AVG Automation

AC AUTOTECH CONTROLS

UTICOR

M1450 Serial Cam Module

(ASY-M1250-SER2, ASY-M1250-SER2L, ASY-M1250-SER4)

Instruction & Operation Manual

Sales and Marketing ▼
343 St. Paul Blvd.
Carol Stream, IL 60188
Tel: (630)668-3900
FAX: (630)668-4676

Factory Customer Service/Order Entry ▼
4140 Utica Ridge Rd.
Bettendorf, IA 52722
Tel: (319)359-7501
(800)711-5109
FAX: (319)359-9094

Application Hotline
1 (800) TEC-ENGR (832-3647)

Vist our web site at: www.avg.net

MAN-M1450-CAM
REV 01 12/07/99

Table of Contents

1. Introduction

The M1250 Serial Cam Module gives the user remote control of the M1450 PLS. The M1450-300 is currently the only Mini-PLS model which supports this serial interface option.

An M1450 PLS with a Serial Cam Module installed can be connected to an Allen-Bradley PC (via a 1771-KG module) or to an ordinary computer capable of simulating the Allen-Bradley protocol. The user can get status, position, tach and channel setpoint information from the M1250. In addition, the user can program new channel setpoints, new tach motion limits and new machine offsets while in motion.

The M1250 Serial Cam module is designed to work with the Allen-Bradley PLC-2 Family/RS-232C Interface Module (Catalog 1771-KG). The unit may also be connected to any other controller or computer that can simulate the supported protocol using an RS-232 or an RS-422 interface. The M1250 can operate in a stand-alone, point-to-point or multi-drop (RS-422 only) link with a PC or PLC. By interfacing the M1450 to an Allen-Bradley 1771-KF module, it can communicate over the Data Highway Network. See figure 1 for example configurations.

The M1250 serial cam module supports peer-to-peer communications through a full-duplex unpolled protocol. The implemented serial commands are a subset of the Allen-Bradley command set.

1.1. Specifications

Electrical interface:

ASY-M1250-SER2 . . . . . .RS-232C

ASY-M1250-SER2L . . . . . .Long line RS-232

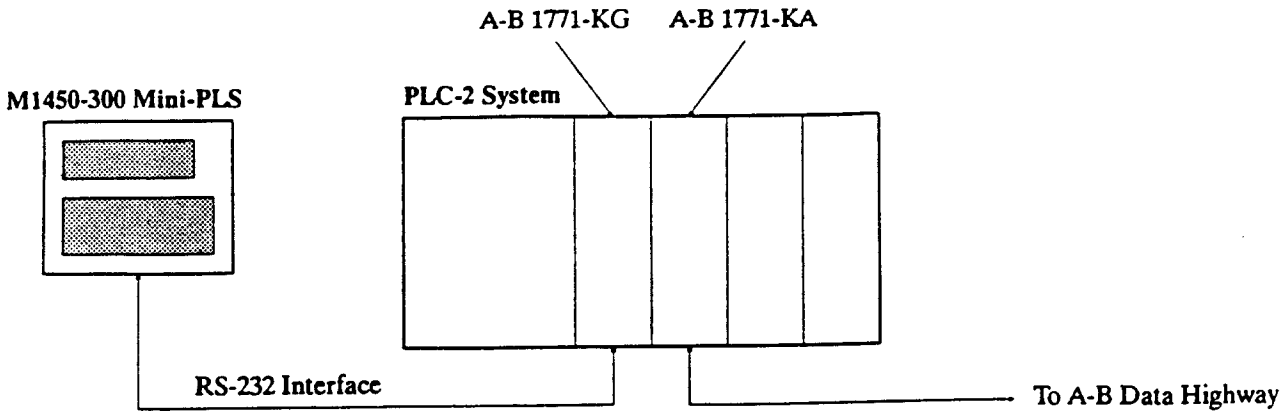
ASY-M1250-SER4 . . . . . RS-422

Communications protocol . . . . . Full duplex (peer-to-peer)

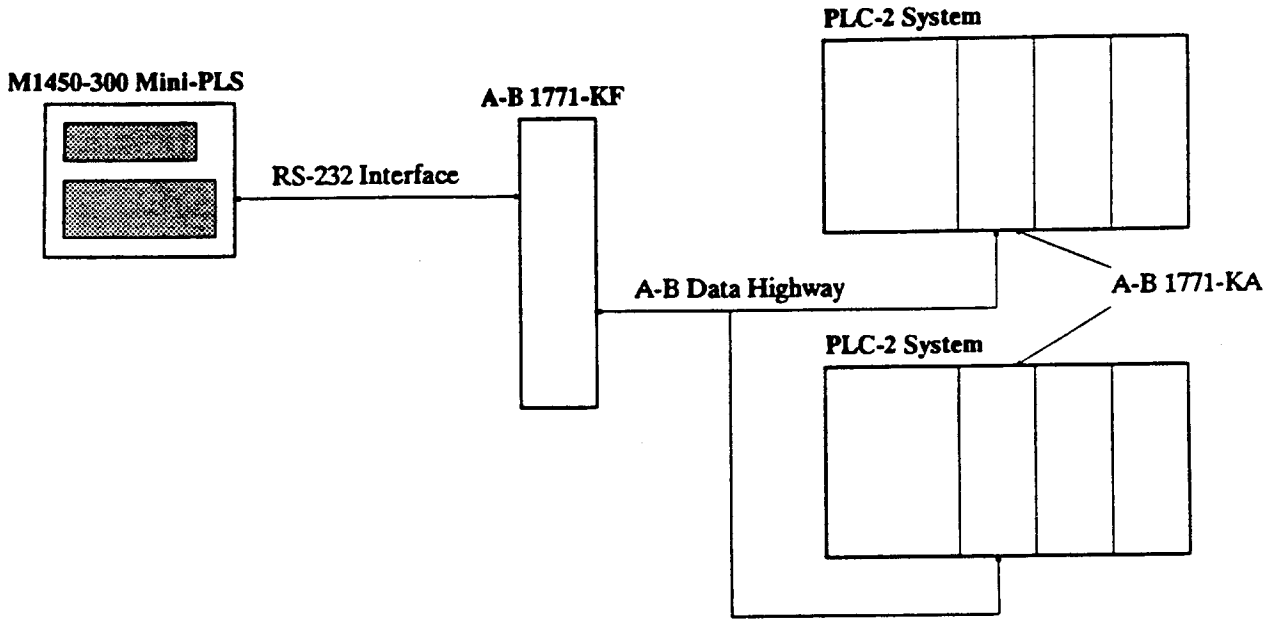
Communication rates . . . . . 110, 300, 600, 1200, 2400,4800, 9600, 19200 bits/s

Station number range . . . . . 0 - 254

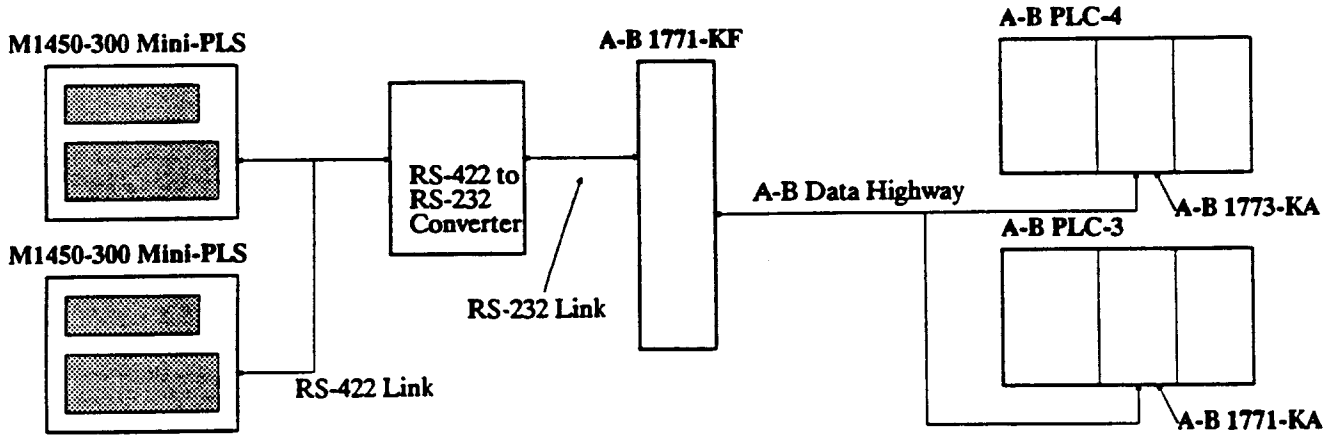
Error detection . . . . . CRC



a) Stand-alone Configuration



b) Data HighwayConfiguration



c) Multiple PLS Data HighwayConfiguration

Figure 1, Interface Configurations

Installation

2.1. 1771-KG Installation

Equipment needed:

- Allen-Bradley 1771-KG (Series B) RS-232-C Interface Module.
- PC Processor to 1771-KG Interface cable, AB 1771-CN (1.5 feet), 1771-CO (3.5 feet) or 1771-CR (10.0 feet).
- 1771-KG to M1250 Interface cable (see figure 3).
- Autotech M1450-300 Mini PLS (SAC-M1250-S10) and Serial Cam module.

Before installing the 1771-KG in the Allen-Bradley I/O chassis, the dip switches on the module must be set up properly. There are three sets of dip switches, SW1 sets up the communications rate, SW2 sets up the options and SW3, SW4 and SW5 set the station number (the switches are located on the back of the module, along the top edge, beneath a cover plate). **Make a note of the switch settings, you will need to know them when you begin programming.** See figure 2 for how to set the switches.

The communication rate setting will depend on a number of different factors. If the M1450 is being connected via a modem link, the communication rate must be equal to that of the modem. If a direct connection is used between the M1450 and another communication module, use the following chart to determine the communication rate. Note that slower communication rates will be more immune to electrical noise.

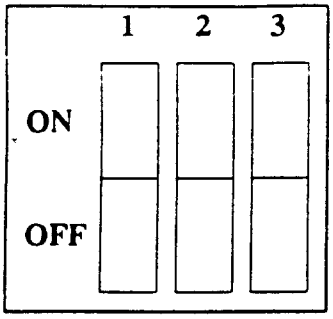
Separation (feet)	Maximum baud rate
1000	19200*
2000	9600*
3000	9600*
4000	4800
5000	4800
6000	4800
7000	2400
	* RS-422 or long-line RS-232 only

Table 1, Baud rate vs. Distance

The option switches on the 1771-KG module should be set for full-duplex protocol (peer-to-peer) protocol with CRC error checking and unprotected writes enabled (set SW2-1 off, SW2-2 on, SW2-3 off, SW2-4 on; if only one 1771-KG is being used, SW2-5 on).

Use switches SW3 - SW5 to select the station number (select a number between 10 to 77 or 110 to 376).

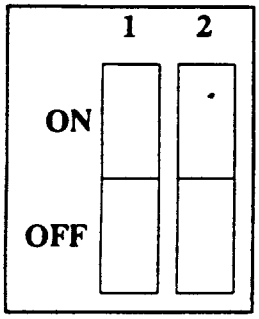
SW 1



SW 1	SW 2	SW 3	Baud rate
Off	Off	Off	110
On	Off	Off	300
Off	On	Off	600
On	On	Off	1200
Off	Off	On	2400
On	Off	On	4800
Off	On	On	9600
On	On	On	19200

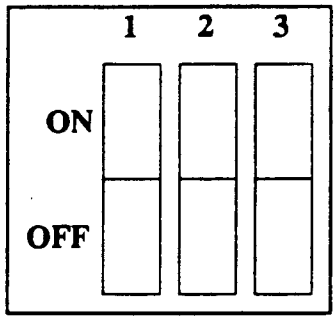
A) Communications Rate Programming

SW 3 (ms digit)



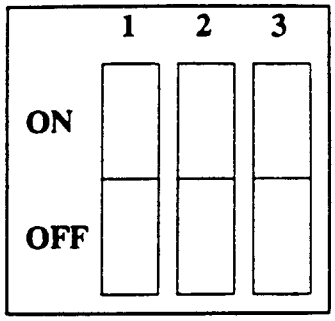
1	2	Value
Off	Off	0
Off	On	1
On	Off	2
On	On	3

SW 4



1	2	3	Value
Off	Off	Off	0
Off	Off	On	1
Off	On	Off	2
Off	On	On	3
On	Off	Off	4
On	Off	On	5
On	On	Off	6
On	On	On	7

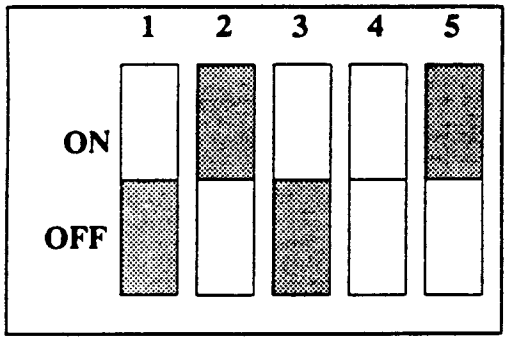
SW 5 (ls digit)



1	2	3	Value
Off	Off	Off	0
Off	Off	On	1
Off	On	Off	2
Off	On	On	3
On	Off	Off	4
On	Off	On	5
On	On	Off	6
On	On	On	7

B) Station Number Programming

SW 2



1	2	3	4	5	KG #
Off	On	Off	Off	On	1
Off	On	Off	On	On	2

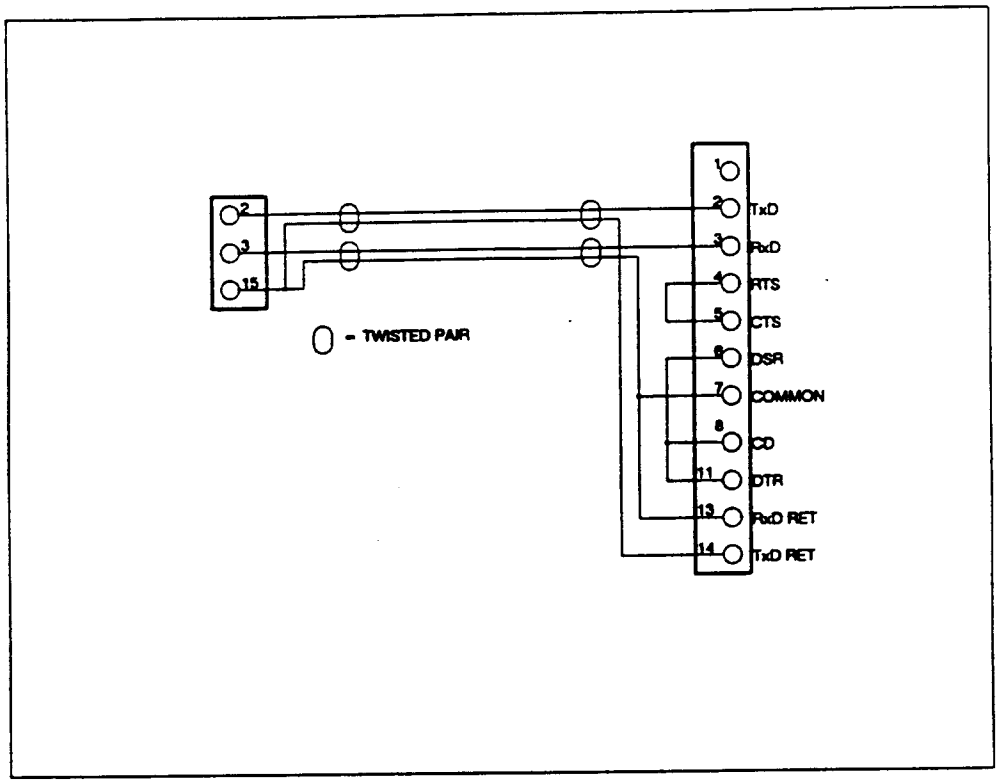
C) Protocol Selection

Figure 2, 1771-KG Configuration

The AB 1771-KG Module can be plugged into any slot (except the left-most) of a Bulletin 1771 I/O Chassis (keying plugs are provided with the 1771-KG module).

2.2. M1450 Installation

The M1450 Serial Cam Module may be installed in slot 4 or slot 5 of the M1250 chassis. The cable from the 1771-KG's RS-232-C port is wired to the back of the serial cam module. See figure 3 for cable wiring.



3. Programming

3.1. M1450 Configuration

After the module is installed, the station number and the communication rate must be programmed into the M1450.

To program the station number, first select the POSITION display. Press RECALL. The display will show "S1" in the channel window, and the current station number in the Position/RPM window. Use the "+" or "-" key or the numeric keys to select a new station number. When the desired station number is in the display, press POSITION to program the number into the M1450's nonvolatile memory.

The station number of the M1450 must be different from the station number selected for the 1771-KG. (The station number of the 1771-KG is set with SW-3, SW-4 and SW-5 on the 1771-KG module, see Figure 2).

To program the communication (baud) rate, press RECALL again. The display will show "S2" in the channel window and the current communication rate in the Position/RPM window. Use the "+" or "-" key to select the new communication rate, and then press POSITION to program the new communication rate. Note that the actual communication rate is 10 times the displayed number ("11" = 110 baud, "30" = 300 baud, "960" = 9600 baud...).

The M1450 communication rate must be the same as that selected for the 1771-KG (SW1 on the 1771-KG, see figure 2).

The M1450 Serial Cam Module is also available with RS-422 type outputs. The connections for that module are listed below.

Pin number	Signal Name
1	$\overline{\text{Received Data}}$
2	Received Data
3	Transmitted Data
4	$\overline{\text{Transmitted Data}}$
5	Signal Reference

Table 2, RS-422 Pinout

Pin number	Signal Name
2	Received Data
3	Transmitted Data
15	Signal Reference

Table 3, RS-232 Pinout

3.2. Serial Link Programming

This section first describes the commands used to interface with the M1450. The commands will be used in several sample programs. **The programming enable jumper on the M1450 has no effect on serial operations (programming via the serial link cannot be locked out at the M1450 end).**

The M1450-300 supports the following serial operations:

- Get status
- Clear channel
- Set offset
- Select channel
- Download
- Get channel setpoints
- Set channel setpoints
- Get motion limits
- Set motion limits

When you design your ladder program for the PC, you will manipulate certain bits to send the above commands to the M1450. Data returned from the M1450 will be stored in areas designated by your program. You can then use this data to monitor or modify the operation of the M1450.

In order for your program to do any communicating, you must insert a "communications zone" into your program to allow the 1771-KG serial module to function. The communications zone (defined by Allen-Bradley) must be in the special format explained below.

The HEADER rung starts the communications zone. The address of the first GET instruction is the "local" station number (the station number of the 1771-KG module). In this example, the local station number is 010. The data at this address is ignored (shown by XXX).

The address of the second GET instruction tells the 1771-KG where to store error information (in word 077 for this example). The data at this address (shown by EEE) is the current status or error word.

The address of the third GET instruction is the communications time-out code. If the 1771-KG module doesn't get a reply from the M1450 within the period specified, an error condition is generated. In this example, the time-out code 015 represents 5 seconds. A time-out code of 010 will disable the time-out function completely (the time-out code is figured by adding 8 to your desired time-out (in seconds) and converting to octal, e.g. 8 + 5 seconds = 13, which is 15 in octal). The header rung MUST end with a LATCH 02707 instruction.

Header rung:

The second rung of the communications zone is a command rung. You will have one command rung for each different command you will want to send to the M1450.

The first EXAMINE ON in the command rung is the START BIT. When this bit is set by your program, the corresponding command rung will be executed.

The second EXAMINE ON in the command rung tells the 1771-KG two things:

- The word address (020 in the example) tells the AB to which (remote) station this command is to be sent. The number you program here must be the same as the number you programmed into the M1250 (converted to octal).
- The bit address (01) tells the AB that we are reading data FROM the M1250. A bit address of 03 tells the AB that we are sending data TO the M1250. The bit address must always be 01 or 03.

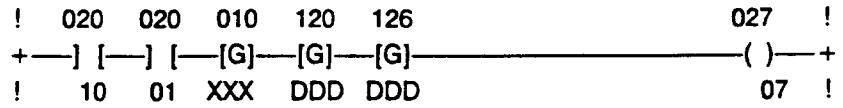
The address of the first GET instruction in the command rung is the serial COMMAND to be sent to the M1250. The data at this address (shown by XXX) is ignored. In this example, the command is 010, a GET STATUS command.

The addresses of the second and third GET instructions tell the AB where to put the received data (in the case of a read command) or where to get the data to be sent (in the case of a write command). The addresses you program here specify a BLOCK of memory - the first address is the BEGINNING of that block and the second address is the END of that block. DDD represents the data to be sent or received.

In this example, the status data we receive from the M1250 will be stored in words 120 to 126.

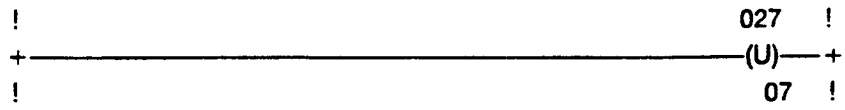
Each command rung must end with an ENERGIZE 02707 instruction.

Command rung:



The delimiter rung is used to end the communications zone and must be in the form shown below.

Delimiter rung:

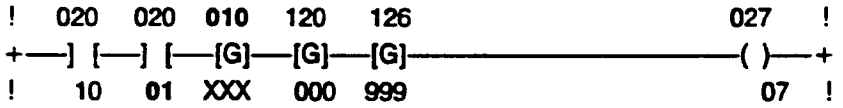


The commands and their expected replies are explained below.

3.2.1. Get Status

The Get Status command returns the current M1450 status to the PC. It is a block **READ** command (command number = **010**). This function returns up to 7 words of status information in the order listed below. Note that all the status words need not be read. If a program only requires RPM data, just read only two words. Reading three words will return RPM and position data.

Example GET STATUS command rung:



After this command is completed, the following data will be in the PLC's data table:

Word 120: 1000's digit of the tach reading (RPM).

Word 121: 100's, 10's and 1's digits of the tach (RPM).

Word 122: Current position, 0 to scale factor (999 max)

Word 123: Machine offset, 0 to scale factor (999 max)

Word 124: Status, encoded as follows:

Bit 0 = End of channel: set when the last channel setpoint is found during the read channel function. Cleared by a Select Channel Number command.

Bit 1 = Busy: a function is already in progress.

All other bits = 0.

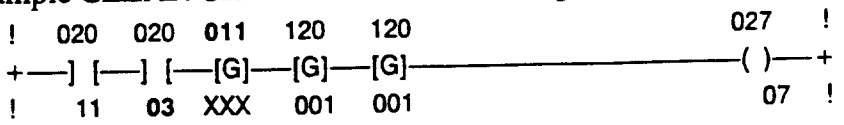
Word 125: Channel number for the read setpoints function

Word 126: Scale factor (999 in example).

3.2.2. Clear Channel

The Clear Channel command clears (erases all setpoints from) a cam channel. This is a block **WRITE** command (command number = **011**). The channel number is the first word of the data field. For large scale factors the clear channel command can take up to 65 seconds to complete, so the automatic time-out function of the 1771-KG module should be disabled (by setting the time-out code to 010).

Example CLEAR CHANNEL command rung:

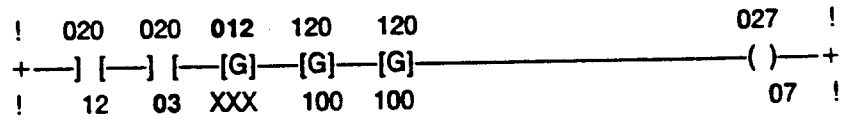


Word 120 = Channel to be cleared (Channel #1 in the example).

3.2.3. Set Offset

The Set Offset command programs a new machine offset into the M1250. This is a block **WRITE** command (command number = **012**). The new machine offset is the first word of the data field. Note that a machine offset of 000 performs an "autozero" function on the M1450.

Example SET MACHINE OFFSET command rung:

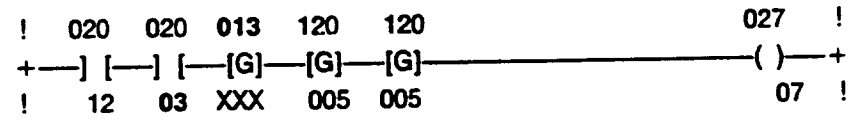


Word 120 = New machine offset (offset = 100 in the example).

3.2.4. Select Channel

The Select Channel command selects the channel number for the next read channel setpoints command. This is a block **WRITE** command (command number = **013**). The new channel number is the first word of the data field. Note that this command only affects the channel setpoint read function.

Example SELECT CHANNEL command rung:



Word 120 = Next channel to be read (channel #5 in the example).

3.2.5. Download

The Download command allows you to combine many write commands into one. This is a block **WRITE** command (command number = **045**). The general format for the data field is shown below. Note that the maximum data field size is 122 words. Since this command can take a long time to complete, the automatic time-out function of the 1771-KG module should be disabled (by setting a time-out code of 010).

Word 1: Command length (number of words that make up the following command string).

Word 2: Command, valid commands are:

11 - Clear channel

12 - Set machine offset

55 - Program channel

56 - Clear then program channel

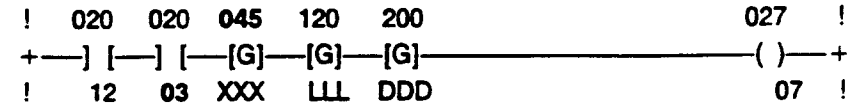
65 - Set motion limits.

Word 3: First word of command data.

Word 3 + N: Last word of command data (N words long).

After the last word of the command data, a new command field begins (all command fields begin with the command length).

Example DOWNLOAD command rung:



Word 120 = Start of first command string (length LLL)

Word 121 = First command (command number DDD)

Words 122 through 122 + N = data for first command (N words)

Word 122 + N + 1 = Start of second command string

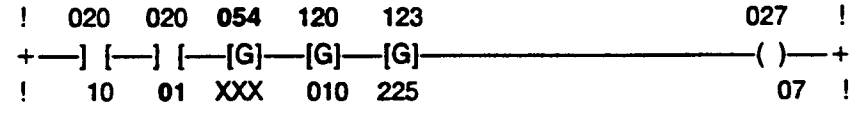
NOTE, see example program #1 for further clarification.

3.2.6. Get Channel Setpoints

The Get Channel Setpoints command reads the setpoints from the selected channel. This is a block **READ** command (command number = **054**). This function returns the setpoints of the channel number set by the SELECT CHANNEL command. This function only returns the number of setpoints specified.

The user should check the end of channel status bit after each read command. When the end of channel bit gets set, the user knows the entire channel has been read. If the user expects more setpoints than are actually stored, the unused setpoints will be returned as zeros.

Example GET CHANNEL SETPOINTS command rung:



Word 120 = First ON angle (10 in example)

Word 121 = OFF angle

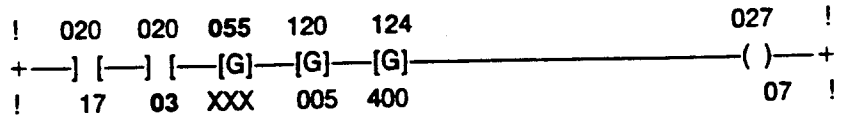
Word 122 = ON angle

Word 123 = OFF angle (225 in example).

3.2.7. Set Channel Setpoints

The Set Channel Setpoints command programs new setpoints into a cam channel. This is a block **WRITE** command (command number = **055**). The first word of the data field specifies the number of setpoints to be programmed, the second word contains the channel number, and the remaining words are the new setpoints. There must be an even number of setpoints, and the ON angle must be first. Depending on scale factor, the length and number of setpoints being programmed, the program channel command can take a long time to complete, so the automatic time-out of the 1771-KG module should be disabled (by setting the time-out code to 010), except when fine tuning a set point.

Example SET CHANNEL SETPOINTS command rung:



Word 120 = Channel number

Word 121 = ON angle

Word 122 = OFF angle

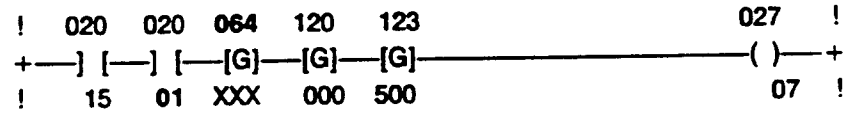
Word 123 = ON angle

Word 124 = OFF angle

3.2.8. Get Motion Limits

The Get Motion Limits command returns the current low and high motion limits programmed in the M1450. This is a block **READ** command (command number = **064**). Four words are always returned, the first pair of data words is the low motion limit and the second pair of words is the high motion limit.

Example GET MOTION LIMITS command rung:



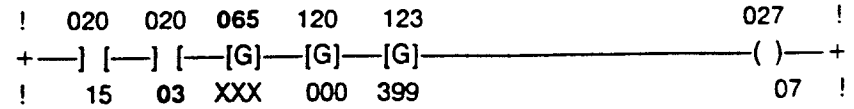
Word 120:121 = low motion limit (1000's digit in word 120)

Word 122:123 = high motion limit (1000's digit in word 122)

3.2.9. Set Motion Limits

The Set Motion Limits command programs new motion limits into the M1250. This is a block **WRITE** command (command number = **065**). The first pair of words of the data field specify the new low motion limit, and the second pair specifies the new high motion limits.

Example SET MOTION LIMITS command rung:



Word 120:121 = low motion limit (1000's digit in word 120)

Word 122:123 = high motion limit (1000's digit in word 122)

3.3. Errors

The error codes returned by the M1450 (in the STS field) are:

Error 01: Invalid command due to error in command, size or data.

Error 09: M1250 is busy executing the previous command and cannot begin another.

All commands reply after completion.

3.4. Timing

The amount of time for the commands to execute and reply depends on the command, the communication rate and the program size. The nominal time intervals given below are from the start of transmission (header rung scanned) from the 1771-KG to the end of response reception (by the 1771-KG).

All times assume the communication rate is 9600 bits/sec.

Status command (full): 50 msec.

Clear channel: depends on scale factor (see table 4).

Set offset: 100 msec.

Select channel: 35 msec.

Set motion limits: 75 msec.

Get motion limits: 40 msec.

Program channel: depends on scale factor & number of setpoints (see table 4).

Read channel: 30 msec + 10 msec per setpoint.

Download: Depends on number/type of commands and scale factor (see table 4). For example, if an M1450 channel has a setpoint of 20 to 30 degrees (scale factor = 359) and it is desired to change to 17 to 30 degrees, it would take about 3 degrees * 100 msec/degree = 300 msec.

3.5. Example Programs

The example programs demonstrate how the above commands may be implemented.

Program 1 (appendix 1) demonstrates a simple initialization sequence. Upon detecting an external switch closure, the program sets the machine offset, sets

Scale Factor	Clear/Pgm Chan (msec/degree)
50	460
100	270
200	155
300	117
359	100
400	98
500	86
600	78
700	73
800	69
900	66
999	64

Table 4, Scale Factor vs. Execution Time

the motion limits, clears and programs a number of cam channels. An output is energized when the initialization is completed.

Program 2 (appendix 2) modifies the M1450 machine offset based on speed. The program will read the tach continuously (by a Get Status command). If an external switch is on (closed), then for every 10 RPM the machine offset is advanced by one degree.

Program 3 (appendix 3) modifies M1450 cam setpoints based on speed. The program normally reads the tach continuously. If an external switch is on (closed), then for every 10 RPM, the ON angle of a cam module channel is advanced by one degree.

Each program is broken into three general sections. The first section of our program contains the program logic.

The second section of each program is the communication zone. This zone consists of a header rung, command rungs and a delimiter rung. Note that there is a separate rung for each command.

The third section of each program monitors each command for completion and resets (unlatches) the appropriate start bit.

1. Appendix 1

This program demonstrates a a procedure for loading the M1450 from an Allen-Bradley PC. The download will program the machine offset, the motion limits and a cam module. You would run this program after a new installation, to change a program, or any time you are unsure of the current setup of the M1450.

Let's say we want to program the following data into our M1450:

- Machine offset = 20
- Low motion limit = 5 RPM
- High motion limit = 1,234 RPM
- Cam channel 1 ON at 0, off at 10 degrees
- Cam channel 2 ON at 10, off at 20 degrees
- Cam channel 3 ON at 20, off at 30 degrees
- Cam channel 4 ON at 30, off at 40 degrees
- Cam channel 5 ON at 40, off at 50 degrees
- Cam channel 6 ON at 50, off at 60 degrees
- Cam channel 7 ON at 60, off at 70 degrees
- Cam channel 8 ON at 70, off at 80 degrees.

Our program will wait for an external switch closure, then send the commands to the M1450. After the M1450 completes the initialization, our program will energize an output to tell us we're all done.

We can break our program into three general sections. The first section of the program contains the program logic, the second section of the program is the communications zone and the third section of the program monitors each command for completion and resets (unlatches) the appropriate start bit.

First, let's figure out where we'll store our data in the Allen-Bradley's data table. Refer to the data table configuration figure for your PC, which shows the data table organization.

Let's start with the communications zone. We'll need a word to hold the START and DONE bits. We'll choose the first available word of bit/word storage, address 020. AB has defined the upper 8 bits of the word as START bits, and the lower 8 bits as DONE bits. The Allen-Bradley automatically uses the next address (word 021) as the remote/local fault storage word.

Start bit word = 020, bit 020.10 = DOWNLOAD function start bit

Done bit word = 020, bit 020.00 = DOWNLOAD function done bit

Local fault word = 021 (bit 021.00)

Remote fault word = 021 (bit 021.10)

We also need to set up a word for fault information storage. We'll use word 077 in this example. Other information we need to know is the station numbers of the M1450 and the Allen-Bradley. These numbers were set up at installation (see the installation section of this manual).

Error word = 077

Local station number = 8 (010 octal)

Remote (M1450) station number = 16 (020 octal)

Since the DOWNLOAD command can take a long time to complete, we will disable the back up timer for the 1771-KG by setting its value = 010 octal (you may program a separate timer to monitor the communications if desired).

Timeout value = disabled (010)

The last information we need for the communications zone is the data to be sent to the M1450. For this, we need a large number of contiguous (adjacent) word storage. From the data table organization figure, we see that a large block of memory starts a word 120, so this is where we'll store our data.

Program data organization:

Offset command

Word 120 command length (2 words)

Word 121 command (12 = set offset)

Word 122 new machine offset (020)

Set motion limits command

Word 123 command length (5 words)

Word 124 command (65 = set motion limits)

Word 125-6 new motion low limit (0005)

Word 127-30 new motion high limit (1234)

Program channel 1

Word 131 command length (4 words)

Word 132 command (56 = clear/program channel)

Word 133 channel number (001)

Word 134 ON angle (000)

Word 135 OFF angle (010)

Program channel 2

Word 136 command length (4 words)

Word 137 command (56 = clear/program channel)

Word 140 channel number (002)

Word 141 ON angle (010)

Word 142 OFF angle (020)

Program channel 3

Word 143 command length (4 words)

Word 144 command (56 = clear/program channel)

Word 145 channel number (003)

Word 146 ON angle (020)

Word 147 OFF angle (030)

Program channel 4

Word 150 command length (4 words)

Word 151 command (56 = clear/program channel)

Word 152 channel number (004)

Word 153 ON angle (030)

Word 154 OFF angle (040)

Program channel 5

Word 155 command length (4 words)

Word 156 command (56 = clear/program channel)

Word 157 channel number (005)

Word 160 ON angle (040)

Word 161 OFF angle (050)

Program channel 6

Word 162 command length (4 words)

Word 163 command (56 = clear/program channel)

Word 164 channel number (006)

Word 165 ON angle (050)

Word 166 OFF angle (060)

Program channel 7

Word 167 command length (4 words)

Word 170 command (56 = clear/program channel)

Word 171 channel number (007)

Word 172 ON angle (060)

Word 173 OFF angle (070)

Program channel 8

Word 174 command length (4 words)

Word 175 command (56 = clear/program channel)

Word 176 channel number (008)

Word 177 ON angle (070)

Word 200 OFF angle (080)

All the above data is stored in the PC from the programming terminal using GET statements. After pressing the [G] key, enter the word address above the [G]. The data at this address will be shown below the [G] and you can alter the data by using the numeric keys. We have the DOWNLOAD switch wired to an input module which is installed in module group 1 of rack 1 (terminal 0) of the AB I/O chassis. A DOWNLOAD COMPLETE indicator is wired to an output module which is installed in module group 1 of rack 1 (terminal 0) of the AB I/O chassis.

- External switch is input 111 10.
- Download complete output is 011 00.

1.1. Program Logic

This section contains the program logic and controls the start bit for the command. The first rung tests the external switch for closure. When the switch is closed, the output indicator is turned off. Rung two latches the start bit for the INITIALIZE command. Note that once the start bit is latched, it cannot be latched again until the START and DONE bits are reset (i.e., the initialization is complete).

Reset DOWNLOAD COMPLETE output

START

Set DOWNLOAD start bit

1.2. Communication Zone

This section is the communications zone of this program. The first rung is the header rung. All communications zones MUST start with a header rung. The address of the first GET specifies the LOCAL station number. In this example, the local station number is 10 (octal). The address of the second GET is the communications error word address. The address of the third GET specifies the timeout period, or how long the Allen-Bradley will wait for the M1450 to reply.

The rung after the header rung is the COMMAND rung. The command rung contains the M1450 command. The first EXAMINE ON of the command rung is the start bit test. When the start bit is set, the EXAMINE ON becomes true

and the command rung is activated. The second EXAMINE ON specifies the REMOTE station number (which M1450 the command is to be sent to) and the command type (unprotected block read = 01, unprotected block write = 03). The address of the first GET instruction is used as the command number for the M1450. This command number tells the M1450 what to do. The next pair of GET addresses specify where to put received information or where to get information to be sent (these are addresses in the PC, not the M1250). Each command rung MUST end with an ENERGIZE 02707 instruction. The last rung of the communications zone is the delimiter rung. This rung is required and consists of a single UNLATCH 02707 instruction. The data denoted by XXX appearing below the GET instructions is not used (ignored). The data denoted by DDD appearing below the GET instructions is part of the data sent or received by its associated command.

Communications zone header rung

```
!   010    077    010                                          027    !
+---[G]----[G]----[G]-------------------------------------------(L)---+
!   XXX    XXX    XXX                                           07    !
```

DOWNLOAD command rung

```
!   020    020    045    120    200                            027    !
+---] [----] [----[G]----[G]----[G]-------------------------------( )---+
!    10     03    XXX    XXX    XXX                             07    !
```

Communications zone delimiter rung

```
!                                                               027    !
+---------------------------------------------------------------(U)---+
!                                                                07    !
```

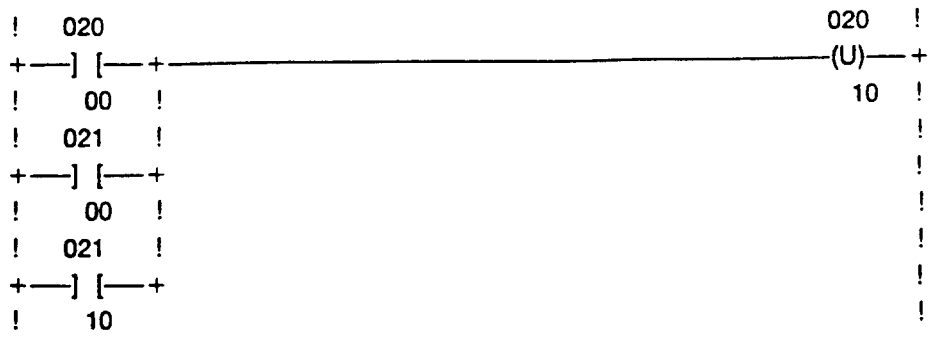
1.3. Completion Logic

This section is used to end the commands. When the command finishes, we must reset (unlatch) the start bit. In this program, the start bits will be unlatched when the DONE bit is set (true) OR when a fault (local or remote) occurs.

Turn on completion output

```
!   020                                                         011    !
+---] [-------------------------------------------------------(L)---+
!    00                                                         00    !
```

Reset DOWNLOAD start bit



Assuming that all the cam channels were empty, this program would take about 8.2 seconds to complete.

1. Appendix 2

Program 2 advances the M1450's machine offset by one degree for every increase of 10 RPM, when an external input is closed.

On each scan, if the external input is on, the M1450 tach and machine offset are read (using a Get Status command). The tach reading is then divided by 10 to get the number of degrees to advance the offset by. That number is subtracted from the 0 RPM offset (100), and the result is the new machine offset. To avoid continually changing the offset (and to make sure the offset did get changed), the new offset is check against the current offset. If the offsets do not match, the new machine offset is copied into the offset command string and the offset command is sent to the M1450.

Note that this program will only work correctly for tach readings of less than 1000 RPM. Also remember that machine offset (and cam programs) are stored in EEPROM, and these devices will 'wear out' after 10000 store cycles. For this reason, new parameters should only be stored when they have changed.

The program is broken into four general sections. The first section of the program contains the program logic. The second section of the program is the communications zone. This zone consists of a header rung, command rungs and a delimiter rung. Note that there is a separate rung for each command.

The third section of the program monitors each command for completion and resets (unlatches) the appropriate start bit.

The fourth section of the program just displays the data that is sent or received for each command.

Memory setup:

Start bit word = 020

bit 020.10 = GET STATUS function start bit

bit 020.11 = SET OFFSET function start bit

Done bit word = 020

bit 020 00 = GET STATUS function done bit

bit 020 01 = SET OFFSET function done bit

Local fault word = 021 (bits 021 00 and 021 01)

Remote fault word = 021 (bits 021 10 and 021 11)

General bit flag word = 022

bit 022 00 = set when computed offset equals current offset

bit 022 01 = dummy bit

External input = 111 10

1771-KG error word = 030

Local station number = 8 (010 octal)

Remote (M1250) station number = 16 (020 octal)

Timeout value = 5 sec

Words 120:123 = status data

Word 120:121 = RPM

Word 122 = position (not used)

Word 123 = offset

Word 124 = new offset

Word 23 = tach divisor (constant, = 10)

Word 24:25 = offset "dwell" (quotient of tach/10)

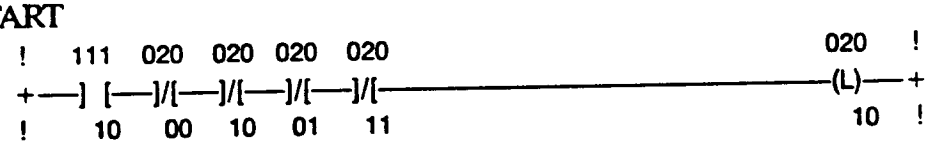
Word 26 = base offset (= 100)

1.1. Start Logic

This section controls the start bits for each of the commands. The Get Status function runs continuously so the tach reading is current. The second rung divides the tach reading by 10 to get the amount to change the offset by. The third rung subtracts the offset adjustment from the base offset (100) and puts the result in the offset command string. The fourth rung compares the new offset to the current offset, and sets (energizes) a bit only when the offsets are equal. The fifth rung sends the offset command only if the offsets were not equal. Note that the get status start rung (#1) and the set offset start rung (#5) can only be active when the external switch (bit 111 10) is on).

GET STATUS start bit rung

START



Get 10% of tach reading

(TTT = current tach data, QQQ.QQQ = "dwell")

Subtract dwell from base offset

(QQQ = 10% of current tach, XXX = new machine offset)

Compare new offset to current offset

```
!    124   123                                                  022   !
+---[G]---[=]-------------------------------------------------( )---+
!    XXX   MMM                                                   00   !
```

(XXX = new offset, MMM = current offset)

Start offset function

```
!    111   022   020   020   020   020                          020   !
+---]/[---]/[---]/[---]/[---]/[---]/[-----------------------(L)---+
!     10    00    00    10    01    11                          11   !
```

1.2. Communication Zone

This section is the communications zone of this program. The first rung is the header rung. All communications zones MUST start with a header rung. The address of the first GET specifies the LOCAL station number. In this example, the local station number is 10. The address of the second GET is the communications error word address. The address of the third GET specifies the timeout period, or how long the Allen-Bradley will wait for the M1450 to reply.

The two rungs after the header rung are COMMAND rungs. Each rung represents a different M1450 command. The first EXAMINE ON of the command rungs is the start bit test. When the start bit is set, the EXAMINE ON becomes true and the command rung is activated. The second EXAMINE ON specifies the REMOTE station number (which station the command is to be sent to) and the command type (unprotected block read = 01, unprotected block write = 03). The address of the first GET instruction is used as the command number for the M1450. The command number tells the M1250 what to do. The next pair of GET addresses specify where to put received information or where to get information to be sent (these are addresses in the PC, not the M1450). Each command rung MUST end with an ENERGIZE 02707 instruction.

The last rung of the communications zone is the delimiter rung. This rung is required and consists of a single UNLATCH 02707 instruction. The data denoted by XXX below the GET instructions is not used. The data denoted by DDD below the GET instructions is part of the data sent or received by the associated command.

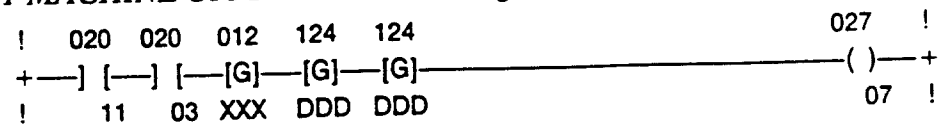
Communications zone header rung

```
!    010   030   015                                            027   !
+---[G]---[G]---[G]-------------------------------------------(L)---+
!    XXX   XXX   XXX                                             07   !
```

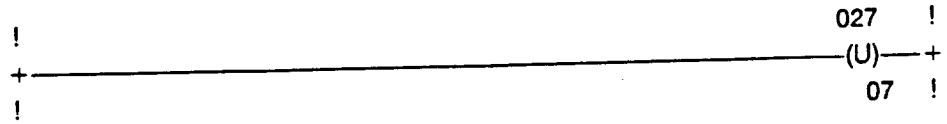
GET STATUS command rung

```
!    020   020   010   120   123                                027   !
+---] [---] [---[G]---[G]---[G]-------------------------------( )---+
!     10    01   XXX   DDD   DDD                                 07   !
```

SET MACHINE OFFSET command rung



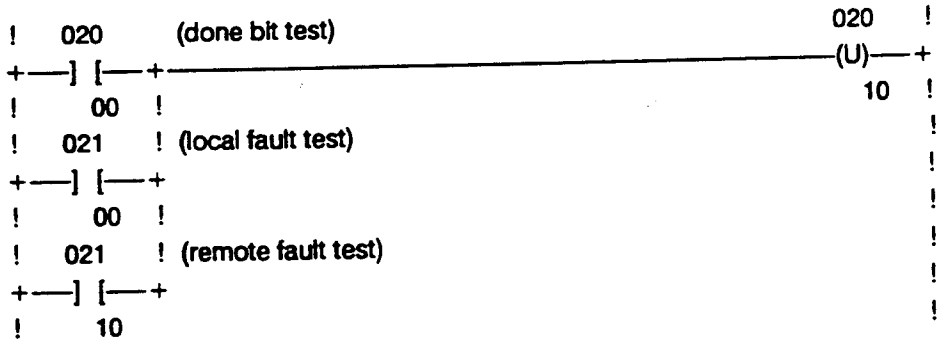
Communications zone delimiter rung



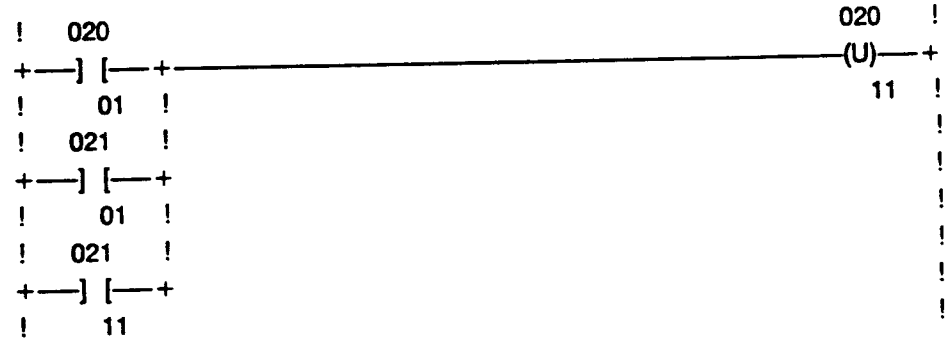
1.3. Completion Logic

This section is used to end the commands. When the command finishes, we must reset (unlatch) the start bit. In this program, the start bits will be unlatched when the DONE bit is set (true) OR when a fault (local or remote) occurs.

GET STATUS completion rung



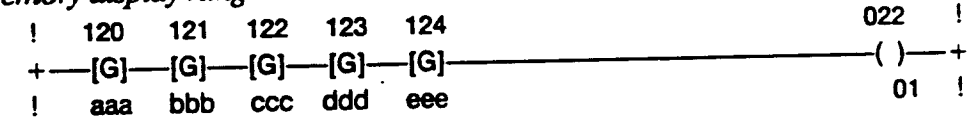
SET OFFSET completion rung



1.4. Program Data

This section is just a display/data area for the above commands.

Memory display rung



aaa:bbb = RPM

ccc = position

ddd = current machine offset

eee = programmed offset

The amount of time from a change in RPM to the new offset stored in the M1450 will be about 140 msec, but this will vary greatly depending on RPM (the higher the RPM, the 'busier' the M1450 becomes, hence it has less time to do everything).

Appendix 3

Program 3 modifies M1450 cam setpoints based on speed and an external input. On each scan, the program reads the current tach from the M1450. For each 10 RPM increment, the cam setpoint ON angle is advanced by one degree (providing an external switch is on). This program is broken into three general sections. The first section of the program contains the program logic.

The second section of the program is the communications zone. This zone consists of a header rung, command rungs and a delimiter rung. Note that there is a separate rung for each command.

The third section of the program monitors each command for completion and resets (unlatches) the appropriate start bit.

Memory setup:

Start bit word = 020 (bits 020 10 to 020 17)

bit 020 10 = GET STATUS function start bit

bit 020 11 = PROGRAM CHANNEL function start bit

Done bit word = 020 (bits 020 00 to 020 07)

bit 020 00 = GET STATUS function done bit

bit 020 01 = PROGRAM CHANNEL function done bit

Word 120:121 = current M1250 tach

Wword 122:125 = program channel command string

Word 023 = tach divisor (constant, = 10)

Word 024:025 = setpoint adjustment (quotient of tach/10)

Word 026 = base ON angle (100)

Word 031 = previous ON angle

Local fault word = 021 (bits 021 00 and 021 01)

Remote fault word = 021 (bits 021 10 and 021 11)

Bit variables = 022

bit 022 00 = change setpoints

bit 022 01 = enable channel programming

bit 022 02 = channel programming complete

bit 022 03 = dummy bit

Error word = 030

Local station number = 8 (010 octal)

Remote (M1250) station number = 16 (020 octal)

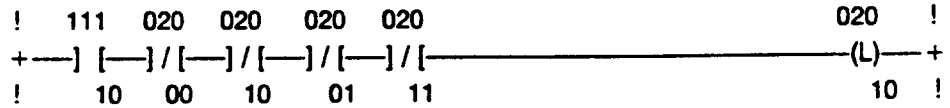
Timeout value = 5 seconds

1.1. Start Logic

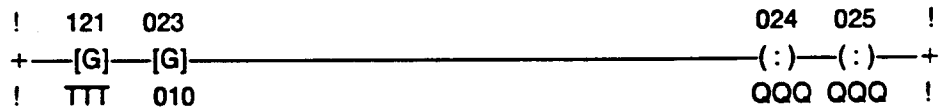
This section controls the start bits for each of the commands. The first rung starts a Get Status function continuously so the tach reading is current. The second rung computes 10% of the tach reading, the result being the "dwell" to apply to the base ON angle. The third rung subtracts the dwell angle from the base ON angle to get the new ON angle. Rung 4 compares the new ON angle to the previous ON angle. If the two angles are equal, bit 022 00 is set.

Rung 5 examines bit 022 00. If the bit is off, meaning the ON angle needs to be changed, bit 022 01 is latched, and remains latched until the program setpoints command completes. Rung 6 also examines bit 022 00, and if the bit is off, the new ON setpoint angle is copied to the program channel command string. Rung 7 examines the latched bit 022 01. If the bit is set (ON angles are not equal), the new ON angle is programmed.

GET STATUS start bit rung

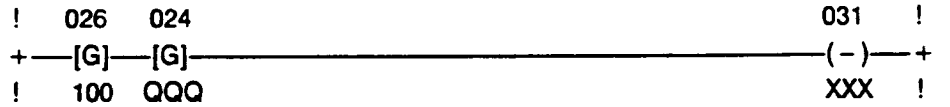


Get 10% of tach



(TTT = current tach data, QQQ.QQQ = "dwell")

Subtract dwell from base ON angle



(QQQ = dwell, XXX = new ON angle)

Compare new ON angle to old ON angle

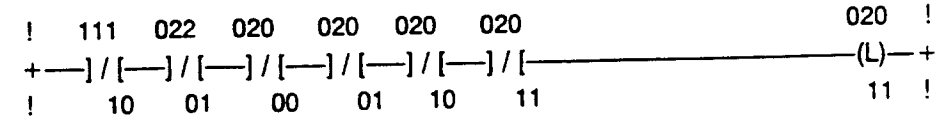
(SSS = new setpoint, XXX = previous setpoint)

Latch angle change status

Load new ON angle setpoint

(SSS = new setpoint, XXX = previous setpoint)

PROGRAM CHANNEL start bit rung



1.2. Communication Zone

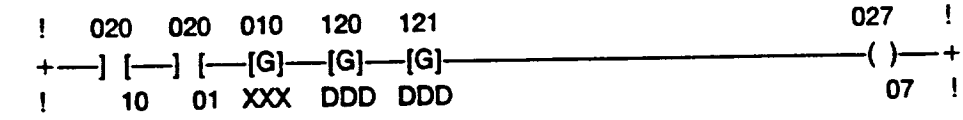
This section is the communications zone of this program. The first rung is the header rung. All communications zones MUST start with a header rung. The address of the first GET specifies the LOCAL station number. In this example, the local station number is 10. The address of the second GET is the communications error word address. The address of the third GET specifies the timeout period, or how long the Allen-Bradley will wait for the M1450 to reply.

The two rungs after the header rung are COMMAND rungs. Each rung represents a different M1450 command. The first EXAMINE ON of the command rungs is the start bit test. When the start bit is set, the EXAMINE ON becomes true and the command rung is activated. The second EXAMINE ON specifies the REMOTE station number (which station the command is to be sent to) and the command type (unprotected block read = 01, unprotected block write = 03). The address of the first GET instruction is used as the command number for the M1450. This command number tells the M1450 what to do. The next pair of GET addresses specify where to put received information or where to get information to be sent (these are addresses in the PC, not the M1450). Each command rung MUST end with an ENERGIZE 02707 instruction.

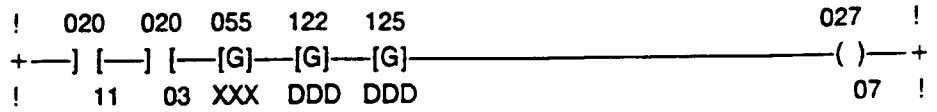
The last rung of the communications zone is the delimiter rung. This rung is required and consists of a single UNLATCH 02707 instruction. The data denoted by XXX below the GET instructions is not used. The data denoted by DDD below the GET instructions is part of the data sent or received by the associated command.

Communications zone header rung

GET STATUS command rung



PROGRAM CHANNEL command rung



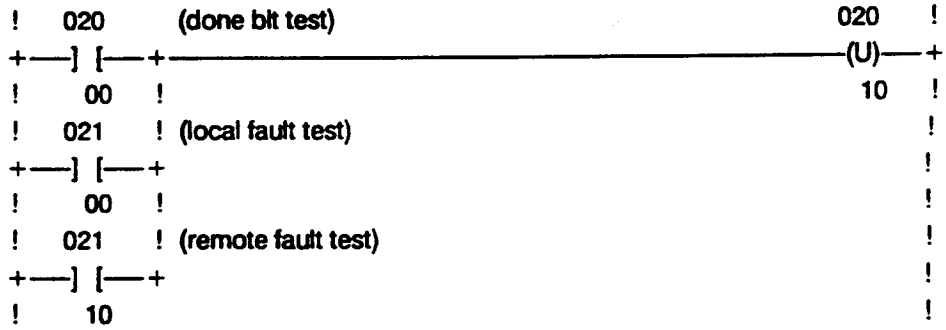
Communications zone delimiter rung

1.3. Completion Logic

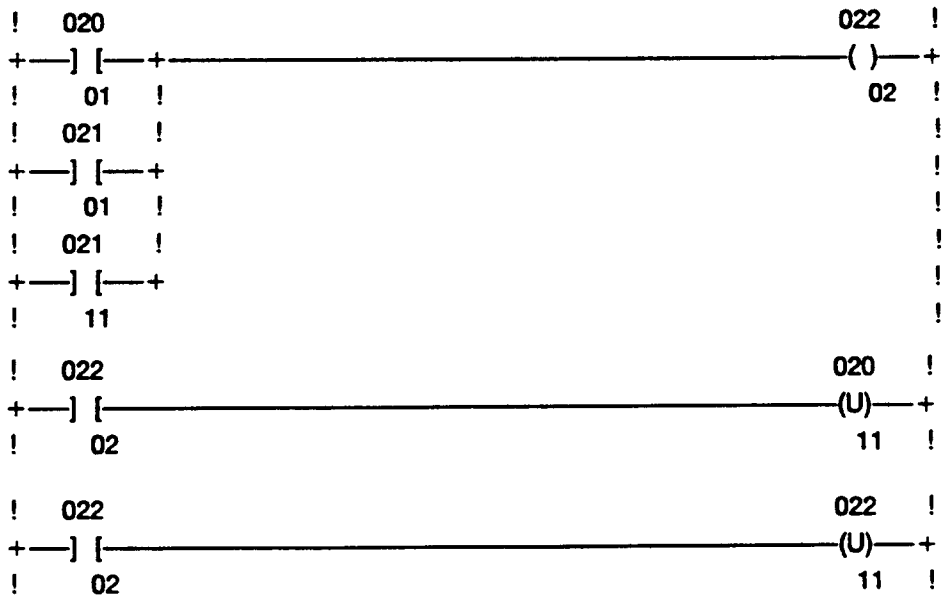
This section is used to end the commands. When the command finishes, we must reset (unlatch) the start bit. In this program, the start bits will be unlatched when the DONE bit is

set (true) OR when a fault (local or remote) occurs. In the case of the PROGRAM CHANNEL command, a bit (022 02) is set which in turn resets the start bit and unlatches the command enable bit.

GET STATUS completion rung



PROGRAM CHANNEL completion rung



The amount of time from a detected RPM change to the new channel program will be about 40 msec + 100 * the number of degrees being added or deleted. Again, this will vary for the same reasons given in the second example.

1. M1450 Serial Module Application Note

This application note covers the technical aspects of interfacing to the M1450 serial module. The serial module is intended to communicate with an Allen-Bradley 1771-KG module, which uses a non-trivial communications protocol. When using the M1450 with the KG module, the user inserts a "communications zone" in the ladder program which instructs the 1771-KG what and when to send. The serial protocol is invisible to the user. However, if you want to connect the M1450 to a different controller or computer, then you must understand the details of the protocol if you want to communicate with the M1450.

This note, as a supplement to the standard manual and the 1771-KG manual, should provide the data you need. The routines in this note are all written in the 'C' programming language for use on an IBM or compatible PC. The low-level transmit & receive routines are not included here because they are system dependent.

For this program, the transmit routine was called for each byte sent, hence a simple polling routine could be used. The receive routine for this program is interrupt-driven. The interrupt routine stores each character in a 256 byte circular buffer named 'rx_buffer'. Another simple routine is required for initializing the UART. The UART should be configured for an 8 bit/word, no parity, 1 stop bit and the desired baud rate. The baud rate at both ends of the communications link must be equal.

```
/* Define communications link control characters */

#define   DLE                  0x10    /* Data Link Escape */
#define   STX                  0x02    /* Start of Text */
#define   ETX                  0x03    /* End of Text */
#define   ENQ                  0x05    /* Enquire */
#define   ACK                  0x06    /* Acknowledge (ok) */
#define   NAK                  0x15    /* Not Acknowledge (not ok) */
#define   UNPROT_BLK_WR_CMD    0x08    /* A-B unprotected block write cmd */
#define   UNPROT_BLK_RD_CMD    0x01    /* A-B unprotected block read cmd */

/* M1450 Serial link commands */

/* Serial link commands (translation from A-B logical to physical address (per Appendix
C.2 of the 1771-KG manual.) */

#define   STAT_CMD             0x0010  /* read 1450 status */
#define   CLR_CH_CMD           0x0012  /* clear channel */
#define   OFS_CMD              0x0014  /* program offset */
#define   SET_CH_CMD           0x0016  /* set channel # */
#define   RD_TACH_CMD          0x0068  /* read motion limits */
#define   RD_CH_CMD            0x0058  /* read channel setpoints */
```

```
#define   WR_TACH_CMD      0x006A   /* program motion limits */
#define   WR_CH_CMD        0x005A   /* program channel setpoints */
#define   DOWN_CMD         0x004A   /* download command */
#define   DOWN_CCH_CMD     0x0011   /* download subcommand- clear chan */
#define   DOWN_OFS_CMD     0x0012   /* download subcommand- set offset */
#define   DOWN_WCH_CMD     0x0055   /* download subcommand- prog chan */
#define   DOWN_CPCH_CMD    0x0056   /* download subcommand- clear & prg chan */
#define   DOWN_WT_CMD      0x0065   /* download subcommand- set motion limits */

/* Global variables... */

char      rx_buffer [256], tx_buffer [80], eom_flag;
int       input_ofs, dsply_ofs, dest_station, src_station, msg_ofs, rx_len;
unsigned  tx_crc, trans, calc_rx_crc;

/*****************************************************************************
*
* status ()
*
* Reads the current M1450 status:
*     channel
*     offset
*     position
*     scale factor
*     RPM
*     status
*
*     Command format:
*          Unprotected block read
*          addr field = STAT_CMD
*          data field = number of bytes to be returned
*
*     Example output string with local station number = 8, remote station = 16
*
*          0x10      DLE
*          0x02      STX
*          0x10      destination station
*          0x10      (DLE insertion)
*          0x08      source station
*          0x01      unprotected block read command
*          0x00      status
*          0x00      transaction # (ls)
*          0x00      transaction # (ms)
*          0x10      M1450 status command (ls)
*          0x10      (DLE insertion)
*          0x00      M1450 status command (ms)
*          0x0E      number of bytes to be read
```

```
*          0x10      DLE
*          0x03      ETX
*          0xE7      checksum (ls)
*          0x38      checksum (ms)
*****************************************************************************/

int        status ()
{
     int          data_buff [32];
     unsigned     status, channel, offset, position, scale_factor, rpm;

/* Reset values to 000 */

     status = channel = offset = position = scale_factor = rpm = 0;

/* Set the number of bytes to be returned by the M1450:
     status:          1 A-B word = 2 bytes
     channel:         1 A-B word = 2 bytes
     offset:          1 A-B word = 2 bytes
     position:        1 A-B word = 2 bytes
     scale factor:    1 A-B word = 2 bytes
     rpm:             2 A-B words = 4 bytes
*/

     data_buff [0] = 14;

/* Send the status command (data field is 1 byte long) */

     send (STAT_CMD, 0, 'r', 1, data_buff);

/* Wait for response from M1450 */

     if ((temp = rx ()) && temp != NAK) {
          next ();          /* fetch destination station # */
          next ();          /* fetch source station # */
          next ();          /* fetch command */
          next ();          /* fetch status */
          next ();          /* fetch tns low */
          next ();          /* fetch tns high */

/* Convert RPM from two 3-digit BCD words to one packed BCD word */

          rpm = next () < 12;
          next ();
          rpm |= next ();
          rpm |= next () < 8;
```

```
/* Combine POSITION into one word */

        position = next ();
        position = position | (next () < 8);

/* Combine OFFSET into one word */

        offset = next ();
        offset = offset | (next () < 8);

/* Combine STATUS into one word */

        status = next ();
        status = status | (next () < 8);

/* Combine CHANNEL into one word */

        channel = next ();
        channel = channel | (next () < 8);

/* Combine SCALE FACTOR into one word */

        scale_factor = next ();
        scale_factor = scale_factor | (next () < 8);
    }
    return (TRUE);
}

/*****************************************************************************
*
* get_motion_limits ()
*     Reads the current motion limits from the M1450.
*     Command format:
*         Unprotected block read
*         addr field = RD_TACH_CMD
*         data field = number of bytes to be returned (8)
*
*     Example output string, local station number = 8, remote station = 16
*
*         0x10      DLE
*         0x02      STX
*         0x10      destination station
*         0x10      (DLE insertion)
*         0x08      source station
*         0x01      unprotected block read command
*         0x00      status
*         0x01      transaction # (ls)
```

```
*          0x00      transaction # (ms)
*          0x68      M1450 read motion limits command (ls)
*          0x00      M1450 read motion limits command (ms)
*          0x08      number of bytes to be read
*          0x10      DLE
*          0x03      ETX
*          0xFD      checksum (ls)
*          0xE9      checksum (ms)
*
*****************************************************************************/
int       get_motion_limits ()
{
     int       data_buff [32];
     unsigned  tach_lo, tach_hi;

/* Reset data to 000 */

     tach_lo = tach_hi = 0;

/* Set the number of bytes to be returned by the M1450:
     low motion limit:     2 A-B words = 4 bytes
     high motion limit:    2 A-B words = 4 bytes
*/

     data_buff [0] = 8;

/* Send the command (data field is 1 byte long) */

     send (RD_TACH_CMD, 0, 'r', 1, data_buff);
     if (rx ()) {
          next ();          /* fetch destination station # */
          next ();          /* fetch source station # */
          next ();          /* fetch command */
          next ();          /* fetch status */
          next ();          /* fetch tns low */
          next ();          /* fetch tns high */

/* Convert low limit from two A-B BCD words to one packed BCD word */

          tach_lo = next () < 12;
          next ();
          tach_lo |= next ();
          tach_lo |= next () < 8;

/* Do the same for the high limit */

          tach_hi = next () < 12;
```

```
            next ();
            tach_hi | = next ();
            tach_hi | = next () < 8;
        }
        return (TRUE);
}

/*****************************************************************************
*
* set_motion_limits (new_motion_high, new_motion_low)
*
*     Sets the M1450 motion limits.
*
*          new_motion_low  < new_motion_high < 1900
*
*          0 < new_motion_low  < new_motion_high
*
*     Command format:
*          Unprotected block write
*          addr field = WR_TACH_CMD
*          data field = motion setpoints
*
*     Example output string, local station number = 8, remote station = 16,  low limit = 12,
*     high limit = 1234:
*
*          0x10      DLE
*          0x02      STX
*          0x10      destination station
*          0x10      (DLE insertion)
*          0x08      source station
*          0x08      unprotected block write command
*          0x00      status
*          0x02      transaction # (ls)
*          0x00      transaction # (ms)
*          0x6A      M1450 set motion limit command (ls)
*          0x00      M1450 set motion limit command (ms)
*          0x00      lsbyte of msword of low limit
*          0x00      msbyte of msword of low limit
*          0x23      lsbyte of lsword of low limit
*          0x01      msbyte of lsword of low limit
*          0x01      lsbyte of msword of hi limit
*          0x00      msbyte of msword of hi limit
*          0x34      lsbyte of lsword of hi limit
*          0x02      msbyte of lsword of hi limit
*          0x10      DLE
*          0x03      ETX
*          0xF4      checksum (ls)
```

```
*           0xE2        checksum (ms)
*
*****************************************************************************/

int         set_motion_limits (new_motion_high, new_motion_low)
int         new_motion_high, new_motion_low;
{
    int     data_buff [32];

/* Split the motion low data into two words */

    data_buff [1] = new_motion_low % 1000;
    data_buff [0] = new_motion_low / 1000;

/* Split the motion high data into two words */

    data_buff [3] = new_motion_high % 1000;
    data_buff [2] = new_motion_high / 1000;

/* Send the command (data field is 4 bytes long) */

    send (WR_TACH_CMD, 0, 'w', 4, data_buff);
    rx ();
    return (TRUE);
}

/*****************************************************************************
*
* set_offset (new_offset)
*     Sets the M1450 offset.
*
*     0 < new_offset < 1000
*
*     Command format:
*         Unprotected block write
*         addr field = OFS_CMD
*         data field = new offset
*
*     Example output string, local station number = 8, remote station = 16, new offst = 123
*
*           0x10      DLE
*           0x02      STX
*           0x10      destination station
*           0x10      (DLE Insertion)
*           0x08      source station
*           0x08      unprotected block write command
*           0x00      status
```

```
*          0x03      transaction # (ls)
*          0x00      transaction # (ms)
*          0x14      M1450 set offset command (ls)
*          0x00      M1450 set offset command (ms)
*          0x23      lsbyte of new offset
*          0x01      msbyte of new offset
*          0x10      DLE
*          0x03      ETX
*          0x01      checksum (ls)
*          0xB2      checksum (ms)
*
*****************************************************************************/

int        set_offset (new_offset)
int        new_offset;
{
     int   data_buff [32];

     data_buff [0] = new_offset;

/* Send the data command (data field is 1 byte long) */

     send (OFS_CMD, 0, 'w', 1, data_buff);
     rx ();
     return (TRUE);
}

/*****************************************************************************
*
* set_channel_number (new_channel)
*
*    1 < new_channel < 40
*
*    Command format:
*         Unprotected block write
*         addr field = SET_CH_CMD
*         data field = channel number
*
*    Example output string, local station number = 8, remote station = 16, new channel = 5
*
*          0x10      DLE
*          0x02      STX
*          0x10      destination station
*          0x10      (DLE insertion)
*          0x08      source station
*          0x08      unprotected block write command
*          0x00      status
```

```
*
*           0x04        transaction # (ls)
*           0x00        transaction # (ms)
*           0x16        M1450 set channel command (ls)
*           0x00        M1450 set channel command (ms)
*           0x05        lsbyte of new channel number
*           0x00        msbyte of new channel number
*           0x10        DLE
*           0x03        ETX
*           0xEE        checksum (ls)
*           0xE9        checksum (ms)
*
*****************************************************************************/

int         set_channel_number (new_channel)
int         new_channel;
{
    int     data_buff [32];

    data_buff [0] = new_channel;

/* Send the command (data field is 1 byte long) */

    send (SET_CH_CMD, 0, 'w', 1, data_buff);
    rx ();
    return (TRUE);
}

/*****************************************************************************
*
* clear_channel (channel)
*     Clears all the setpoint data for the given channel.
*
*     1 <  channel  < 40
*
*     Command format:
*           Unprotected block write
*           addr field = CLR_CH_CMD
*           data field = channel to be cleared
*
*     Example output string, local station number = 8, remote station = 16, channel = 3
*
*           0x10        DLE
*           0x02        STX
*           0x10        destination station
*           0x10        (DLE insertion)
*           0x08        source station
*           0x08        unprotected block write command
```

```
 *          0x00        status
 *          0x05        transaction # (ls)
 *          0x00        transaction # (ms)
 *          0x12        M1450 clear channel command (ls)
 *          0x00        M1450 clear channel command (ms)
 *          0x03        lsbyte of channel number
 *          0x00        msbyte of channel number
 *          0x10        DLE
 *          0x03        ETX
 *          0xEF        checksum (ls)
 *          0xE8        checksum (ms)
 *
 ******************************************************************************/

int         clear_channel (channel)
int         channel;
{
    int   data_buff [32];

    data_buff [0] = channel;

/* Send the command (data field is 1 byte long) */

    send (CLR_CH_CMD, 0, 'w', 1, data_buff);
    rx ();
    return (TRUE);
}

/******************************************************************************
 *
 * read_channel_setpoints (count)
 *     Reads 'count' setpoints from the currently selected
 *     channel.
 *
 *     1 < count < 20
 *
 *     Command format:
 *          Unprotected block read
 *          addr field = READ_CH_CMD
 *          data field = number of setpoints to be returned
 *
 *     Example output string, local station number = 8, remote station = 16, 8 setpoints
 *
 *          0x10        DLE
 *          0x02        STX
 *          0x10        destination station
 *          0x10        (DLE insertion)
```

```
*          0x08      source station
*          0x01      unprotected block read command
*          0x00      status
*          0x06      transaction # (ls)
*          0x00      transaction # (ms)
*          0x58      M1450 read setpoints command (ls)
*          0x00      M1450 read setpoints command (ms)
*          0x10      number of bytes to be read
*          0x10      (DLE insertion)
*          0x10      DLE
*          0x03      ETX
*          0xF9      checksum (ls)
*          0x5E      checksum (ms)
*
*****************************************************************************/

int       read_channel_setpoints (count)
int       count;
{
     char status;
     int  i, num_spts, data_buff [32];

/* Convert number of setpoints to number of bytes */

     data_buff [0] = num_spts = count * 2;

/* Send the command (data field is 1 byte long) */

     send (RD_CH_CMD, 0, 'r', 1, data_buff);
     if (rx ()) {
          next ();          /* fetch destination */
          next ();          /* fetch source */
          next ();          /* fetch command */
          status = next ();      /* check status */
          if (status) {
               puts ("Reply STATUS = Error");
               return (FALSE);
          }
          next ();          /* fetch tns low */
          next ();          /* fetch tns high */

          for (i = 0; i  num_spts / 2; i++) {
               setpoints [i] = next ();
               setpoints [i] |= (next () < 8);
          }
     }
     return (TRUE);
```

```
}

/*****************************************************************************
*
* set_channel_setpoints (channel, setpoints,
* num_setpoints)
*     Programs new setpoints for the given channel.
*
*     1 < channel < 40
*     0 < setpoint < scale_factor
*
*     Command format:
*          Unprotected block write
*          addr field = CLR_CH_CMD
*          data field = channel #, setpoint, setpoint, ...
*
*     Example output string, local station number = 8, remote station = 16,
*     channel 4, 0 - 45, 45 - 90, 90 - 180
*
*          0x10      DLE
*          0x02      STX
*          0x10      destination station
*          0x10      (DLE insertion)
*          0x08      source station
*          0x08      unprotected block write command
*          0x00      status
*          0x07      transaction # (ls)
*          0x00      transaction # (ms)
*          0x5A      M1450 read motion limits command (ls)
*          0x00      M1450 read motion limits command (ms)
*          0x04      lsbyte of channel number
*          0x00      msbyte of channel number
*          0x00      lsbyte of setpoint
*          0x00      msbyte of setpoint
*          0x45      lsbyte of setpoint
*          0x00      msbyte of setpoint
*          0x45      lsbyte of setpoint
*          0x00      msbyte of setpoint
*          0x90      lsbyte of setpoint
*          0x00      msbyte of setpoint
*          0x90      lsbyte of setpoint
*          0x00      msbyte of setpoint
*          0x80      lsbyte of setpoint
*          0x01      msbyte of setpoint
*          0x10      DLE
*          0x03      ETX
*          0x3C      checksum (ls)
```

```
*          0xD5      checksum (ms)
* ************************************************************************************/

int        set_channel_setpoints (channel, setpoints, num_setpoints)
int        channel, *setpoints, num_setpoints;
{
     int   i, data_buff [32];

     data_buff [0] = channel;
     for (i = 0; i  num_setpoints; i++)
          data_buff [i + 1] = *setpoints++;

     send (WR_CH_CMD, 0, 'w', i + 1, data_buff);
     rx ();
     return (TRUE);
}

/************************************************************************************
*
* download ()
*     Download a fixed command set
*
*     Command format:
*          Unprotected block write
*          addr field = DOWN_CMD
*          data field = command list
*          (see DOWNLOAD section of M1450 serial manual
*          for valid commands)
*
*     Example output string, local station number = 8, remote station = 16
*
*          0x10      DLE
*          0x02      STX
*          0x10      destination station
*          0x10      (DLE insertion)
*          0x08      source station
*          0x08      unprotected block write command
*          0x00      status
*          0x0A      transaction # (ls)
*          0x00      transaction # (ms)
*          0x4A      M1450 download limits command (ls)
*          0x00      M1450 download command (ms)
*
*          0x02      lsbyte of byte count/offset command
*          0x00      msbyte of byte count
*          0x12      lsbyte of command
*          0x00      msbyte of command
```

```
*       0x20    lsbyte of data
*       0x00    msbyte of data
*
*       0x05    lsbyte of byte count/set motion limits command
*       0x00    msbyte of byte count
*       0x65    lsbyte of command
*       0x00    msbyte of command
*       0x00    lsbyte of data
*       0x00    msbyte of data
*       0x05    lsbyte of data
*       0x00    msbyte of data
*       0x01    lsbyte of data
*       0x00    msbyte of data
*       0x34    lsbyte of data
*       0x02    msbyte of data
*
*       0x04    lsbyte of byte count/program channel
*       0x00    msbyte of byte count
*       0x56    lsbyte of command
*       0x00    msbyte of command
*       0x01    lsbyte of data (channel number)
*       0x00    msbyte of data
*       0x00    lsbyte of data (setpoint)
*       0x00    msbyte of data
*       0x10    lsbyte of data (setpoint)
*       0x10    (DLE insertion)
*       0x00    msbyte of data
*
*       0x04    lsbyte of byte count/program channel
*       0x00    msbyte of byte count
*       0x56    lsbyte of command
*       0x00    msbyte of command
*       0x02    lsbyte of data (channel number)
*       0x00    msbyte of data
*       0x10    lsbyte of data (setpoint)
*       0x10    (DLE insertion)
*       0x00    msbyte of data
*       0x20    lsbyte of data (setpoint)
*       0x00    msbyte of data
*
*       0x04    lsbyte of byte count/program channel command
*       0x00    msbyte of byte count
*       0x56    lsbyte of command
*       0x00    msbyte of command
*       0x03    lsbyte of data (channel number)
*       0x00    msbyte of data
*       0x20    lsbyte of data (setpoint)
```

```
*          0x00      msbyte of data
*          0x30      lsbyte of data (setpoint)
*          0x00      msbyte of data
*          0x10      DLE
*          0x03      ETX
* ************************************************************************/

int        download ()
{
     int   data_buff [32];

/* set offset command */

     data_buff [0] = 2;          /* byte count */
     data_buff [1] = 12;         /* download offset command */
     data_buff [2] = 20;         /* new offset data  = 20 */

/* set motion limits */

     data_buff [3] = 5;          /* byte count */
     data_buff [4] = 65;         /* download motion limits command */
     data_buff [5] = 00;         /* new motion low value = 0005 */
     data_buff [6] = 05;
     data_buff [7] = 1;          /* new motion high value = 1234 */
     data_buff [8] = 234;

/* program channel 1 */

     data_buff [9] = 4;          /* byte count */
     data_buff [10] = 56;        /* download setpoints command */
     data_buff [11] = 1;         /* channel number */
     data_buff [12] = 0;         /* on setpoint = 0 */
     data_buff [13] = 10;        /* off setpoint = 10 */

/* program channel 2 */

     data_buff [14] = 4;         /* byte count */
     data_buff [15] = 56;        /* download setpoints command */
     data_buff [16] = 2;         /* channel number */
     data_buff [17] = 10;        /* on setpoint = 10 */
     data_buff [18] = 20;        /* off setpoint = 20 */

/* program channel 3 */

     data_buff [19] = 4;         /* byte count */
     data_buff [20] = 56;        /* download setpoints command */
     data_buff [21] = 3;         /* channel number */
```

```
        data_buff [22] = 20;        /* on setpoint = 20 */
        data_buff [23] = 30;        /* off setpoint = 30 */

        send (DOWN_CMD, 0, 'w', 24, data_buff);
        rx ();
        return (TRUE);
    }

    /*****************************************************************************
    *
    * send (command, status, r/w, count, buffer)
    *     Forms the transmit string & then outputs it to the UART.
    *
    *     command
    *     status
    *     r/w flag (for unprot. block read/write)
    *     length of the data field
    *     address of the data buffer
    *
    *     Outputs:
    *
    *     DLE, STX, DST, SRC, UNPROT BLK R/W CMD, STS,
    *     TNS(word), command(word), data..., DLE, ETX, CRC(word).
    * ***************************************************************************/

    void  send (cmd, sts, rw, count, buffer)
    char  cmd, sts, rw;
    int   count, *buffer;
    {
        char  *tx_ptr, *put();
        int   cmd_len, i, number, tens, huns;
        unsigned  get_crc();

    /* Reset variables */

        dsply_ofs = input_ofs = 0;
        tx_crc = 0;
        tx_ptr = tx_buffer;

    /* All messages start with DLE/STX */

        *tx_ptr++ = DLE;
        *tx_ptr++ = STX;
        cmd_len = 2;

    /* Insert the destination & source station numbers */
```

```
        tx_ptr = put (tx_ptr, dest_station, &cmd_len);
        tx_ptr = put (tx_ptr, src_station, &cmd_len);

/* Insert the command type (read or write) */

        if (rw = = 'r')
            tx_ptr = put (tx_ptr, UNPROT_BLK_RD_CMD, &cmd_len);
        else
            tx_ptr = put (tx_ptr, UNPROT_BLK_WR_CMD, &cmd_len);

/* Insert the status */

        tx_ptr = put (tx_ptr, sts, &cmd_len);

/* Insert the current transaction number (ls : ms) */

        tx_ptr = put (tx_ptr, trans & 0xff, &cmd_len);
        tx_ptr = put (tx_ptr, trans > 8, &cmd_len);

/* Copy the command to the address field (ls:ms) */

        tx_ptr = put (tx_ptr, cmd, &cmd_len);
        tx_ptr = put (tx_ptr, 0, &cmd_len);

/* If this is a read command, insert the number of bytes to be read.  If this  is a write command, insert the data. */

        if (rw = = 'r')
            tx_ptr = put (tx_ptr, *buffer + +, &cmd_len);
        else {
            while (count) {

/* Convert binary to BCD/hex!  Note that the largest A-B 'word' is 999 */

                number = *buffer;
                tens = (number % 100 / 10) * 16 + number % 100 % 10;
                huns = number / 100;

/* Insert data into tx string */

                tx_ptr = put (tx_ptr, tens, &cmd_len);
                tx_ptr = put (tx_ptr, huns, &cmd_len);
                count-,
                buffer + +;
            }
        }
```

```
/* End of data, append DLE/ETX and the CRC */
      *tx_ptr++ = DLE;
      *tx_ptr++ = ETX;
      tx_crc = get_crc (ETX, tx_crc);
      *tx_ptr++ = tx_crc & 0xff;
      *tx_ptr = tx_crc > 8;
      cmd_len += 4;
      trans++;

/* transmit the message */

      for (i = 0; i  cmd_len; i++)
            tx_char (tx_buffer[i]);
      return;
}

/*****************************************************************************
*
* put (destination, data, length)
*
*     Inserts data into the destination string. The CRC is updated. If the data == DLE,
*     an extra DLE is inserted. The length of the string is also updated, and the new
*     pointer is returned.
*
*****************************************************************************/

char *put (ptr, data, len)
char *ptr, data, *len;
{
      unsigned get_crc ();

      *ptr++ = data;
      (*len)++;
      tx_crc = get_crc (data, tx_crc);

/* Note DLE insertion (extra DLE is NOT part of the CRC) */

      if (data == DLE)
            *ptr++ = DLE, (*len)++;
      return (ptr);
}

/*****************************************************************************
*
* get_crc (new_data, current_crc)
*
*     Computes and returns a new CRC.
```

```
*
*     new_data = new byte to be added to the CRC
*     current_crc = current CRC value
*
*****************************************************************************/

unsigned  get_crc (new_data, current_crc)
char      new_data;
unsigned  current_crc;
{
     int   i;

     current_crc ^ = new_data;

     for (i = 0; i  8; i+ +) {
          if (current_crc & 1)
               current_crc = (current_crc > 1) ^ 0xa001;
          else
               current_crc = current_crc > 1;
     }
     return (current_crc);
}

/*****************************************************************************
*
* RX ()
*     Collects the received data.  It will monitor data until the end of a valid message
*     occurs.
*
*****************************************************************************/

char rx ()
{
     char rx_char, dle_etx, dle_stx, crc_seq, dle_flag, eom_flag;
     int   crc_hi, crc_lo, rx_crc;
     unsigned  get_crc ();

/* Reset all flags and the CRC */

     eom_flag = dle_flag = dle_etx = dle_stx = crc_seq = FALSE;
     rx_crc = 0;

/* Fetch data from the receive buffer until it is empty */

     while (dsply_ofs ! = input_ofs) {
          rx_char = rx_buffer [dsply_ofs];
```

```
/* If a DLE/ETX sequence has been found, we're at the end of the message.  Verify the
CRC.  If the CRCs don't match, ring the bell, otherwise return a DLE/ACK to the sender. */

        if (dle_etx) {
            if (crc_seq) {
                crc_hi = rx_char;
                rx_crc = (crc_hi < 8) | crc_lo;
                if (rx_crc != calc_rx_crc)
                    putchar ('\7');
                crc_seq = dle_etx = dle_stx = FALSE;
                eom_flag = TRUE;
                tx_char (DLE), tx_char (ACK);
            }
            else
                crc_lo = rx_char, crc_seq = TRUE;
        }
        switch (rx_char) {

/* DLE received.  Check for DLE/DLE sequence. */

            case DLE:
                if (dle_flag) {
                    dle_flag = FALSE;

/* If DLE/DLE occurs after message start, put just 1 DLE in the rx buffer */

                if (dle_stx) {
                    rx_len++;
                    calc_rx_crc = get_crc (rx_char, calc_rx_crc);    } } else
                    dle_flag = TRUE;
                break;

/* ETX received.  If DLE was just received, then we're at the end of the message. */

            case ETX:

/* If ETX occurs within message, add it to the message buffer */

                if (dle_stx) {
                    calc_rx_crc = get_crc (rx_char, calc_rx_crc);
                    rx_len++;
                }
                if (dle_flag) {
                    dle_etx = TRUE;
                    dle_flag = dle_stx = FALSE;
                }
                break;
```

```
/* STX received.  If DLE was just received, then we're at the start of a message */

                case STX:
                    if (dle_flag) {
                        dle_stx = TRUE;
                        dle_flag = FALSE;
                        msg_ofs = dsply_ofs;
                        calc_rx_crc = rx_len = 0;
                    }
                    else if (dle_stx) {
                        rx_len++;
                        calc_rx_crc = get_crc (rx_char, calc_rx_crc);    }    break;

/* If DLE/NAK receive, an error has occurred, ring the bell */

                case NAK:
                    if (dle_flag) {
                        putchar ('\7');
                        eom_flag = NAK;
                    }
                    else if (dle_stx) {
                        rx_len++;
                        calc_rx_crc = get_crc (rx_char, calc_rx_crc);
                    }
                    break;

/* Any other chars received after start of message are inserted into the message buffer */

                default:
                    dle_flag = FALSE;
                    if (dle_stx) {
                        rx_len++;
                        calc_rx_crc = get_crc (rx_char, calc_rx_crc);
                    }
                    break;
                }
            dsply_ofs = ++dsply_ofs % 256;
            if (eom_flag)
                break;
        }
        return (eom_flag);
}

/*****************************************************************************
*    NEXT () parses the rx buffer.  It skips all DLE DLE sequences and returns the next
*    char from the buffer.
```

```
*
*****************************************************************************/

int         next ()
{
    char data;

/* If the next char is DLE, skip 1 byte since each DLE in the data field is automatically fol-
lowed by a second DLE */

    if (rx_buffer [msg_ofs] = = DLE)
        msg_ofs = (msg_ofs + 1) % 256;
    data = rx_buffer [msg_ofs];
    msg_ofs = (msg_ofs + 1) % 256;
    return (data);
}
```