IMPLEMENTING THE PROFIBUS-DP MESSAGE MODE

This document explains how to communicate with the RMC using the PROFIBUS-DP in message mode. The following explanation is copied from the RMCWIN.HLP. Below that is pseudo code that demonstrates how to implement PROFIBUS-DP Message Mode communications to the RMC efficiently.

Message Mode Overview

Message Mode is one of two modes that can be used with the RMC PROFIBUS-DP module. The other mode is called Compact Mode, which is described in *Using the PROFIBUS-DP Compact Mode* RMCWin online help topic. Be sure to read both topics and consider each carefully before choosing the mode you will use.

Message Mode is so called because it imposes a messaging type interface on PROFIBUS-DP. That is, PROFIBUS-DP only supports cyclic data transfer between a master and a slave. However, the RMC has nearly 64K registers, all of which cannot possibly be sent over PROFIBUS-DP every PROFIBUS-DP scan. Therefore, blocks of these registers can be written to and read from the RMC by using a messaging mechanism: the request in placed in one block of registers, and a response is received in the other block of registers.

In addition to the Command and Response blocks of data, there is also a third block of data—the Status Block—that includes status information that is constantly available. A summary of the Message Mode register blocks follows:

Status Block

This is a block of 32 contiguous input registers that are constantly updated by the RMC. The values will be updated every 2ms when no commands are in progress, and every 6ms when a command is in progress. This block is otherwise independent from the Command and Response blocks. The contents of these 32 registers are configurable using the Status Map editor. See Using the Status Map Editor for details.

Command Block

This is a block of 64 contiguous output registers. These registers are sent from the PLC or PC to the RMC constantly via PROFIBUS-DP, however, the last register in this block contains two flag bits that are toggled to indicate a request. The command block is described in further detail below.

NOTE: Users of Compact Mode should be aware that commands issued over Message Mode are always handled. Specifically this means that is you send a command multiple times it will be processed every time it is received. Under Compact Mode we ignored duplicate commands. This meant that Compact Mode users who wanted to re-issue a command (for example, an ‘E’ command to start an event sequence) needed to toggle the case of the command (for example, toggle between ‘E’ and ‘e’).

Response Block

This is a block of 64 contiguous input registers. These registers are sent from the RMC to the PLC or PC constantly via PROFIBUS-DP, however, the values will not change unless a new request was triggered using the flag bits in the last register of the command block. This mechanism is described below.

Therefore, Message Mode requires 96 input words and 64 output words on the PROFIBUS master. Not all PROFIBUS-DP masters support this; if your master does not support this many registers, you will need to use Compact Mode.

Messaging in Greater Detail:

The Command Block has the following structure:

Register	Description
0-58	Write Data
59	Write address (0-65,535). See the RMC Register Map (PROFIBUS-DP Message Mode) topic for a description of all RMC registers and their addresses.
60	Write length (in words; 0-59)
61	Read address (0-65,535). See the RMC Register Map (PROFIBUS-DP Message Mode) topic for a description of all RMC registers and their addresses.
62	Read length (in words; 0-63)
63	Output Synchronization Register: Bit 15 - Read Request Bit 14 - Write Request

The Response Block has the following structure:

Register	Description
0-62	Read Data
63	Input Synchronization Register Bit 15 - Read Acknowledge Bit 14 - Write Acknowledge

To request a read from the RMC, use the following steps:

1. Wait until the Read Request bit is equal to the Read Acknowledge bit. If they are not equal, the RMC is currently processing a read request.

2. Set the Read Address and Read Length output registers. See RMC Register Map (PROFIBUS-DP Message Mode) topic for a description of all RMC registers and their addresses.

3. Toggle the Read Request bit.

4. Wait until the Read Request bit is equal to the Read Acknowledge bit. When they are equal, the RMC will have updated the Read Data area with the requested data.

5. Use the data in the Read Data area of the Response Block input registers. Make sure that you do not change the Read Request bit until you are done with the data in the Read Data area.

To request a write to the RMC, use the following steps:

1. Wait until the Write Request bit is equal to the Write Acknowledge bit. If they are not equal, the RMC is currently processing a write request.

2. Copy the values you wish to write to the RMC into the Write Data area of the Command Block output registers.

3. Set the Write Address and Write Length output registers. See RMC Register Map (PROFIBUS-DP Message Mode) topic for a description of all RMC registers and their addresses.

4. Toggle the Write Request bit.

5. Wait until the Write Request bit is equal to the Write Acknowledge bit. When they are equal, the RMC has received the data written to it.

The RMC processes reads and writes separately. That is, it is not necessary for a write to complete before starting a read, and in fact, you can simultaneously request both a read and a write; the write to the RMC will occur first, and the read from the RMC will occur after the write has completed. When both are done, both acknowledge bits will be toggled simultaneously.

To further clarify the ordering, keep these basic rules in mind in Message Mode:

1. Do change all write data, the write address, and the write length before toggling the Write Request bit.

2. Do change the read address and read length before toggling the Read Request bit.

3. Do not change the Read Request bit after a read request until you have processed the data in the Read Data area.

4. Do not change the write data, the write address, or the write length when the Write Request bit in the output synchronization register does not match the Write Acknowledge bit in the input synchronization register.

5. Do not change the read address or the read length when the Read Request bit in the output synchronization register does not match the Read Acknowledge bit in the input synchronization register.

The following pseudo code demonstrates a possible logical solution to reading and writing registers in the RMC. The intended sequence is to read the RMC registers, do processing in the PLC, and then write the results back to the RMC. The following pseudo code follows this methodology.

Refer to the RMCWin Online Help for more details on the PROFIBUS-DP interface implementation.

```
#define     NAXES 2           // Number of axes defined as 2.
                              // This can be as many as 8 per RMC.

// GLOBALS


typedef short int             INT16;      // 16 BIT SIGNED INTEGER
typedef unsigned short int    UNS16;      // 16 BIT UNSIGNED INTEGER

Struct axis_t
{
      UNS16 com_pos;          // COMMAND POSITION
      UNS16 tar_pos;          // TARGET POSITION
      UNS16 act_pos;          // ACTUAL POSITION
      UNS16 counts_;          // ENCODER OR TRANSDUCER COUNTS
      UNS16 status_;          // STATUS BITS
      INT16 drive_;           // CONTROL OUTPUT +/- 10000 mV
      UNS16 act_spd;          // ACTUAL SPEED
      INT16 null_drive;       // CONTROL OUTPUT OFFSET OR BIAS
      UNS16 step;             // CURRENT STEP NUMBER
      UNS16 link_value;       // CURRENT LINK VALUE
      UNS16 mode_;            // MODE BITS
      UNS16 accel_;           // ACCELERATION
      UNS16 decel_;           // DECELERATION
      UNS16 speed_;           // REQUESTED SPEED
      UNS16 req_pos;          // REQUESTED POSITION
      UNS16 command_;         // COMMAND
      UNS16 config_;          // CONFIGURATION BITS
      INT16 scale_;           // SCALE COUNTS TO POSITION UNITS
      INT16 offset_;          // OFFSET POSITION UNITS
      UNS16 ext_limit;        // EXTEND LIMIT IN POSITION UNITS
      UNS16 ret_limit;        // RETRACT LIMIT IN POSITION UNITS
      UNS16 pro_gain;         // PROPORTIONAL GAIN
      UNS16 int_gain;         // INTEGRAL GAIN
      UNS16 dif_gain;         // DIFFERENTIAL GAIN
      UNS16 ext_vff;          // EXTEND VELOCITY FEED FORWARD
      UNS16 ret_vff;          // RETRACT VELOCITY FEED FORWARD
      UNS16 ext_aff;          // EXTEND ACCELERATION FEED FORWARD
      UNS16 ret_aff;          // RETRACT ACCELERATION FEED FORWARD
      UNS16 dead_band;        // CONTROL OUTPUT DEAD BAND ELIMINATOR
      UNS16 in_position;      // IN POSITION BAND
      UNS16 following_error;  // FOLLOWING ERROR BAND
      UNS16 auto_stop;        // AUTO STOP BITS
} AXIS;

AXIS RMC[NAXES];              // Array of axis structures, can be 2 to 8 axes


int RMCStatus[32];            // Status information from RMC

#define     RMC_ACK     RMCMsgIn[63]
```

```
int RMCMsgIn[64];             // Message registers from RMC

#define     RMC_WRITE_ADDRESS RMCMsgOut[59]
#define     RMC_WRITE_LENGTH  RMCMsgOut[60]
#define     RMC_READ_ADDRESS  RMCMsgOut[61]
#define     RMC_READ_LENGTH   RMCMsgOut[62]
#define     RMC_REQ           RMCMsgOut[63]

int RMCMsgOut[64];            // Message registers from PLC to RMC
bool RMCReadActive;
int PLCReadAdr;
int RMCReadAdr;
int RMCReadLen;
int ReadThisTime;
int PLCWriteAdr;
int RMCWriteAdr;
int RMCWriteLen;
int WriteThisTime;

#define     MAXWORDSTOREAD    63
#define     MAXWORDSTOWRITE   59

#define     RMCREADREADY      ((RMCMsgIn[63]&0x8000)==(RMCMsgOut[63]&0x8000))
#define     RMCWRITEREADY     ((RMCMsgIn[63]&0x4000)==(RMCMsgOut[63]&0x4000))


// SUBROUTINES TO START READS AND WRITES

void RMCRead(PLCAdr,RMCAdr,WordsToRead)
{
      while ( RMCReadLen )    // WAIT IF ANOTHER READ IS IN PROGRESS
      {
      }                       // After RMCReadLen decrements to Zero, then
                              // start new read
      PLCReadAdr = PLCAdr;
      RMCReadAdr = RMCAdr;
      RMCReadLen = WordsToRead
}

void RMCWrite(RMCAdr,PLCAdr,WordsToWrite)
{
      while ( RMCWriteLen )   // WAIT IF ANOTHER WRITE IS IN PROGRESS
      {
      }                       // After RMCReadLen decrements to Zero, then
                              // start new read
      RMCWriteAdr = RMCAdr;
      PLCWriteAdr = PLCAdr;
      RMCWriteLen = WordsToWrite;
}
```

```
// EXECUTE AT BEGINNING OF SCAN

      if ( RMCReadActive && RMCREADREADY )
                                              // Copy Data if a NEW read has
                                              // occurred and data is ready
      {
            memcpy(PLCReadAdr,RMCMsgIn,ReadThisTime);
                  // Copy PROFIBUS input from RMC to PLC memory
            if ( RMCReadLen )                 // If more data remains to be
            {                                 // read then increment addresses
                  PLCReadAdr += MAXWORDSTOREAD;
                  RMCReadAdr += MAXWORDSTOREAD;
            }
            RMCReadActive = FALSE;            // Clear read here, it
                                              // gets set later
      }

// PLACE RMC CONTROL CODE HERE

// The RMCRead and RMCWrite subroutines will be called from PLC/Control code in
// this section


// EXECUTE AT END OF SCAN

      if ( RMCReadLen && RMCREADREADY )
      {                                       // Start a new read or
                                              // continue existing read
            if ( RMCReadLen > MAXWORDSTOREAD )
            {
                  ReadThisTime = MAXWORDSTOREAD;
                  RMCReadLen -= MAXWORDSTOREAD;
            }
            else
            {
                  ReadThisTime = RMCReadLen;
                  RMCReadLen = 0;
            }
            RMC_READ_ADDRESS = RMCReadAdr;
            RMC_READ_LENGTH = ReadThisTime;
            RMC_REQ ^= RMC_READ_BIT;          // Toggle read request bit
            RMCReadActive = TRUE;             // Set the RMCReadActive flag
                                              // to indicate a read is started
      }
```

```
if ( RMCWRITEREADY )
      {
            for ( i = 0 ; i < NAXES ; i++ )     // Check for pending commands
            {
                  if ( RMC[i].command )
                        Break;
            }
            if ( i < NAXES )
            {                                   // We have a command, so let's
                                                // process it
                                                // Copy the command from the PLC
                                                // memory to the PROFIBUS output
                                                // registers

                  memcpy(RMCMsgOut,RMC[0].mode,12);
                  RMC_WRITE_ADDRESS = 80;       // 0x80 is axis 0 Mode
                  RMC_WRITE_LENGTH = NAXIS * 6; // Each command is 6 words
                  RMC_REQ ^= RMC_WRITE_BIT;     // Toggle write request bit
                  for ( i = 0 ; i < NAXES ; i++ )
                  {
                        RMC[i].command = 0;     // Clear the command pending bit
                  }
            }
            else
            {                                   // No Commands pending
                  if ( RMCWriteLen )            // Check for writes in-process
                  {
                        if ( RMCWriteLen > MAXWORDSTOWRITE )
                        {
                                                // Limit writes to the RMC
                                                // Message mode limit of 59
                              WriteThisTime = MAXWORDSTOWRITE;
                                                // This will split the write
                                                // into multiple messages
                              RMCWriteLen -= MAXWORDSTOWRITE;
                        }
                        else
                        {
                              WriteThisTime = RMCWriteLen;
                              RMCWriteLen = 0;
                        }
                                                // Copy the data in PLC memory
                                                // to Profibus output
                        memcpy(RMCMsgOut,PLCWriteAdr,WriteThisTime);
                        RMC_WRITE_ADDRESS = RMCWriteAdr; // RMCMsgOut[59]
                        RMC_WRITE_LENGTH = WriteThisTime;// RMCMsgOut[60]
                        RMC_REQ ^= RMC_WRITE_BIT;       // RMCMsgOut[63]
                                                        // Toggle write request
                        if ( RMCWriteLen )
                        {
                              PLCWriteAdr += MAXSWORDSTOWRITE;
                              RMCWriteAdr += MAXSWORDSTOWRITE;
                        }
                  }
            }
      }
```