not try the fast
                                         // read commands
#define MCB_CFGFLAG_NOFASTWRITE  0x04    // do not try the fast
                                         // write commands
#define MCB_CFGFLAG_EXACCESSONLY 0x08    // do not try 16-bit
                                         // address access

typedef struct
{
    BYTE protVer;               // protocol version
    BYTE cfgFlags;              // MCB_CFGFLAG_ bits
    BYTE dataBusWdt;            // data bus width (bytes)
    BYTE globVerMajor;          // board firmware version major number
    BYTE globVerMinor;          // board firmware version minor number
    BYTE cmdBuffSize;           // receive/transmit buffer size
                                //(without SOB, CMD/STS and CS)
    WORD recBuffSize;           // recorder buffer memory
    WORD recTimeBase;           // recorder time base
                                // 2 MSB exponent 1 = m, 2 = µ, 3 = n
    BYTE descr[MCB_DESCR_SIZE]; // ANSI string, board description

} MCB_RESP_GETINFO, FAR* LPMCB_RESP_GETINFO;
```

protVer

Protocol version number, the valid values are 1, 2, or 3. Note that if the value is set as 1, only commands which are not declared as V2 or V3

commands will be available (see **Fast Commands** and **Standard Commands**). When the value is set to 2, only the commands declared as V3 will not be available. If the value is set to 3, all the commands will be available.

cfgFlags
Board configuration flags, which can be any combination of flag bits are described in the following table.

Flag Bits	Value	Description
MCB_CFGFLAG_BIGENDIAN	0x01	The board uses the big-endian number format
MCB_CFGFLAG_NOFASTREAD(1)	0x02	Disable trying to use fast read commands
MCB_CFGFLAG_NOFASTWRITE(1)	0x04	Disable trying to use fast write and masked-write commands (valid only for PC master software protocol V2 or higher)
MCB_CFGFLAG_EXACCESSONLY(1)	0x08	Disable trying to use commands which address the 16-bit address space (valid for PC master software protocol V3)

1. If the *protVer* variable is 1 the flags MCB_CFGFLAG_NOFASTREAD, MCB_CFGFLAG_NOFASTWRITE, and MCB_CFGFLAG_EXACCESSONLY will have no impact.

dataBusWdt
The width of a data word accessible on a single address

globVerMajor, globVerMinor
Board firmware version

cmdBuffSize
The size of the transmit/receive buffer (buffer for receiving commands and transmitting the responses)

recBuffSize
The size of the buffer used for the internal recorder

recTimeBase
The time base for the internal recorder (the recorder interrupt period). The lower 14 bits of the word are the mantissa and two MSB bits specifically encoded exponent (1 = ms, 2 = μs, 3 = ns).

Example values:
0x4001 ... 1 ms
0x400A ... 10 ms
0x8014 ... 20 μs
0xc1F4 ... 500 ns

descr
A zero terminated string with board description

2. **MCB_STC_INVCMD (0x81)**
   The command was not implemented. Beginning with protocol V2, the board need not implement this command and the command **MCB_CMD_GETINFOBRIEF (0xC8)** must be implemented instead.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_GETINFOBRIEF (0xC8)

This command directs the target to determine the internal board configuration and can be implemented by the board application instead of MCB_CMD_GETINFO, to save memory resources.

This command has been specified in PC master software protocol V2. All the new commands of PC master software protocol V2 or V3 (specified with the *protVer* variable) are implemented unless the `MCB_CFGFLAG_NOFASTREAD`, `MCB_CFGFLAG_NOFASTWRITE`, or the `MCB_CFGFLAG_EXACCESSONLY` flag in the response message is set.

Command data:

Command carries no data

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation. The data returned contains the structure along with the subset of board configuration information. This structure is in the same format as the full information structure received from **MCB_CMD_GETINFO (0xC0)**. Refer to the following code example.

```
#define MCB_CFGFLAG_BIGENDIAN     0x01   // board uses bigEndian
                                         // byte order
#define MCB_CFGFLAG_NOFASTREAD    0x02   // do not try the fast
                                         // read commands
#define MCB_CFGFLAG_NOFASTWRITE   0x04   // do not try the fast
                                         // write commands
#define MCB_CFGFLAG_EXACCESSONLY  0x08   // do not try 16-bit
                                         // address access

typedef struct
{
    BYTE protVer;          // protocol version
    BYTE cfgFlags;         // MCB_CFGFLAG_ bits
    BYTE dataBusWdt;       // data bus width (bytes)
    BYTE globVerMajor;     // board firmware version major number
    BYTE globVerMinor;     // board firmware version minor number
    BYTE cmdBuffSize;      // receive/transmit buffer size
                           // (without SOB, CMD/STS and CS)

} MCB_RESP_GETINFOBRIEF, FAR* LPMCB_RESP_GETINFOBRIEF;
```

protVer
Protocol version number; the valid value is 2 or 3 since this command was first specified in protocol V2. When the value is set to 2 the commands declared as V3 will not be available.

cfgFlags
Board configuration flags, this can be any combination of flag bits described in the following table.

Flag Bits	Value	Description
MCB_CFGFLAG_BIGENDIAN	0x01	The board uses the big-endian number format
MCB_CFGFLAG_NOFASTREAD	0x02	Disable trying to use fast read commands
MCB_CFGFLAG_NOFASTWRITE	0x04	Disable trying to use fast write and masked-write commands (valid only for PC master software protocol V2 or higher)
MCB_CFGFLAG_EXACCESSONLY(1)	0x08	Disable trying to use commands which address the 16-bit address space (valid for PC master software protocol V3)

1. If the *protVer* variable is 2, the flag MCB_CFGFLAG_EXACCESSONLY will have no impact.

dataBusWdt
The width of a data word accessible on a single address

globVerMajor, globVerMinor
Board firmware version

cmdBuffSize
The size of the transmit/receive buffer (buffer for receiving commands and transmitting the responses)

2. **MCB_STC_INVCMD (0x81)**
The command was not implemented. In this case, the standard **MCB_CMD_GETINFO (0xC0)** command must be implemented.

3. **MCB_STC_CMDCSERR (0x82)**
A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_READVAR8 (0xD0) and MCB_CMD_READVAR8EX (0xE0)

These commands direct the target to read the BYTE variable.

MCB_CMD_READVAR16 (0xD1) and MCB_CMD_READVAR16EX (0xE1)

These commands direct the target to read the WORD variable.

MCB_CMD_READVAR32 (0xD2) and MCB_CMD_READVAR32EX (0xE2)

These commands direct the target to read the DWORD variable.

The "EX" commands have been specified in PC master software protocol V3 to support devices with a 32-bit address space.

Command data:

The address of the memory location to read. The length of the data part is 2 bytes for the "no-EX" commands (16-bit address) and 4 bytes for "EX" commands (32-bit address).

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains the variable value and according to the command, the data part length is 1, 2, or 4 bytes.
2. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_WRITEVAR8 (0xE3)

This is a fast command that directs the target to write the BYTE variable.

MCB_CMD_WRITEVAR16 (0xE4)

This is a fast command that directs the target to write the WORD variable.

MCB_CMD_WRITEVAR32 (0xF0)

This is a fast command that directs the target to write the DWORD variable.

These commands have been specified in PC master software protocol V2 as fast alternatives to the **MCB_CMD_WRITEMEM (0x02) and MCB_CMD_WRITEMEMEX (0x05)** commands.

NOTE: *The data part of the BYTE write command cannot be optimized to 3 bytes (see **PC Master Software Protocol Messages Structure**). Therefore, it has the same length as a standard 1 BYTE write using the MCB_CMD_WRITEMEM command.*

Command data:

The structure specifying the destination address and the variable value. Refer to the following code example.

```
typedef struct
{
    WORD addr;          // memory location address
    union {
        BYTE val_b;     //  8-bit value for MCB_CMD_WRITEVAR8
        WORD val_w;     // 16-bit value for MCB_CMD_WRITEVAR16
    };

} MCB_DATA_WRITEVAR, FAR* LPMCB_DATA_WRITEVAR;

typedef struct
{
    WORD addr;          // memory location address
    DWORD val;          // 32-bit value

} MCB_DATA_WRITEVAR32, FAR* LPMCB_DATA_WRITEVAR32;
```

addr
The address of the variable's memory location

val_b
The value of the BYTE variable for the **MCB_CMD_WRITEVAR8 (0xE3)** command

val_w
The value of the WORD variable for the **MCB_CMD_WRITEVAR16 (0xE4)** command

val
The value of the DWORD variable for the **MCB_CMD_WRITEVAR32 (0xF0)** command

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STC_INVCMD (0x81)**
   The command is not implemented.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_WRITEVAR8MASK (0xE5)

This is a fast command directing the target to write the BYTE variable with AND mask.

MCB_CMD_WRITEVAR16MASK (0xF1)

This is a fast command that directs the target to write the WORD variable with AND mask.

These commands have been specified in PC master software protocol V2 as the fast alternatives to the **MCB_CMD_WRITEMEMMASK (0x03) and MCB_CMD_WRITEMEMMASKEX (0x06)** commands.

Command data:

This command must include the structure specifying the destination address, the variable value and masking value. Refer to the following code example.

```
typedef struct
{
    WORD addr;          // memory location address
    BYTE val;           // variable value
    BYTE mask;          // mask value

} MCB_DATA_WRITEVAR8MASK, FAR* LPMCB_DATA_WRITEVAR8MASK;

typedef struct
{
    WORD addr;          // memory location address
    WORD val;           // variable value
    WORD mask;          // mask value

} MCB_DATA_WRITEVAR16MASK, FAR* LPMCB_DATA_WRITEVAR16MASK;
```

addr
The address of the variable's memory location

val
The variable value

mask
The mask value

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STC_INVCMD (0x81)**
   The command is not implemented.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_STARTREC (0xC1)

This command directs the target to start the recorder. When the recorder is running, it can be stopped manually by using the **MCB_CMD_STOPREC (0xC2)** command, or is stopped automatically when the trigger condition is satisfied. When using the recorder, the **MCB_CMD_GETINFO (0xC0)** command must be implemented to initiate the communication, since it contains basic information about the recorder.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_OK (0x00)**
   The recorder was started, the response carries no data.
2. **MCB_STS_RECRUN (0x01)**
   The recorder was already running, the response carries no data.
3. **MCB_STC_NOTINIT (0x88)**
   An error has occurred, the recorder was not initialized with the **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands.
4. **MCB_STC_INVCMD (0x81)**
   The recorder service is not implemented.
5. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_STOPREC (0xC2)

This command directs the target to stop the recorder. When using the recorder, the **MCB_CMD_GETINFO (0xC0)** command must be implemented to initiate the communication, since it contains basic information about the recorder.

Command data:

This command carries no data.

Possible responses:

6. **MCB_STS_OK (0x00)**
   A successful operation, the recorder was stopped. The response carries no data.
7. **MCB_STS_RECDONE (0x02)**
   The recorder was already stopped, the response carries no data.
8. **MCB_STC_NOTINIT (0x88)**
   An error has occurred, the recorder was not initialized with the **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands.

9. **MCB_STC_INVCMD (0x81)**
   The recorder service is not implemented.
10. **MCB_STC_CMDCSERR (0x82)**
    A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_GETRECSTS (0xC3)

This command directs the target to retrieve the recorder status. When using the recorder, the **MCB_CMD_GETINFO (0xC0)** command must be implemented to initiate the communication, since it contains basic information about the recorder.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_RECRUN (0x01)**
   The recorder is running, the response carries no data.
2. **MCB_STS_RECDONE (0x02)**
   The recorder is stopped, the response carries no data.
3. **MCB_STC_NOTINIT (0x88)**
   The recorder was not initialized with **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands.
4. **MCB_STC_INVCMD (0x81)**
   The recorder service is not implemented.
5. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_GETRECBUFF (0xC4) and MCB_CMD_GETRECBUFFEX (0xC9)

These commands direct the target to retrieve the recorder data buffer description, and can be used when the recorder is stopped to get information on how to read the recorded data values. When using the recorder, the **MCB_CMD_GETINFO (0xC0)** command must be implemented to initiate the communication, since it contains basic information about the recorder.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains the structure of the recorder buffer information. Refer to the following code example.

```
// data part of the response to the MCB_CMD_GETRECBUFF command
typedef struct
{
    WORD buffAddr;          // cyclic buffer address
    WORD startIx;           // first sample index

} MCB_RESP_GETRECBUFF, FAR* LPMCB_RESP_GETRECBUFF;

// data part of the response to the MCB_CMD_GETRECBUFFEX command
typedef struct
{
                DWORD buffAddr;     // cyclic buffer address
                WORD       startIx; // first sample index
} MCB_RESP_GETRECBUFFEX, FAR* LPMCB_RESP_GETRECBUFFEX;
```

   buffAddr
   The physical address of the recorder buffer. The buffer is considered to be circular, its length is set by the **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands.

   startIx
   The index of first variable set in the circular buffer. See **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** for a description of the recorder data sets.

2. **MCB_STC_SERVBUSY (0x87)**
   The recorder is running, it must be in a stopped state before this operation begins. See **MCB_CMD_STOPREC (0xC2)** and **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** for the information about stopping the recorder automatically and manually.

3. **MCB_STS_NOTINIT (0x88)**
   The recorder was not initialized with the **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands.

4. **MCB_STC_INVCMD (0x81)**
   The recorder service is not implemented.

5. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_READSCOPE (0xC5)

This command directs the target to read the oscilloscope variables.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains the values of all valid oscilloscope variables. The total data length is a sum of "*varDef[i].size*" members of **MCB_CMD_SETUPSCOPE (0x08) and MCB_CMD_SETUPSCOPEEX (0x0A)** commands used for initialization.
2. **MCB_STS_NOTINIT (0x88)**
   The oscilloscope was not initialized with the **MCB_CMD_SETUPSCOPE (0x08) and MCB_CMD_SETUPSCOPEEX (0x0A)** commands.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_GETAPPCMDSTS (0xC6)

This command directs the target to retrieve the application command status. Application commands are described under the **MCB_CMD_SENDAPPCMD (0x10)** topic.

Command data:

This command carries no data.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains a single byte with the application command status information. Refer to the following code example.

```
#define MCB_APPCMDRESULT_NOCMD        0xff
#define MCB_APPCMDRESULT_RUNNING      0xfe

typedef struct
{
    BYTE result;                // command return value

} MCB_RESP_GETAPPCMDSTS;
```

result
The application command status

2. **MCB_STC_INVCMD (0x81)**
   Application commands not implemented
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_READMEM (0x01) and MCB_CMD_READMEMEX (0x04)

These commands direct the target to read the block of memory.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

This command must contain the structure specifying the address and size of the target memory block. The caller must be aware that the size of the target transmit buffer is limited and cannot accommodate large data blocks. The target data bus width must also be considered when splitting the memory block request into multiple **MCB_CMD_READMEM (0x01) and MCB_CMD_READMEMEX (0x04)** commands. When the "size" bytes are read from the address *A0*, the next read should access the address *A1 = A0 + size*/*dataBusWdt*. The transmit buffer size and data bus width can be determined using the **MCB_CMD_GETINFO (0xC0)** command or the **MCB_CMD_GETINFOBRIEF (0xC8)** command. Refer to the following code example.

```
// for the MCB_CMD_READMEM command typedef struct
 typedef struct
{
                BYTE size;          // required size
                WORD addr;          // memory block address

} MCB_DATA_READMEM, FAR* LPMCB_DATA_READMEM;

// for the MCB_CMD_READMEMEX command
typedef struct
{
                BYTE size;          // required size
                DWORD addr;         // memory block address

} MCB_DATA_READMEMEX, FAR* LPMCB_DATA_READMEMEX;
```

size
The size of required memory block

addr
The address of required memory block

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, the data returned contains the block of requested memory.
2. **MCB_STS_RSPBUFFOVF (0x84)**
   An overflow has occurred in the response transmit buffer because the required memory block is too long.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_WRITEMEM (0x02) and MCB_CMD_WRITEMEMEX (0x05)

These commands direct the target to write the block of memory.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

The structure specifying the address and size of the memory block, followed by the block of data bytes. The caller must be aware that the size of the target's receive buffer is limited and cannot write large data blocks. The target data bus width must also be considered, when splitting a large memory block, write into multiple **MCB_CMD_WRITEMEM (0x02) and MCB_CMD_WRITEMEMEX (0x05)** commands. When the "size" bytes are written to address *A0*, the next write should access the address *A1 = A0 + size/dataBusWdt*. The size of the receive buffer and data bus width can be determined using either the **MCB_CMD_GETINFO (0xC0)** command or the **MCB_CMD_GETINFOBRIEF (0xC8)** command. Refer to the following code example.

```
// for the MCB_CMD_WRITEMEM command
typedef struct
{
    BYTE size;          // write buffer size
    WORD addr;          // dest memory address
    BYTE buffer[1];     // data block follows
} MCB_DATA_WRITEMEM, FAR* LPMCB_DATA_WRITEMEM;

// for the MCB_CMD_WRITEMEMEX command
typedef struct
{
    BYTE size;          // write buffer size
    DWORD addr;         // dest memory address
    BYTE buffer[1];     // data block follows

} MCB_DATA_WRITEMEMEX, FAR* LPMCB_DATA_WRITEMEMEX;
```

size
The size of memory block being written

addr
The address where to store the memory block

buffer
The array of "size" data bytes

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STS_CMDBUFFOVF (0x83)**
   An overflow occurred in the command receive buffer because the command message is too long.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred; the command's checksum value was invalid.

MCB_CMD_WRITEMEMMASK (0x03) and MCB_CMD_WRITEMEMMASKEX (0x06)

These commands direct the target to write the block of memory with AND mask. Only the data bits which correspond to those set in the mask are written to target memory.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

This command must carry the structure specifying the address and size of the memory block, followed by the block of data bytes and the block of mask bytes. The caller must be aware that the size of the target's receive buffer is limited and cannot write large data blocks. The target data bus width must also be considered, when splitting a large memory block, write into multiple **MCB_CMD_WRITEMEMMASK (0x03) and MCB_CMD_WRITEMEMMASKEX (0x06)** commands. When the "size" bytes are written to the address *A0*, the next write should access the address *A1 = A0 + size/dataBusWdt*. The size of the receive buffer and data bus width can be determined using the **MCB_CMD_GETINFO (0xC0)** command or the **MCB_CMD_GETINFOBRIEF (0xC8)** command. Refer to the following code example.

```
// for the MCB_CMD_WRITEMEMMASK command
typedef struct
{
    BYTE size;          // write buffer size
    WORD addr;          // dest memory address
    BYTE buffer[1];     // data block follows

} MCB_DATA_WRITEMEM, FAR* LPMCB_DATA_WRITEMEM;
```

```
// for the MCB_CMD_WRITEMEMMASKEX command
typedef struct
{
    BYTE size;          // write buffer size
    DWORD addr;         // dest memory address
    BYTE buffer[1];     // data block follows

} MCB_DATA_WRITEMEMEX, FAR* LPMCB_DATA_WRITEMEMEX;
```

size
The size of memory block being written

addr
The address where the memory block is to be stored

buffer
The array of "size" data bytes, followed by the array of "size" mask bytes

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STS_CMDBUFFOVF (0x83)**
   An overflow occurred in the command receive buffer overflow because the command message is too long.
3. **MCB_STC_INVCMD (0x81)**
   This response indicates that masked write is not supported.
4. **MCB_STC_CMDCSERR (0x82)**
   This response indicates that a communication error has occurred, the command's checksum value was invalid.

MCB_CMD_SETUPSCOPE (0x08) and MCB_CMD_SETUPSCOPEEX (0x0A)

These commands direct the target to set up the oscilloscope variables. For the fastest possible read access to the selected memory locations, up to eight locations can be selected as the "oscilloscope variables" and their values can be obtained immediately using the special fast command **MCB_CMD_READSCOPE (0xC5)**.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

This command must include the number of scope variables followed by the appropriate array of definition structures. Refer to the following code example.

```
// variable-definition structure
typedef struct {
                BYTE size;      // variable size
                WORD addr;      // variable address
} MCB_DATA_SETUPSCOPE_VARDEF;

// data passed to the MCB_CMD_SETUPSCOPE command
typedef struct
{
                BYTE varCnt;    // number of valid var definitions
                MCB_DATA_SETUPSCOPE_VARDEF varDef[1];

} MCB_DATA_SETUPSCOPE, FAR* LPMCB_DATA_SETUPSCOPE;

// variable-definition structure
typedef struct {
                BYTE size;      // variable size
                DWORD addr;     // variable address
} MCB_DATA_SETUPSCOPEEX_VARDEF;

// data passed to the MCB_CMD_SETUPSCOPEEX command
typedef struct
{
                BYTE varCnt;    // number of valid var definitions
                MCB_DATA_SETUPSCOPEEX_VARDEF varDef[1];

} MCB_DATA_SETUPSCOPEEX, FAR* LPMCB_DATA_SETUPSCOPEEX;
```

varCnt
The number of variable definition structures which follow

size
The size of the memory location, which must be a multiple of the target data bus width

addr
The address of the memory location

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STC_INVCMD (0x81)**
   Oscilloscope variables are not supported.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)

These commands direct the target to set up the data recorder and start it. When data access speed requirements exceed the serial line's capability, even using the oscilloscope feature, the variable values must be recorded into internal controller memory. The **MCB_CMD_SETUPREC (0x09) and MCB_CMD_SETUPRECEX (0x0B)** commands change the internal recorder settings. When using the recorder, the **MCB_CMD_GETINFO (0xC0)** command must be implemented to initiate communication, since it contains basic information about the recorder.

When running, the recorder periodically copies the selected memory locations into the internal circular buffer. In a single action, it transfers one set of defined memory locations (variables). When it reaches the end of the buffer, the recorder wraps the pointer back to its beginning. When the trigger occurs (see below), the postTrigger variable sets are recorded and the recorder is stopped. If the value of postTrigger is less than the total buffer length (counting the variable sets), sets that existed before the trigger condition occurred remain in the buffer.

The "EX" modification of the command has been specified in PC master software protocol V3 to support devices with 32-bit address space.

Command data:

This command must carry the recorder setup structure and the recorder variable definitions in the same form as the **MCB_CMD_SETUPSCOPE (0x08) and MCB_CMD_SETUPSCOPEEX (0x0A)** commands. Refer to the following code example.

```
// data passed to the MCB_CMD_SETUPREC command
typedef struct
{
       BYTE trgMode;              // trigger mode
                                  // (0 = disabled, 1 = _/ , 2 = \_)
       WORD totalSmps;            // buffer size in samples
       WORD postTrigger;          // number of samples after
                                  // trigger to save
       WORD timeDiv;              // time base unit multiplier
                                  // (0 = fastest)

       WORD trgVarAddr;           // trigger variable address
       BYTE trgVarSize;           // trigger variable/threshold
                                  // size (1,2,4)
       BYTE trgVarSigned;         // trigger compare mode
                                  // (0 = unsigned, 1 = signed)

    union {
        BYTE b;                   // trgVarSize == 1
        WORD w;                   // trgVarSize == 2
        DWORD dw;                 // trgVarSize == 4
    } trgThreshold;               // trigger comparing threshold

    MCB_DATA_SETUPSCOPE vars;     // recorded variables
```

```
} MCB_DATA_SETUPREC, FAR* LPMCB_DATA_SETUPREC;

// data passed to the MCB_CMD_SETUPRECEX command
typedef struct
{
        BYTE trgMode;             // trigger mode
                                  // (0 = disabled, 1 = _/ , 2 = \_)
        WORD totalSmps;           // buffer size in samples
        WORD postTrigger;         // number of samples after
                                  // trigger to save
        WORD timeDiv;             // time base unit multiplier
                                  // (0 = fastest)

        DWORD trgVarAddr;         // trigger variable address
        BYTE trgVarSize;          // trigger variable/threshold
                                  // size (1,2,4)
        BYTE trgVarSigned;        // trigger compare mode
                                  // (0 = unsigned, 1 = signed)

    union {
        BYTE b;                   // trgVarSize == 1
        WORD w;                   // trgVarSize == 2
        DWORD dw;                 // trgVarSize == 4
    } trgThreshold;               // trigger comparing threshold

    MCB_DATA_SETUPSCOPEEX vars;   // recorded variables

} MCB_DATA_SETUPRECEX, FAR* LPMCB_DATA_SETUPRECEX;
```

trgMode
Trigger mode, when 0, the trigger feature is disabled and the recorder must be stopped manually using **MCB_CMD_STOPREC (0xC2)** command. When this value is non-zero, it determines the threshold crossing slope for the trigger condition.

totalSmps
The recorder buffer length (in variable sets count)

postTrigger
The number of sets, which are recorded after the trigger occurs

timeDiv
This variable specifies the frequency at which samples are recorded:

0 = record samples every call of the recorder function

1 = record samples every second call of the recorder function

2 = record samples every third call of the recorder function

etc...

trgVarAddr
The address of trigger variable

trgVarSize
The size of trigger variable and threshold, this can be 1, 2, or 4 bytes

trgVarSigned
The trigger compare mode, when non-zero the trigger comparator treats the trigger variable and the threshold value as signed values.

trgThreshold
The threshold value, the *trgVarSize* determines what bytes of this field are valid.

vars
The recorder variable definitions, in exactly the same format as the **MCB_CMD_SETUPSCOPE (0x08) and MCB_CMD_SETUPSCOPEEX (0x0A)** commands.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation, no data is returned.
2. **MCB_STC_INVCMD (0x81)**
   The recorder service is not implemented.
3. **MCB_STC_INVBUFF (0x85)**
   The recorder buffer could not be initialized because the *totalSmps* argument is too large.
4. **MCB_STC_INVSIZE (0x86)**
   The recorder variable sizes must be a multiple of data bus width. See **MCB_CMD_GETINFO (0xC0)** command.
5. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

MCB_CMD_SENDAPPCMD (0x10)

This command directs the target to send the application command, which serves as the transport protocol to the target application. Command data is delivered to the target application and the application is notified.

Command data:

This command must include specific data known to the board application.

Possible responses:

1. **MCB_STS_OK (0x00)**
   A successful operation; the application command was sent to the board application and the status of the call can be determined using the **MCB_CMD_GETAPPCMDSTS (0xC6)** command.
2. **MCB_STC_INVCMD (0x81)**
   Application commands are not implemented.
3. **MCB_STC_CMDCSERR (0x82)**
   A communication error has occurred, the command's checksum value was invalid.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2471/D
Rev. 0
3/2003