T100MD+

(& T100MX+)

Super Programmable Controllers

Triangle Research
International, Inc.

User's
Manual

Copyright Notice and Disclaimer

Conditions of Sale and Product Warranty

Triangle Research International Inc. (TRi) and the Buyer agree to the following terms and conditions of Sale and Purchase:

1. The T100MD+and T100MX+ Programmable Controllers are guaranteed against defects in materials or workmanship for a period of one year from the date of registered purchase. Any unit which is found to be defective will, at the discretion of TRi, be repaired or replaced.

2. TRi will not be responsible for the repair or replacement of any unit damaged by user modification, negligence, abuse, improper installation, or mishandling.

3. TRi is not responsible to the Buyer for any loss or claim of special or consequential damages arising from the use of the product. The product must **NOT** be used in applications where failure of the product could lead to physical harm or loss of human life. Buyer is responsible to conduct their own tests to meet the safety regulation of their respective industry.

4. Products distributed, but not manufactured by TRi, carry the full original manufacturers warranty. Such products include, but are not limited to: power supplies, sensors, I/O modules and battery backed RAM.

5. TRi reserves the right to alter any feature or specification at any time.

Notes to Buyer: If you disagree with any of the above terms or conditions you should promptly return the unit to the manufacturer or distributor within 30 days from date of purchase for a full refund.

Table of Contents

Chapter 1: Special I/Os and Analog Interfacing Page

1.1 Introduction 1-1
1.2 Special Digital I/Os 1-1
1.3 Stepper Motors Controller Outputs 1-2
1.4 PWM Outputs 1-4
1.5 Using High Speed Counter Inputs with Rotary Encoder 1-7
1.6 Using Interrupt Inputs 1-9
1.7 Using Pulse Measurement Inputs 1-10
1.8 Analog I/Os 1-10
1.9 Serial Communication Ports 1-13
- 1.9.1 COMM1: RS232C with Female DB9 1-13
- 1.9,2 COMM3: Two-wire RS485 Port (& Applications) 1-14
- 1.9.3 Changing Baud Rate and Communication Format: Use of SETBAUD Statement 1-16
- 1.9.4 Support of Multiple Communication Protocols 1-18
- 1.9.5 Accessing the COMM Ports from within TBASIC 1-19
- 1.9.6 Using Modem to Remotely Program/Monitor PLC 1-21
- 1.9.7 Constructing a 2nd Multi-drop Network 1-22

1.10 DIP Switches 1-22
1.11 CPU Status Indicators 1-23
1.12 Internal Relays, Timers & Counters, etc. 1-24

Chapter 2: Operating Procedure

2.1 Programming 2-1
2.2 Simulation 2-1
2.3 Transferring Program to PLC 2-1
2.4 Errors and Problems 2-2
2.5 On-Line Monitoring & Control 2-3
- 2.5.1 Monitoring PLC's I/O Logic States 2-3
- 2.5.2 Viewing and Modifying PLC's Internal Variables 2-3
- 2.5.3 Force Setting/Resetting I/O Bits 2-4
- 2.5.4 Suspending PLC's Ladder Program 2-4

2.6 Ladder Monitoring 2-4
2.7 Get PLC's Hardware Info 2-5
2.8 Set PLC's Real Time Clock 2-5
2.9 Trouble-Shooting Communication Errors 2-6

Chapter 3: Host Communication

3.1 Point-to-point Communication 3-2
3.2 Multi-Point Communication System 3-3
3.2.1 RS485 Network Interface Hardware 3-4
3.2.2 Protection of RS485 Interface 3-5
3.2.3 Single Master RS485 Networking Fundamentals 3-6
3.2.4 Multi-Masters RS485 Networking Fundamentals 3-7
3.2.5 Command/Response Block Format (Multipoint) 3-10
3.2.6 Communication Procedure 3-11
3.3 Shoud You Use Point-to-Point or Multi-point Protocol? 3-11
3.4 Trouble-Shooting RS485 Network 3-13

Chapter 4: Command / Response Format

4.1 Device ID Read 4-1
4.2 Device ID Write 4-1
4.3 Read Input Channels 4-1
4.4 Read Output Channels 4-2
4.5 Read Relay Channels 4-3
4.6 Read Timer Contacts 4-3
4.7 Read Counter Contacts 4-4
4.8 Read Timer Present Value (P.V.) 4-4
4.9 Read Timer Set Value (S.V.) 4-5
4.10 Read Counter Present Values (P.V.) 4-5
4.11 Read Counter Set Value (S.V.) 4-6
4.12 Read Variable - Integers (A to Z) 4-6
4.13 Read Variable - Strings (A$ to Z$) 4-6
4.14 Read Variable - Data Memory (DM[1] to DM[4000]) 4-6
4.15 Read Variable - System Variables 4-7
4.16 Read Variable - High Speed Counter HSCPV[ ] 4-8
4.17 Write Inputs 4-8
4.18 Write Outputs 4-8
4.19 Write Relays 4-9
4.20 Write Timer-Contacts 4-9
4.21 Write Counter-contacts 4-9
4.22 Write Timer Present Value (P.V.) 4-9
4.23 Write Timer Set Value (S.V.) 4-10
4.24 Write Counter Present Value (P.V.) 4-10
4.25 Write Counter Set Value (S.V.) 4-10
4.26 Write Variable - Integers (A to Z) 4-11
4.27 Write Variable - Strings (A$ to Z$) 4-11
4.28 Write Variable - Data Memories (DM[1] to DM[4000]) 4-11

4.29 Write Variable - System Variables 4-12
4.30 Write Variable - High Speed Counter HSCPV[ ] 4-12
4.31 Update Real Time Clock Module 4-12
4.32 Halting the PLC 4-13
4.33 Resume PLC Operation 4-13
4.34 Read Analog Input 4-14
4.35 Read EEPROM Integer Data 4-15
4.36 Read EEPROM String Data 4-15
4.37 Write Analog Output 4-15
4.38 Write EEPROM Integer Data 4-16
4.39 Write EEPROM String Data 4-16
4.40 Force Set/Clear Single I/O Bit 4-16
4.41 Testing of Host Link Commands 4-17
4.42 Visual Basic Sample Program 4-17
4.43 Inter-PLC Networking Using NETCMD$ command 4-18
4.44 Inter-PLC Networking Using MODBUS Protocols 4-19
4.45 Using OMRON's Host Link Commands
I. Read IR Registers 4-20
II. Write IR Registers 4-21
I. Read Multiple DMs 4-21
I. Write Multiple DMs 4-22

Chapter 5: MODBUS / OMRON Protocols Support

5.1 MODBUS ASCII Protocol Support 5-1
5.2 MODBUS RTU Protocol Support 5-4
5.3 OMRON Host Link Command Support 5-5
5.4 Appliation Example: Interfacing to SCADA Software 5-5
5.5 Using The T100M+ PLC as MODBUS Master 5-6

Chapter 1 Special I/Os and Analog Interfacing

1.1 Introduction

A Standard T100MD+ PLC features the following:

1) 4 to 8 channels of 10-bit Analog Inputs. (4 on T100MD1616+)
2) 1 to 2 channels of 8-bit Analog outputs.
3) 2-channel programmable Motion Controllers for controlling stepper motors up to 20,000 pulses-per-second.
4) 2-channel Pulse Width Modulated (PWM) outputs.
5) 2-channel 32-bit High Speed Counters (HSC) counts up to 10,000 Hz.
6) 4-channel Interrupt Inputs.
7) 2-channel pulse measurement inputs capable of measuring frequency and pulse-width of incoming pulses up to 10,000 Hz.
8) Real time Clock/Calendar for programming scheduled ON/OFF events.
9) 6016 Words (16-bit) of EEPROM Program Memory, expandable to 8190 Words with optional IC - M2018P.
10) 1700 Words (16-bit) of programmable EEPROM for user's data, expandable to 7750 Words with optional IC – M2018P.
11) Built-in 16 channels of PID-computation engines let T100M+ PLCs directly provide PID type digital control for process automation.
12) One Independent RS232 port for connection to a host PC for programming or monitoring.
13) One independent RS485 port for networking or for connecting to external peripherals such as LCD display and RS485-based analog I/O cards, etc.
14) Industry Standard Protocols: Both RS232 and RS485 serial port simultaneously support multiple communication protocols, as follow:
    i) Native ASCII based Host Link Commands.
    ii) MODBUS RTU protocols
    iii) MODBUS ASCII Protocols
    iv) OMRON C20H Host Link Commands.
15) Watch-Dog Timer (WDT) which resets the PLC if the CPU malfunctions due to hardware or software error. A system reset by WDT can be determined by the STATUS(1) command.

1.2 Special Digital I/Os

Four of the first 8 ON/OFF inputs of the T100M+ PLC can be configured as "special inputs" such as High Speed Counters, Interrupts and Pulse Measurement. Some of the first 8 outputs can also be configured as PWM and the stepper controller pulse-outputs. If these special I/Os are not used, then they can be used as ordinary ON/OFF type I/O in the ladder diagram. Note that if two special functions share the same I/O then only one of them can be active at any one time. The location of these special I/O are tabulated as follows:

Special Inputs

Input #	High Speed Counter	Interrupt	Pulse Measurement
1	-	-	-
2	-	-	-
3	Ch #1: Phase A	Ch #1	Ch #1
4	Ch #1: Phase B	Ch #2	Ch #2
5	Ch #2: Phase A	Ch #3	-
6	Ch #2: Phase B	Ch #4	-
7	-	-	-
8	-	-	-

Note: A pin defined as a special input cannot simultaneously act as another special input. E.g. Pin 3 cannot be used as high speed counter and at the same time serves as a pulse measuring pin.

Special Outputs

Output #	Stepper pulse output	PWM output
1	Direction for Ch #1	-
2	Direction for Ch #2	-
3		-
4		-
5	Ch #1	-
6	Ch #2	-
7	-	Ch #1
8	-	Ch #2

These special I/O therefore share the same electrical specifications as the ON/OFF type I/O, which have already been described in the Installation Guide.

1.3 Stepper Motors Controller Outputs

Technical Specifications:

No. of Channels	2
Max. Pulse Rate (pps)	20000 (single channel running) 10000 (two channels running)
Maximum Load Current	1A @24V DC
Velocity Profile (Defined by STEPSPEED)	Trapezoidal -accelerate from 1/8 max pps to max pps. -decelerate from max pps to 1/8 max pps)
Maximum number of steps	$2 \sim 2^{31}$ (= 2.1×10^{9})
TBASIC commands	STEPSPEED, STEPMOVEABS, STEPCOUNTABS( ), STEPMOVE, STEPSTOP, STEPCOUNT( )

It is essential to understand the difference between a stepper motor “Controller” and a stepper motor “Driver”. A stepper motor “Driver” comprises the power electronics circuitry that provides the voltage, current and phase rotation to the stepper motor coils.

The T100M+'s built-in Stepper-Motor Controller, on the other hand, only generates the required number of "pulses" and sets the direction signal according to the defined acceleration and maximum pulsing rate specified by "STEPSPEED" and "STEPMOVE" commands. You cannot directly connect the "pulses" to the stepper motor. You will need a stepper motor "driver" which you can buy from the motor vendor. Depending on the power output, the number of phases of the stepper motor, and whether you need micro-stepping, the driver can vary in size and cost. Most stepper motor drivers have opto-isolated inputs which accept a direction signal and stepping-pulse signal from the "Stepper Motor Controller". In this case the T100M+ is the "Stepper Motor Controller" which will supply the required pulse and direction-select signals to the driver.

Note that the digital output #1 and #2 automatically become the direction-select signals for Stepper controller #1 and #2, respectively when the stepper controllers are being used. The direction pin is turned ON when the motor moves in the negative direction and turned OFF when the stepper motor moves in the positive direction. The STEPMOVEABS command makes it extremely simple to position the motor at an absolute location, while the STEPMOVE command let you implement incremental move in either directions for each channel.

Interfacing to 5V Stepper Motor Driver Inputs

Some stepper motor drivers accept only 5V signals from the stepper motor controller. In such case you need to determine whether the driver's inputs are opto-isolated. If they are then you can simply connect a 2.2K current limiting resistor in series to the path from the PLC's output to the driver's inputs, as shown in the following diagram:

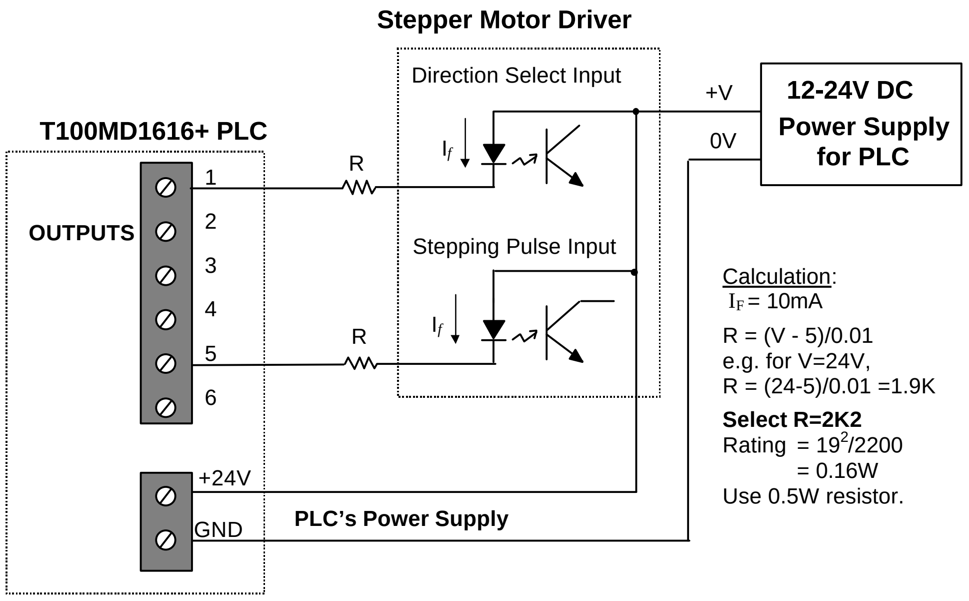


Figure 1.1

However, if the stepper motor driver input is only 5V CMOS level and non opto-isolated, then you need to convert the 12-24V outputs to 5V. This can be achieved using low cost transistor such as a 2N4403. A better way is to use an opto-isolator with logic level output as shown in Figure 1.2. This provides a galvanic isolation between the PLC and the stepper motor driver.

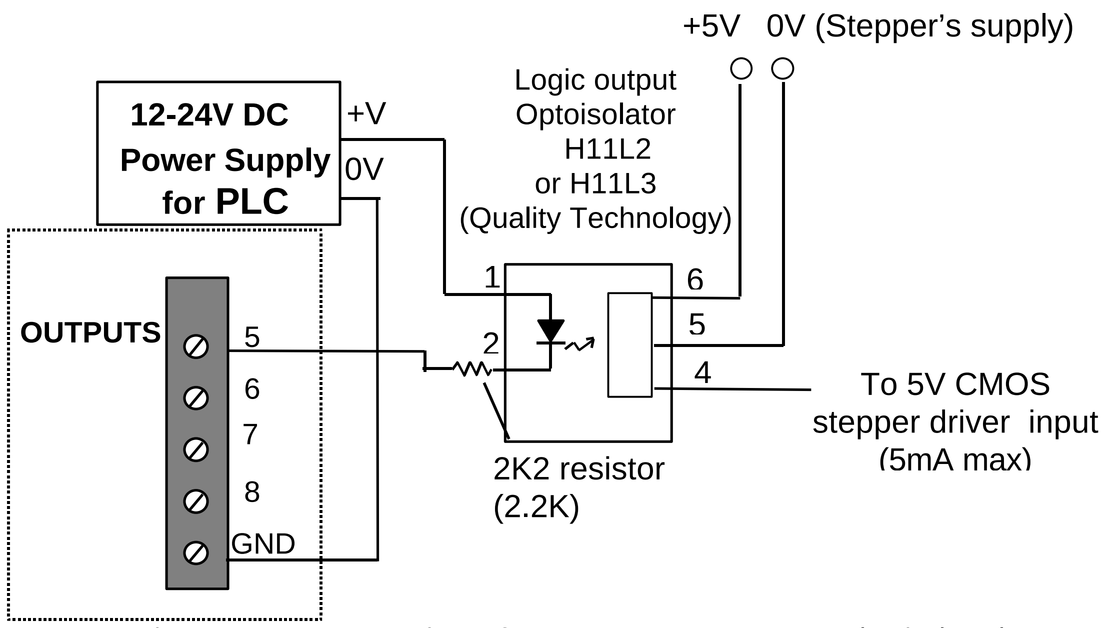
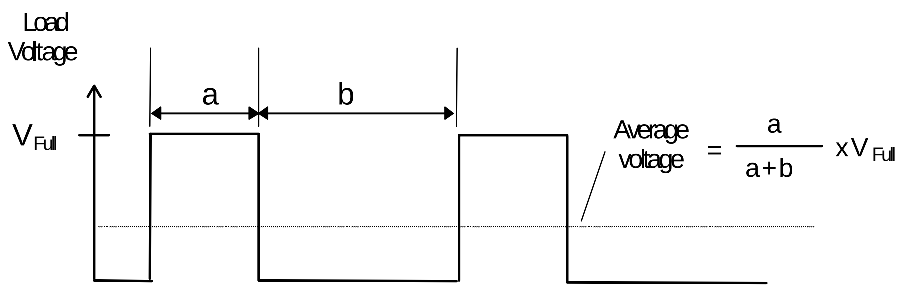

+5V 0V (Stepper's supply)
12-24V DC Power Supply for PLC
+V
0V
Logic output Optoisolator H11L2 or H11L3 (Quality Technology)
OUTPUTS
5
6
7
8
GND
1
2
6
5
4
2K2 resistor (2.2K)
To 5V CMOS stepper driver input (5mA max)

Figure 1.2 Conversion of T100MD+ outputs to 5V logic level

1.4 PWM Outputs

Pulse-Width Modulation (PWM) is a highly efficient and convenient way of controlling output voltage to devices with large time constants, such as controlling the speed of a DC motor, the power to a heating element or the position of a proportional valve.

PWM works by first turning the output to full voltage for a short while and then shutting it off for another short while and then turning it on again and so on in accurate time intervals. This can be illustrated in the following diagram:



The average voltage seen by the load is determined by the "duty cycle" of the PWM wave form. The duty cycle is defined as follow:

$$\text{Duty Cycle} = \frac{a}{a+b} \times 100\%$$

Period = (a + b)
Frequency = 1/period Hz

Average voltage = % duty cycle multiplied by the full load voltage V_{Full}. Since the voltage applied to the load is either “Fully ON” or “Fully OFF”, it is highly efficient because the switching transistors are working in their saturated and cut-off region and dissipate very little power when it is fully turned ON or OFF.

Technical Specifications:

No. of Channels	2
Duty Cycle range	0.00 to 100.00
Actual Resolution	0.4%
Available Frequencies (Hz)	16, 32, 63, 250, 500, 2000, 8000 and 32000 Hz
Relevant TBASIC commands	setPWM

The frequency of the PWM waveform can also be varied. T100M+ supports the following frequencies: 16, 32, 63, 250, 500, 2000, 8000 and 32000 Hz. Usually it is better to select as high a frequency as possible because the resulting effect is smoother for higher frequencies. However, some systems may not respond properly if the PWM frequency is too high, in such cases a lower frequency should be selected.

The TBASIC **setPWM** statement controls the frequency and duty-cycle settings of the PWM channel. The T100M+ PLC features two channels of PWM on its outputs #7 (PWM ch #1) and #8 (PWM ch#2). Since these two outputs are high voltage, high current outputs (24V, 1A or 10A on some PLC models) they can be used to directly control the speed of a small DC motor. They can also directly drive proportional (variable position) valves whose opening is dependent on the applied voltage.

Increasing Output Drive Current (Non Opto-Isolated)

The T100MD888+ has two channel of 10A, current sinking PWM outputs which should be sufficient for many applications. However, the two channel PWM outputs on T100MD1616+ as well as the T100MX-3224R+ and T100MX-4832+ are all limited to 1A current each. If you need to control power devices that demand more than the 1A current on these PLCs you can use the following circuit to amplify the drive current:

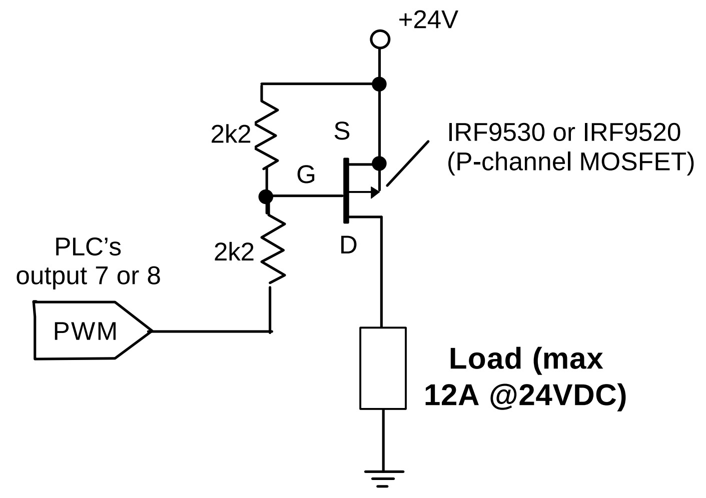


Figure 1.3

The MOSFET driver IRF9530 can drive up to 12A of currents (this is the actual circuit designed into the two 10A PWM outputs on T100MD2424+). However, note that the output will be converted into a "source" (PNP) type, The above circuit is also **not** opto-isolated and hence you have to take the usual precautions of preventing the large current load demand from interfering with the power supply voltage of the PLC.

Increasing Output Drive Current (Opto-Isolated)

The advantage of using PWM is that you can easily amplify the drive current to a larger load such as a larger permanent magnet DC motor by using a power transistor or power MOSFET to boost the current switching capability. If the load is of different voltages and the load current is high, you should use an opto-isolator to isolate the PLC from the load, as in Figure 1.4

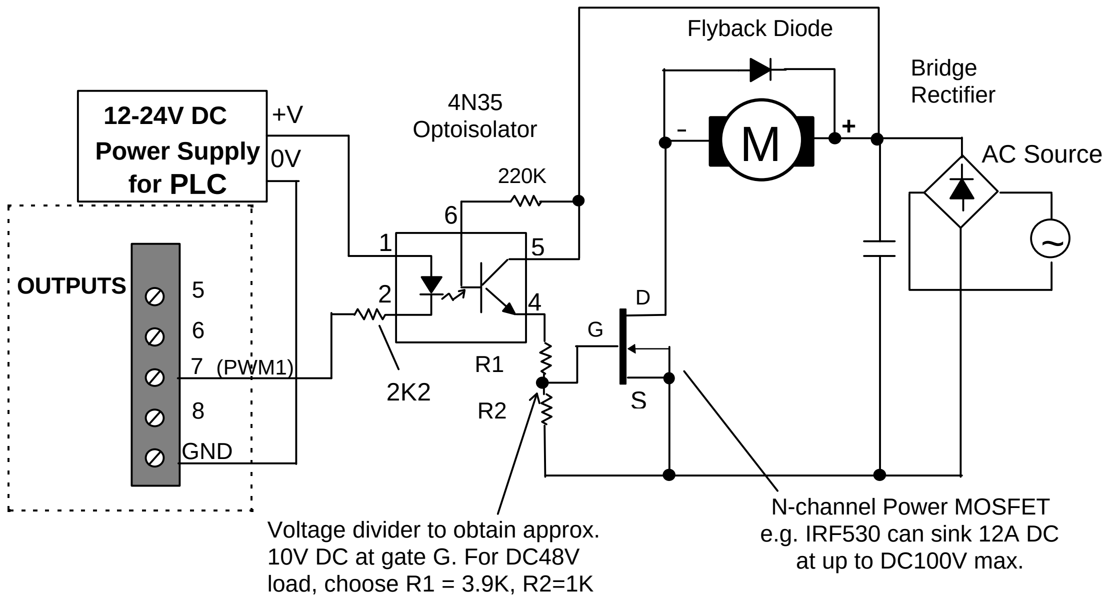


Figure 1.4 PWM Speed Control of a large DC Motor.

Note:

a) The opto-isolator must be able to operate at a frequency matching that of the PWM frequency, otherwise the resulting output waveform will be distorted and effective speed control cannot be attained.

b) The simple PWM speed control scheme described above is open-loop type and does not regulate the speed with respect to changing load torque. Closed-loop speed control is attainable if a tachometer (either digital or analog) is used which feeds back to the CPU the actual speed. Based on the error between the set point speed and the actual speed, the software can then adjust the PWM duty cycle accordingly to offset speed variation caused by the varying load torque. A PID function may also be invoked to provide sophisticated PID type of speed control.

c) The T100M+ PWM can be used to control the speed of small motors. For larger motors, industrial- strength variable-speed drivers should be used instead.

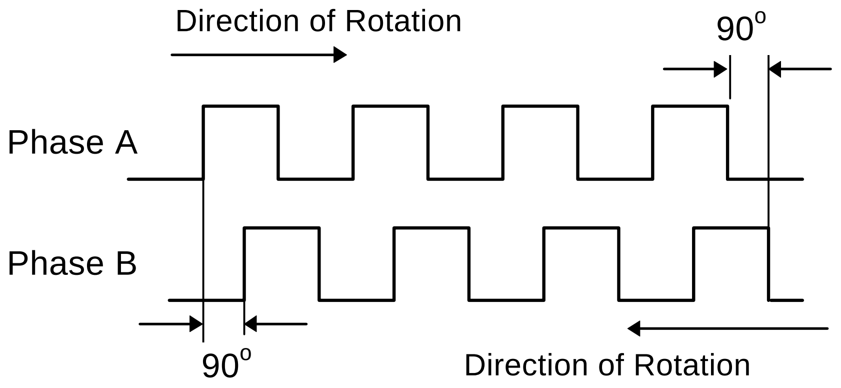
1.5 Using High Speed Counter Inputs with A Rotary Encoder

Technical Specifications:

No. of Channels	2
Maximum acceptable pulse rate	10KHz for T100MD 4KHz for T100MX
Quadrature signal decoding	Automatic
Relevant TBASIC Commands	HSCDEF, HSCOFF, HSCPV[ ]

Descriptions:

Input #3, 4 and Inputs #5, 6 form two channels of high speed counter inputs which can interface directly to a rotary encoder that produces "quadrature" outputs. A quadrature encoder produces two pulse trains at 90^o phase shift from each other as follows:



When the encoder shaft rotates in one direction, phase A leads phase B by 90 degrees. When the shaft rotates in the opposite direction, phase B will lead phase A by 90 degrees. The quadrature signals therefore provide an indication of the direction of rotation.

T100M+ handles the quadrature signals as follows: if the pulse train arriving at input #3 leads the pulse train at input #4, the High Speed Counter (HSC)

#1 increments on every pulse. If the pulse train arriving at input #3 lags the pulse trains at input #4, then the HSC #1 decrements. Note that if input #4 is OFF, then pulse trains arriving at input #3 is considered to lead the input #4 and HSC #1 will be incremented. Likewise if input #3 is OFF, then pulse trains arriving at input #4 will decrement HSC #1.

Input #5 and #6 form the inputs for High Speed Counter channel #2 and they operate in the same way as Input#3 and #4 for HSC#1 described above.

The fact that the T100M+ PLC automatically takes care of the direction of rotation of the quadrature encoder greatly simplifies the programmer's task of handling high-speed encoder feedback. The HSCdef statement can be used to define a CusFn to be executed when the HSC reaches a certain pre-defined value. Within this CusFn you can define the action to be taken and define the next CusFn to be executed when the HSC reaches another value.

Enhanced Quadrature Decoding

The default method in which the PLC handles quadrature signal as described above is somewhat simplistic. It does not take into consideration the "jiggling" effect that occurs when the encoder is positioned at the transition edge of a phase. Mechanical vibration could cause multiple counts if the rotor shaft "jiggle" at the transition edge of the phase, resulting in multiple triggering of the counter. This simplistic implementation, however, does have the advantage that the HSC can also be used for single-phase high-speed counting.

For M-series PLC with firmware revision of r39 and above, an enhanced quadrature decoding routine is provided which will lock out multiple counting by examining the co-relationship between the two phases. You can configure the M-series PLC to use the enhanced quadrature counting by using the SETSYSTEM command, as follows:

```
SETSYTEM 4, n.
```

n=0 == simple decoding for both HSC1 & HSC2 (default).
n=1 for enhanced quadrature decoding in HSC1 only.
n=2 for enhanced quadrature decoding in HSC2 only
n=3 for enhanced quadrature decoding in both HSC1 & HSC2.

Interfacing to 5V type Quadrature Encoder

If you have a choice, you should select an encoder that can produce 12V or 24V output pulses so that they can drive the inputs #3,4,5 or 6 directly. If you have 5V type of encoder only, then you need to add a transistor driver to interface to the PLC's inputs. The simplest way is to use an IC driver ULN2003 connected as shown in Figure 1.5.

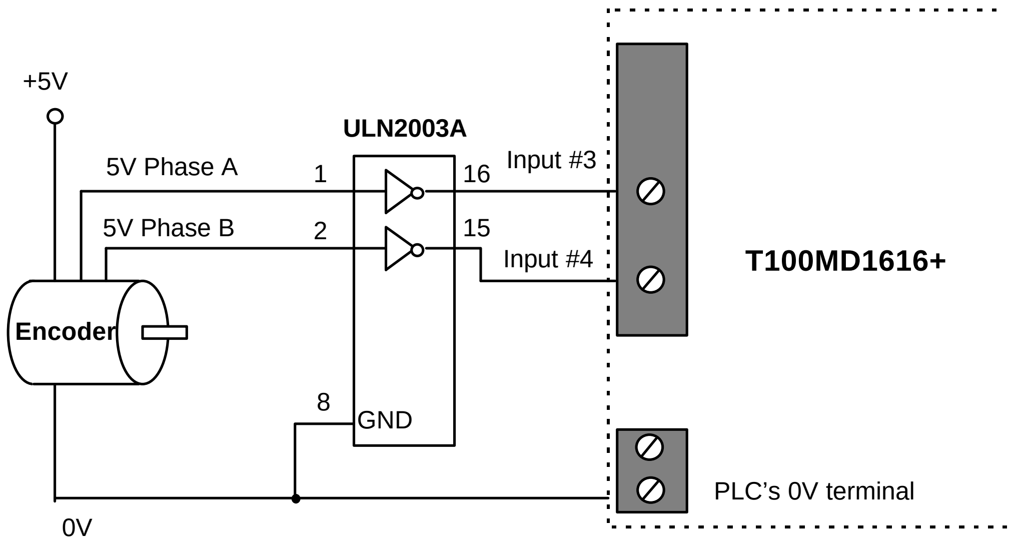


Figure 1.5 Interfacing 5V type Rotary Encoder

1.6 Using Interrupt Inputs

During normal PLC ladder program execution, the CPU scans the entire ladder program starting from the first element, progressively solving the logic equation at each circuit until it reaches the last element. After which it will update the physical Inputs and Outputs (I/O) at the end of the scan. Hence the location of a logic element within the ladder diagram is important because of this sequential nature of the program execution.

When scanning the ladder program, the CPU uses some internal memory variables to represent the logic states of the inputs obtained during the last I/O refresh cycle. Likewise, any changes to the logic state of the outputs are temporarily stored in the output memory variable (not the actual output pin) and will only be updated to the physical output during the next I/O refresh.

You may see that any changes to the input logic state will only be noticed by the CPU when it has completed the current scan and starts to refresh its input variables. The input logic state must also persist for at least one scan time to be recognized by the CPU. In some situations this may not be desirable because any response to the event will take at least one scan time or more.

An interrupt input, on the other hand, may occur randomly and the CPU will have to immediately suspend whatever it is doing and start "servicing" the interrupt. Hence the CPU responds much faster to an interrupt input. In addition, interrupts are "edge-triggered", meaning that the interrupt condition occurs when the input either changes from ON to OFF or from OFF to ON. Consequently, the input logic state need not persist for longer than the logic scan time for it to be recognized by the CPU.

Any one or all of inputs #3 to #6 can be used as interrupt inputs when defined by the INTDEF statement. The Interrupt inputs may also be defined as either rising-edge triggered (input goes from OFF to ON) or falling-edge triggered (input goes from ON to OFF). When the defined edges occur, the defined CusFn will be immediately executed irrespective of the current state of execution of the ladder program.

1.7 Using Pulse Measurement Inputs

T100M+ PLC provides a very straightforward means to measure the pulse width or frequency of a square-wave pulse-train arriving at its Pulse Measurement (PM) inputs #3 or #4.

To use the input to measure pulse width or frequency, execute the PMON statement to configure the relevant input to become a pulse measurement input. Thereafter the pulse width (in μs) or the pulse frequency (in Hz) can be easily obtained from the PULSEWIDTH(n) or PULSEFREQUENCY(n) function.

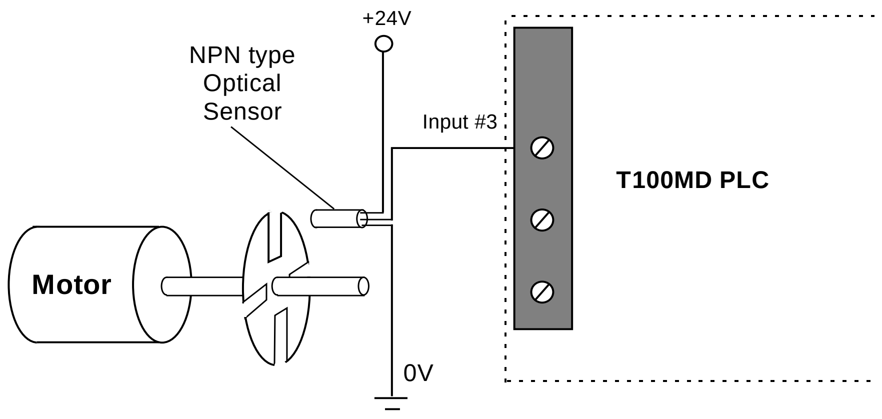


Figure 1.6 Setting Up a Simple Tachometer or Encoder

<u>Applications</u>

1) One useful application of the PM capability is to measure the speed of rotation of a motor. A simple optical sensor, coupled with a rotating disk with slots fitted to the shaft of a motor (see Figure 1.6) can be fabricated economically. When the motor turns, the sensor will generate a series of pulses. The frequency of this pulse train relates directly to the rotational speed of the motor and can be used to provide precise speed control. Note that the above setup can also double as a low cost position-feedback encoder when used with the high speed counter, since the number of pulses counted can be used to determine the displacement.

2) Some transducers incorporate Voltage-Controlled-Oscillator (VCO) type of outputs that represent the measured quantities in terms of varying

frequency of the output waveform. Such transducers may be used conveniently by T100M+ PLCs using the pulse measurement capability. However, the frequency of such signal must be below 10,000 Hz.

3) For an application that requires measurement of the frequency of a high-speed counter, you will need to feed the pulse inputs into both input #3 and Input #5. In this case HSC #2 is used together with PM #1 to count the input pulses as well as measure its frequency. This is because an input pin that has been defined as High Speed Counter cannot simultaneously be defined as Pulse measurement pin. If you execute both the HSCDEF 1 and PMON 1 in the same program, the last executed command will take precedence.

1.8 Analog I/Os

<u>A/D Electrical Characteristics</u>

No. of A/D channel	: 4 to 8 depending on the model.
Resolution:	:10-bit
Built-in Sample & Hold	: Yes.
Conversion Time	:10µs per channel.

<u>D/A Electrical Characteristics</u>

No. of A/D channel	: 1 or 2, depending on the model.
Resolution:	: 8-bit
Conversion Time	: 10µs per channel.

Notes:

1) Although the A/D converters' actual resolutions are only 10-bit and the D/A converters' actual resolutions are only 8-bit, T100M+ PLCs normalize all the analog data to 12-bit numbers. Hence you will find that ADC(n) function returns the value as 0,4,8,12,16....4092 (not 4095 since the least significant two bits are always zero). Similarly, the D/A converters shift the 12-bit normalized value applied to it by four bits to the right to convert it into an 8-bit quantity before applying the value to the DAC hardware. Hence the full scale value of D/A occur when the actual digital code = 255. When normalized to 12-bit quantities = 255 x 16 = 4080.

   The reason for normalizing all analog data to 12-bit is that in future if new models of PLCs with higher resolution A/D or D/A converters are introduced, the user's PLC program need not be modified since there will not be needs to change the computational expression when all data are already treated as 12-bit full-scale.

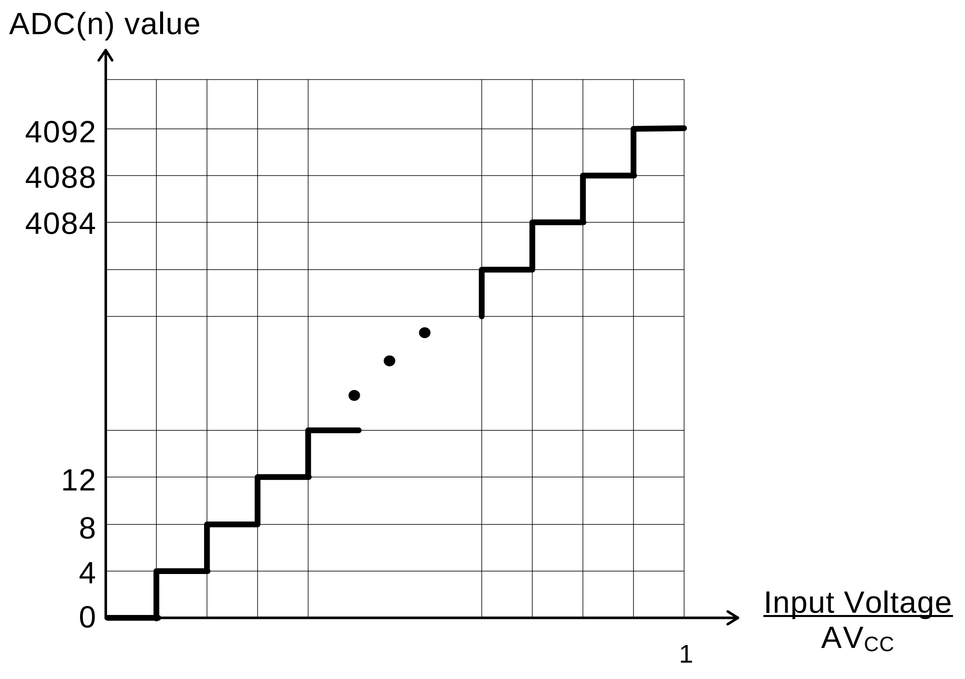


Figure 1.7 Transfer Function for 10-bit ADC.

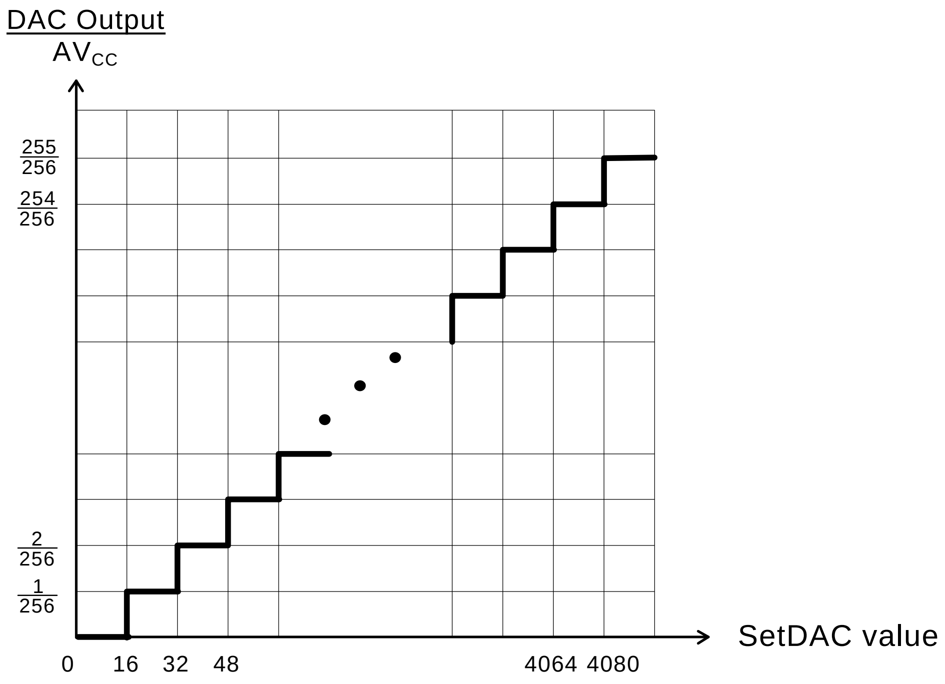


Figure 1.8 Transfer Function for 8-bit DAC.

Interfacing to Industrial Analog Sensors

Real world sensors such as a J- or K-type thermo-couple temperature probe produce only micro volts of signal voltage in response to temperature changes. These signals are too weak to be read by the A/D converters and hence they must be amplified to a higher voltage and current level before they can be read by the 0-1V or 0-5V range of the Analog inputs. The amplification stage is known as a **Signal Conditioner**. A Signal Conditioner consists of a precision instrumentation amplifier circuit to eliminate common mode noise that will swamp the weak signal if not handled properly. You can buy standard ready-made signal conditioners for a J or K type thermocouple or you can create you own using a highly integrated single-chip IC available from vendors such as Analog Device Inc (e.g. AD594/AD595) or from Linear Technology Inc.

The signal conditioners may have their own power supply. When selecting a signal conditioner, make sure that you select one with output signal in either 0-1V, 0-5V, 0-10V, 0-20mA or 4-20mA ranges to match that available on the PLC so that the analog data can be read easily.

1.9 Serial Communication Ports

The latest revision (Rev. D or D-1) of the T100M+ features two independent serial ports that can simultaneously communicate with other devices using a variety of protocols. They can also be programmed to accept or send ASCII or binary data using the TBASIC built-in commands such as INPUT$(n), INCOMM(n), PRINT #n, OUTCOMM n, d.

The first serial port (COMM1) is an RS232C port, which is compatible with most PC RS232C ports. The second serial port (COMM3) is a two-wire RS485 port that allows multiple PLCs to be connected to a single host computer or a master PLC for networking or to implement a distributed control system.

1.9.1 COMM1: RS232C Port with Female DB9 Connector

This port is configured as a DCE (Data Communication Equipment) and is designed to connect directly to the PC's serial port without the need for a null modem. COMM1 communicates with the host computer at a default baud rate of *38,400* bit-per-second with 8 data bits, 1 stop bit and no parity. if DIP switch SW1-4 is set during power-on, COMM1 default baud rate will be changed to 9600 baud. This is the main communication port for program transfer and on-line monitoring of the PLC. The pin connections with the host PC are shown below:

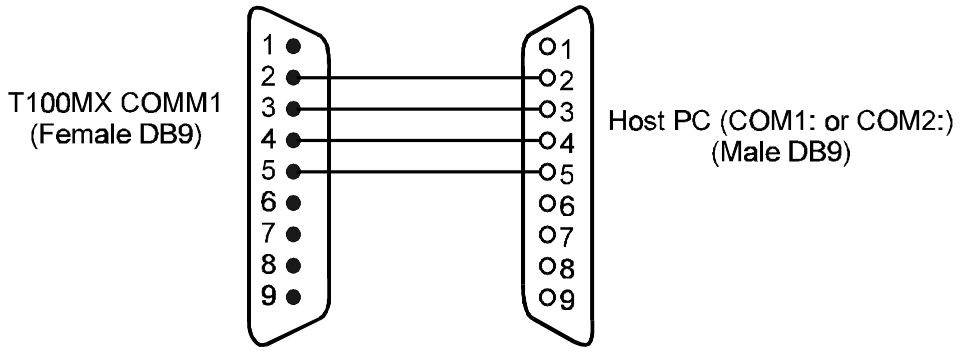


Figure 1.9 Connecting COMM1 with PC

However, to connect COMM1 to another DCE device (e.g., a modem), you need to make a special cable which swaps the transmit and receive signals, as follow:

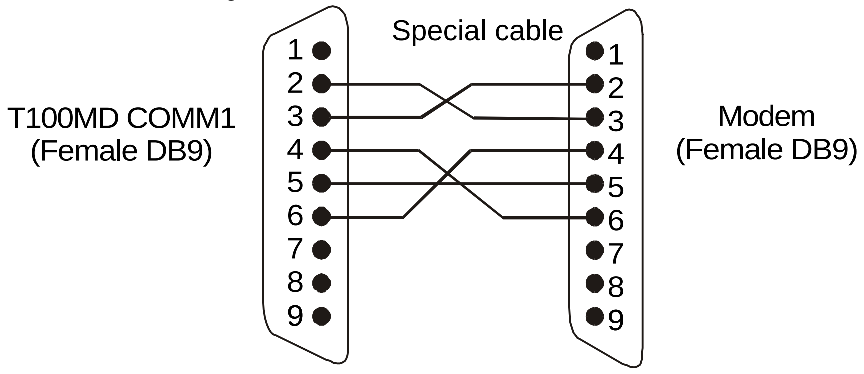


Figure 1.10 Connecting COMM1 to a MODEM

Pin 4 and 6 are handshaking signals whose presence may be required by some modems to work properly, so these pins are connected as shown in the diagram.

1.9.2 <u>COMM3: Two-wire RS485 Port</u>

This half-duplex port is meant for networking or for connecting to optional peripherals such as a serial LCD message-display unit (MDS100), touch panel HMI or for inter-communication between PLCs.

Up to 32 RS232 devices may be linked together in an RS485 network. By replacing the RS485 driver with a low power RS485 driver IC such as SN75HVD3082, up to 256 PLCs may be connected together.

The RS485 port is available on a 2-way screw terminal to the left side of the power supply terminal (please refer to Installation Guide). For successful communication using the RS485 port, you need to correctly connect the '+' and '-' terminals to the RS485 equipment using a twisted pair cable. If you are using the PC as the network host, you will need a RS232C-to-RS485 converter such as the "Auto485". The following describes some possible uses of the RS485 port.

a) PROGRAMMING AND MONITORING

A T100M+ PLC can be programmed via its RS485 port on a one-to-one or multi-drop manner. Since most PCs only have RS232 port(s) you need to purchase a RS232-to-RS485 converter in order to program the PLC via its COMM3 port. Most commonly available type of RS485 converters today use the RTS signal to control the RS485 transmitter direction, which is supported by TRiLOGI Version 4.x and the TLServer software. However, we strongly recommend auto-turnaround type of converter such as the Auto485 adapter (configured in 'Auto' mode) for use with Windows program. Under

Windows, the application software does not have very deterministic control of the RTS pin at precise timing and that can contribute to occasional loss of communication when the RS485 transceiver is not switched at proper moment.

Programming via COMM3 is particularly useful if COMM1 is already assigned to other tasks such as interfacing to modem, bar code readers, SCADA system or MMI, the programmer can continue to program and monitor the PLC using its RS485 port while its COMM1 is actively communicating with other devices. This makes it much easier to troubleshoot communication problems at COMM1 since you can continuously monitor the data exchange between the PLC and the external devices connected to its COMM1.

b) **CONNECT MANY PLCs TO A ONE TLSERVER OR ETHERNET XSERVER:** An Ethernet XServer or a single PC running the TLServer program can provide network services to all the PLCs connected to it via RS485 for remote programming, monitoring and control via the Internet using the Internet TRiLOGI 5.x or TRi-ExcelLink clients.

c) **DISTRIBUTED CONTROL:** Another important use of the RS485 port will be to connect a T100M+ PLC to other M-series, H-series or E10+ PLCs. One T100M+ PLC will act as the master and all other PLCs will act as slaves. Each PLC must be given a unique ID. The master will send commands to all the slaves using the "NETCMD" or READMODBUS, WRITEMODBUS, READMB2, WRITEMB2 statements and coordinate information flow between the PLCs. This allows a big system to be built by employing multiple units of M or H-series PLCs connected in a network. This results in more elegant implementation of complex control systems and simplifies maintenance jobs.

d) **INTERFACING OTHER DEVICES TO MODBUS OR INTERNET**: Since the T100M+ PLC supports MODBUS protocols, a master PLC can serve as the gateway to interface non MODBUS-enabled PLCs such as the H-series and E10+ PLCs, or the I-7000 analog modules to third party SCADA software or MMI hardware that speaks MODBUS. It also allows these devices to be controlled or monitored on the Internet via a T100M+ PLC. The master T100M+ will use its RS485 port to pull data from these devices into its data-memory. The data memory in the T100M+ PLCs are in turn accessible by a SCADA program using the MODBUS protocol and are also accessible from the Internet using the TRiLOGI client/server software.

1.9.3 Changing Baud Rate and Communication Formats: Use of the SETBAUD Statement

The T100M+ PLC's COMM ports are highly configurable. Both COMM ports can be set to a wide range of baud rates. You can also program them to communicate in either 7 or 8 data bits, 1 or 2 stop bits, odd, even or no parity. The baud rate and communication formats of the serial ports are set by the following command:

SETBAUD *ch, baud_no*

ch represents the COMM port number (1 or 3 only). The *baud_no* parameters takes value from 0 - 255 (&H0 to &HFF) which gives additional configuration of communication format. The upper 4 bits of *baud_no* specify the communication format (number of data bits, number of stop bits and parity) and the lower 4 bits represent the baud rate. Hence the *baud_no* for 8 data bit,1 stop bit and no parity is the same as the old models, providing compatibility across the family. Once the new baud rate has been set, it will not be changed until execution of another SETBAUD statement or when the power is turned OFF. The baud rate is not affected by software RESET. The available baud rates and their corresponding baud rate numbers for COMM1 are shown below:

Format	*baud_no*
8, 1, n	0000 xxxx
8, 1, e	0100 xxxx
8, 1, o	0110 xxxx
7, 1, n	1000 xxxx
7, 1, e	1100 xxxx
7, 1, o	1110 xxxx

Format	*baud_no*
8, 2, n	0001 xxxx
8, 2, e	0101 xxxx
8, 2, o	0111 xxxx
7, 2, n	1001 xxxx
7, 2, e	1101 xxxx
7, 2, o	1111 xxxx

Where xxxx represents the baud rate of the comm port, as follow:

x x x x	0000	0001	0010	0011	0100	0101	0110	0111
Baud Rate	2400	2400	4800	9600	19200	31250	38400	62500

x x x x	1000	1001	1010	1011	1100	1101	1110	1111
Baud Rate	100K	250K	500K	110	150	300	600	1200

A table of all the available baud rates and COMM formats is shown in the following page. The communication format written as 7,2,e means 7 data bits, 2 stop bits and even parity. Likewise, 8,1,n means 8 data bits, 1 stop bit and no parity. You can use the table to select the baud number for a certain baud rate and COMM format. Note that the circuit design of COMM1 limits its physical maximum baud rate to 100kbps, although its UART can work at up to 500K bits per second. COMM3 can work at the higher baud rate of up to 500K bps.

Baud No Table (All numbers in Hexadecimal: &H00 to &HFF)

Format / Baud	8,1,n	8,1,e	8,1,o	7,1,n	7,1,e	7,1,o	8,2,n	8,2,e	8,2,o	7,2,n	7,2,e	7,2,o
110	0B	4B	6B	8B	CB	EB	1B	5B	7B	9B	DB	FB
150	0C	4C	6C	8C	CC	EC	1C	5C	7C	9C	DC	FC
300	0D	4D	6D	8D	CD	ED	1D	5D	7D	9D	DD	FD
600	0E	4E	6E	8E	CE	EE	1E	5E	7E	9E	DE	FE
1200	0F	4F	6F	8F	CF	EF	1F	5F	7F	9F	DF	FF
2400	01	41	61	81	C1	E1	11	51	71	91	D1	F1
4800	02	42	62	82	C2	E2	12	52	72	92	D2	F2
9600	03	43	63	83	C3	E3	13	53	73	93	D3	F3
19200	04	44	64	84	C4	E4	14	54	74	94	D4	F4
31250	05	45	65	85	C5	E5	15	55	75	95	D5	F5
38400	06	46	66	86	C6	E6	16	56	76	96	D6	F6
62500	07	47	67	87	C7	E7	17	57	77	97	D7	F7
100K	08	48	68	88	C8	E8	18	58	78	98	D8	F8
250K	09	49	69	89	C9	E9	19	59	79	99	D9	F9
500K	0A	4A	6A	8A	CA	EA	1A	5A	7A	9A	DA	FA

E.g. To set baud rate of COMM3 to 19200, 7 data bit, 1 stop bit and even parity, execute the statement: `SETBAUD 3, &HC4`

Important: Since the two COMM ports are independent, they can be set to different format and baud rate from each other. Please note that if you change the baud rate or communication format to something that is different from that set in the **TLServer**, then both the TLServer and TRiLOGI will no longer be able to communicate with the PLC via this COMM port. You will have to either configure the TLServer's serial port setting using its "Serial Communication Setup" routine to match the PLC, or you can cycle the power to the PLC to reset the COMM port to the default format (38,400, 8,n,1).

If you had used "1st.Scan" contact to activate the SETBAUD command than you will need to cycle the power to the PLC with DIP switch #4 set to ON to halt the execution of the SETBAUD command. (Also remember that when the PLC is reset this way, its COMM1 will power up at 9600 bps only so you will need to temporarily configure TLServer's serial port to 9600bps to communicate with it.) If you need to re-access the port using TRiLOGI, then you will need to reset the PLC with DIP switch #4 set to ON so that the program will not execute a SETBAUD command.

1.9.4 <u>Support of Multiple Communication Protocols</u>

The T100M+ PLC is a real communication wizard! It has been designed to understand and speak many different types of communication protocols, some of which are extremely widely used *de facto* industry standard, as follows:

a) NATIVE HOST LINK COMMAND
b) MODBUS ASCII (Trademark of Groupe Schneider)
c) MODBUS RTU* (Trademark of Groupe Schneider)
d) OMRON C20H protocols. (Trademark of Omron Corp of Japan)

The command and response formats of the "NATIVE" protocols are described in details in Chapter 3 & 4. The other protocols and their address mapping to T100M+ are described in Chapter 5. The two independent COMM ports 1 & 3 support all the above protocols. Each COMM port can communicate using the same or different protocols independent of the other. The most wonderful feature of T100M+ PLC is that the support of all the above-mentioned protocols can be fully automatic and totally transparent to the users. There is no DIP switch to set and no special configuration software to run to configure the port for a specific communication protocols. The following describes how the automatic protocol recognition scheme works:

1) When the PLC is powered ON, both COMM ports are set to the "AUTO" mode, which means that they are open-minded and listen to all serial data coming through the COMM ports. The CPU tries to determine if the serial data conforms to a certain protocol and if so, the COMM mode is determined automatically.

2) Once the protocol is recognized, the CPU sets that COMM port to a specific COMM mode which enables it to process and respond only to commands that conform to that protocol. Error detection data such as the "FCS", "LRC" or CRC are computed accordingly which are used to verify the integrity of the received commands. If errors are detected in the command the CPU responds in accordance with the action specified in the respective protocols.

3) When the COMM port enters a specific COMM mode, it will regard commands of other protocol as errors and will not accept them. Hence for example if COMM #1 has received a valid MODBUS RTU command which puts it in a "RTU" mode, it will no longer respond to TRiLOGI's attempts to communicate with it using the "NATIVE" mode. You will receive a communication error if you try to use TRiLOGI to access a PLC COMM port that has just been communicating in other protocol modes.

4) To improve the flexibility of switching from one COMM mode to another, The T100M+ incorporates a COMM mode self-reset timer such that a specific COMM mode will time out automatically and enters into "AUTO" mode after 10 seconds if no more commands are received from that COMM port. When a user wants to switch

from one COMM mode to another he/she often will be changing the serial connector from one device to another. During this time there is no data received by the COMM port which presents an opportunity for it to reset its COMM mode. However, the surest way to reset the specific COMM mode is to cycle the power to the PLC so that its COMM port will be reset to "AUTO" mode and ready to communicate with any supported protocols.

5) if you wish to use the COMM port for serial data input only, you can use the **SETPROTOCOL** command to set the COMM port to **NO PROTOCOL**. This can prevent the PLC from erroneously treating some serial data as the header of an incoming communication protocol and respond to it automatically.

   SETPROTOCOL can also be used to set the PLC to a specific protocol. This may be desirable if the COMM port has a specific role and you do not want it to enter other modes by mistake. Please refer to the TBASIC Programmer's Reference manual for detailed description of the SETPROTOCOL command.

   Note: if you fix a COMM port to a non-native, non-auto mode TRiLOGI will not be able to communicate with the PLC anymore. You may have to power-cycle the PLC to reset the COMM mode. If you use "1st.Scan" contact to activate the SETPROTOCOL command than you will need to cycle the power to the PLC with DIP switch #4 set to ON to halt the execution of the SETPROTOCOL command. (Also remember that when the PLC is reset this way, its COMM1 will power up at 9600 bps only so you will need to temporarily configure TLServer's serial port to 9600bps to communicate with it.)

1.9.5 Accessing the COMM Ports from within TBASIC

Besides responding automatically to specific communication protocols described in section 1.9.4, both the serial ports COMM #1 and #3 are fully accessible by the user program using the TBASIC commands: INPUT$, INCOMM, PRINT # and OUTCOMM. It is necessary to understand how these commands interact with the operating system, as follow:

When serial data are received by a COMM port, the operating system of T100M+ automatically stores them into a 256 bytes circular buffer so that they can be retrieved by user programs later. The serial data are buffered even if they are incoming commands of one of the supported protocols described in section 1.9.4. In addition, processing of a recognized protocol command does not remove the characters from the serial buffer queue so these data are still visible to the user's program.

Each COMM port has its own separate 256-byte serial-in buffer. As long as the user-program retrieves the data before the 256-byte buffer is filled up, no data will be lost. If more than 256 bytes have been stored the buffer wraps around and the oldest data is overwritten first and so on. The following describes how INCOMM and INPUT$, PRINT # and OUTCOMM functions interact with the serial buffer:

a) <u>INCOMM (n)</u>

Every execution of the INCOMM(n) function removes one character from the circular buffer. When no more data is available in the buffer this function returns a -1. The data removed by INCOMM will no longer be available for the INPUT$ command.

b) <u>INPUT$(n)</u>

When the INPUT$(n) function is executed, the CPU checks the COMM #n buffer to see if there is a byte with the value 13 (the ASCII CR character) which acts as a terminator for the string. If a string is present, all the characters that make up the string will be removed from the COMM buffer. If a completed string is not present then the COMM buffer will not be affected and INPUT$(n) returns a null string. This ensures that before a complete string is received the serial characters will not be lost because of the unsuccessful execution of the INPUT$(n) function.

c) <u>PRINT #n</u>

The PRINT statement transfers its entire argument to a 256 byte serial-out buffer which is separate from the serial-in buffer. The PRINT statement therefore does not affect the content of the serial buffer for incoming characters. The operating system handles the actual transfer of each byte of data out of the serial-out buffer in a timely manner. Again each COMM port has its own independent, 256-byte serial-out buffer and hence the two serial ports can operate totally independent of each other.

Note that the PLC automatically enables the RS485 transmit driver when it sends serial characters out of its COMM3 port. When the stop bit of the last character in the serial-out buffer has been sent out, the operating system immediately disables the RS485 driver and enables the receiver. This greatly eases the use of the RS485 port since there is no need for user to bother with the often-critical timing of controlling the RS485 driver/receiver direction.

d) OUTCOMM

This command sends only a single byte out of the serial COMM port without going through the serial out buffer. For COMM3, it enables the RS485 transmitter before sending the character and disables it immediately after the stop bit has been sent out.

1.9.6 Using Modem to Remotely Program/Monitor T100MD+ PLC

TLServer 2.x supports remote dial up to T100MD+ PLC via standard, off-the-shelf modems. It takes two modems to communicate between two devices. The host end of the modem setup and configuration is handled TLServer software itself, whereas on the PLC side the PLC has to configure the modem so that it can successfully communicate with the host computer running TRiLOGI.

a) Wiring

The modem is often connected to the PLC's COMM1. Since the serial port on most modems are DCE type, you will need a make a special (also known as null-modem) cable to connect them as shown in figure 1.10. If the modem only has a DB25 connector you can connect the wires as shown in the following diagram:

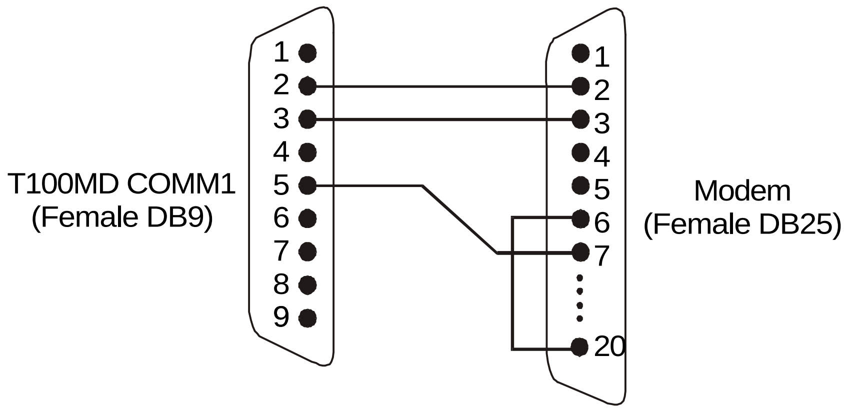


Figure 1.11 Connecting T100MD+ COMM1 to a modem's DB25 port

Note that pin 6 (DSR) and pin 20 (DTR) at the modem end are tied together. This is often required to inform the modem that the device is ready for operation so that the modem can work properly. A modem may also be connected to COMM3 for multi-drop remote programming and monitoring using TRiLOGI 5.x software. However, you will need to purchase an auto-turnaround type RS232-to-RS485 converter, such as the "Auto485".

b) Programming

Please refer to the Internet TRiLOGI version 5.2 Programmer's Reference guide, Chapter 3 for programming details for the PLC to communicate with the PC via modem.

1.9.7 Constructing a 2nd Multi-drop Network

For complex distributed applications, the built-in RS485 port may be required for internal networking between PLCs for data exchange. Yet some or all the PLCs may need to be connected to a SCADA system or MMI. It is possible to construct a second multi-drop network around the PLC COMM port #1. However, this will require a 4-wire RS485 or RS422 construction since the PLC COMM port #1 does not have built-in signal to enable/disable the transmitter and receiver of an RS485 driver IC. It is required that each PLC has an RS232-to-4-wire conversion interface so that they can be connected by a four-wire RS485/RS422 network to the SCADA host system. Of course there must also be a 4-wire RS485/RS422 converter at the host computer. The two COMM ports capability of the T100MD+ (Rev D) can be used to their fullest extent in such a situation. Please consult your local supplier or email to: info@tri-plc.com for questions regarding such applications.

1.10 DIP SWITCHES

DIP Switch	**OFF**	**ON**
SW1-1	All outputs, relays, timers and counter values are non-retentive.	Without MX-RTC module - no effect. If MX-RTC module has been installed, then all the I/Os, timers and counters as well as all internal variables retain their value after power off in the battery-backed RAM. DAC, PWM data will not be retained, however.
SW1-2	-	-
SW1-3	-	-
SW1-4	Normal Run mode	Suspends execution of ladder logic program. But host communication remains active. When power-on with this switch closed, default baud rate for COMM1 = 9600 bps instead of 38,400 bps.

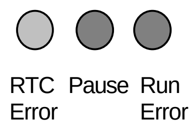
Usefulness of SW1-4

We have taken every effort to ensure that the host communication is always available even when the user-program ends up in a dead-loop. This allows the user to re-transfer a new program to the PLC and overwrite the bad program. However, you may still encounter a situation whereby after transferring a new program to the PLC, you keep encountering communication error and could not erase the bad program. This is especially common if you are playing with the communication commands such as `SETBAUD`, `SETPROTOCOL`, `PRINT` or `OUTCOMM` which may modify the communication baud rate, communication format or protocol or sending data out of a COMM port that conflicts with TRiLOGI. In such cases you can turn ON DIP Switch SW1-4 and perform a power-on reset for the PLC. The PLC will not execute the bad program that causes communication problem and you can then transfer a new program into the PLC to clear up the problem.

Note that when the PLC is power-reset with DIP Switch #4 set to ON, it's default baud rate and communication format for COMM1 becomes 9600, 8,n,1 (COMM3 is not affected). You will need to manually change the TLServer's serial port settings to 9600,n,8,1, in order to communicate with the PLC after a power-reset with DIP Switch #4 set to ON. (Remember to switch back to 38,400 after you have cleared the offending PLC program, otherwise if the PLC is again power-reset with DIP Switch #4 turned OFF, you will face problem communicating with it again because this time it would assume a baud rate of 38,400!!).

1.11 CPU Status Indicators

There are three LED indicators on T100MD+ with the markings shown on the right (T100MX has a fourth indicator named "WDT" which indicate a "Watch Dog Timer" reset)



All these indicators will be lighted up during power-on when the CPU loads the PLC program from EEPROM. Thereafter they should go off and if any one of them remains lighted it represents the various operating status of the PLC as follow:

a) RTC Error (Green LED)

This indicator will be turned ON after a power-ON or WDT reset unless an optional battery-backed MX-RTC module has been installed. This indicates that the real-time clock (RTC) has been reset to some factory's pre-set date and time. The RTC.Err flag in the "Special Bit" menu will also be turned ON. This indicator will be turned OFF automatically after you have set the PLC's date and time using the "Set PLC's Real Time Clock" command in the "Controller" pull-down menu.

b) Pause (Red LED)

This indicator will be turned ON if one of the following occurred:

i) PLC's EEPROM is corrupted.
ii) A PAUSE statement has been executed
iii) The user halts the PLC by pressing the <P> key during On-Line Monitoring.
iv) DIP-Switch SW1-4 is turned ON which halts the program.

If this light is ON, please connect the host computer running TRiLOGI to the PLC and run the "On-Line Monitoring" program. You will be informed of the reason that caused the PAUSE condition. Except for condition i) and iv), you can release the PLC from the PAUSE state by pressing the <P> key during "On-Line Monitoring". If the PLC's EEPROM is corrupted then you must re-transfer your program to the PLC again.

c) <u>Run Error (Red LED)</u>

When this indicator turns ON it shows that a run-time error had occurred during execution of a TBASIC command. The system will halt at the CusFn where the error took place. If the programmer now executes the "On-Line Monitoring" command in TRiLOGI, the cause of the run-time error and the CusFn where the error occurred will be reported on TRiLOGI screen.

TBASIC simulator captures many possible run-time errors including out-of-range values, but in T100M+ PLC only a few most important run-time errors are reported. The remainings are ignored. The following are the few run-time errors that will be reported in T100M+ PLC:

I) Divide By Zero
ii) FOR-NEXT loop with STEP = 0!
iii) Call Stack Overflow! Circular CALL suspected!
iv) Illegal Opcode - Please inform manufacturer!
v) System Variable Index out-of-range: This is normally caused by using an unavailable subscript. E.g. DM[0], INPUT[-1], DM[5000], etc. Check the subscript value especially if it contains a variable (e.g. DM[X], if X=0 this will lead to a runtime error).

All run-time errors should be identified and corrected before proceeding any further.

1.12 Internal Relays, Timers & Counters, etc.

All T100M+ PLCs support up to 512 internal relays, 64 timers (any one or all can be configured as "High Speed" timers), 64 counters, 8 clock sources of various periods: 0.01s, 0.02s, 0.05s, 0.1s, 0.2s, 0.5s, 1 sec and 1 minute.

T100M+ also supports 8 sequencers of 32 steps each. A sequencer is a highly convenient feature for programming machines or processes that operate in fixed sequences. Any one or all of the first 8 counters can be used as step counters for the sequencers that correspond to sequencers "Seq1" to "Seq8". Each step of the sequencer (up to 31) can be used as a contact to the ladder diagram as "SeqN:XX" where N = sequencers # 1 to 8. XX = Step # 0 - 31. Please refer to TRiLOGI Programmer's Reference for detailed descriptions of the built-in sequencers.

Chapter 2 Operating Procedure

2.1 Programming

The T100MD+ controller is programmed using the software DOS TRiLOGI Version 4.X or Internet TRiLOGI 5.x which run on an IBM compatible PC. This is a full-screen ladder logic editor, compiler and simulator software. TRiLOGI is a software package that provides a powerful programming and debugging environment for programming using the powerful ladder+TBASIC programming language. Please refer to the Internet TRiLOGI Programmer's Manual for details.

2.2 Simulation

A great feature unique to the TRiLOGI development environment is the built-in simulator. With the simulator, you can interact with your program by simulating the input conditions using your mouse or keyboard and examine the status and present values of the outputs, relays, timers and counters on screen immediately. Most Custom functions written in TBASIC can also be simulated and all the variables can be examined readily on the simulator screen.

The simulator does not require any physical connection to the target PLC, and thus offers the most effective way of testing and debugging your ladder logic program prior to the installation of the hardware. Programming and debugging time can be greatly reduced if you make good use of the simulator feature to eliminate as many logic errors as possible before testing the program on the actual hardware. It also helps to reduce the chances of costly damage to the machine due to programming errors.

2.3 Transferring Program to the PLC

Once you are satisfied with the TRiLOGI-simulated scenarios, return to the ladder logic editor by pressing the <ESC> key. To transfer the ladder program to T100MD, first connect the PC's serial port to COMM1 of the PLC and then turn on its power supply. You may press <Ctrl-T> on the keyboard or open the "Controller" pull-down menu and select item "Program Transfer". TRiLOGI will query the target controller to obtain its maximum number of inputs, outputs, etc. TRiLOGI will recompile the ladder program to ensure that these limits are not violated. When compilation is successful, the compiled code will be transferred to the T100MD PLC in within seconds.

After the program has been successfully transferred, you will be prompted to indicate if you wish to clear all outputs, relays, timers, counters and all the internal system variables to "OFF". A program that is successfully transferred will be executed at once. If you do not want the program to execute immediately, you may turn ON DIP switch SW1-4 before transferring the program, and then turn it OFF when you want the program to run.

If errors occur during program downloading and communication is aborted, the CPU will not execute the partially transmitted program to forestall undesirable consequences. If everything goes well, you may return to the editor by pressing any key.

Transfer Protection Password

The DOS version TRiLOGI 4.1x allows the user to define a Transfer protection password of between 1 to 6 characters by selecting the "Set Password" item from the "Target Access" menu. Once a password has been defined, you will be prompted to enter the password whenever you want to transfer a program to the PLC. Program transfer will be aborted if incorrect password is entered. This is to prevent alteration of the PLC program by unauthorized personnel.

If you have forgotten the password, then the only way to re-program the PLC is to first delete the password using the “Delete Password and Clear Program” command in the “Target Access” menu. The program in the PLC will be deleted when this command is executed. You have to download the new program into the PLC for it to operate.

* The password security against unauthorized programming is not supported on Internet TRiLOGI Version 5.x. There are already two levels of password structure on Internet TRiLOGI – one is defined on the TLServer and the other is defined by executing the SETPASSWORD TBASIC command. We feel that adding one more password layer to the whole PLC programming environment will only serve to confuse the users. We have thus decided to omit this from the Internet TRiLOGI Version 5.x. However, if you attempt to use Version 5.x to transfer program to a PLC previously protected by TL41.EXE, TRiLOGI Version 5.x will still prompt you for a “Prog. Transfer password”. You will need to enter the authenticated password in order to proceed any further. In other words, you can still use TL41.EXE to manage (define or delete) the Transfer Protection password for the PLC.

2.4 Errors and Problems

Any error in the source file detected during compilation will abort the program transfer process immediately. The cause of the first error will be reported on screen, although you should never encounter this problem if you had simulated the program successfully in TRiLOGI. This is because TRiLOGI's ease of programming reduces the possibility of errors to a minimum, and any error would have been detected and rectified before any simulation can take place.

PLC Program length is measured in (16-bit) “words”. Up to 6016 words may be programmed into a T100MD+ PLC (expandable to 8190 words with an optional hardware module M2018P, available from the manufacturer).

If your program exceeds the maximum allowable program size after compilation, the compiler will record this as an error and the downloading process will be aborted. If this happens, you need to simplify your program to optimize the use of program memory.

2.5 On-Line Monitoring & Control

TRiLOGI allows direct control of the PLC operation from within the program. You can enter this mode by selecting the "On-Line Monitoring" command from the "Controller" main menu, or by pressing the "Ctrl-M" hot-key. A "Full Screen On-Line Monitoring" screen will appear. The following are what may be done in this mode:

2.5.1 Monitoring PLC’s I/O Logic States

TRiLOGI continuously monitors the I/O logic states and present values of the timers and counters of the controller and displays them on screen. You may use the mouse to click on the scroll bar of each I/O column to scroll up and down to view elements outside of the window. The on/off logic state of each I/O element is clearly visible on screen.

2.5.2 Viewing and Modifying PLC’s Internal Variables

If you click on the “View” button while you are within the “Full Screen On-Line Monitoring” screen, a “View Variables” window will be opened. You can examine the values of all the 26 integer variables A to Z, string variables A$ to Z$, Data Memory DM[1] to DM[4000] and other special internal variables such as ADC, DAC, PWM and the Real-Time-Clock. The values displayed in this window reflect the actual values of these variables in real time. The numbers are usually displayed in decimal form, but if you press the <H> key it will change into hexadecimal form. Pressing the <D> key will change it back to decimal mode.

You can also examine the values of other system variables such INPUT[ ], OUTPUT[ ], EMINT[ ] etc. by right scrolling until you reach the last window. If you wish to modify the content of any variable, simply press the <E> key (as for “Edit”) and you can enter the variable name followed by the “=“ sign and the value. The entered value for the variable will be immediately updated into the PLC.

2.5.3 Force Setting/Resetting I/O Bits

While in TRiLOGI 5.x Full Screen on-Line Monitoring window, you can use left mouse button to click on any I/O bit shown on the screen. The selected I/O bit of the controller will be forced to ON by TRiLOGI using host link commands. When you release the left mouse button the affected I/O bit will be turned OFF. You can also force-toggle the I/O bit using the right mouse button.

If the selected bit is a physical input bit or has been assigned to an output coil controlled by the ladder diagram, it will only be affected for one-scan time. After that the controller will refresh its input/output according to the actual states of the physical inputs and outputs determined by the outcome of the ladder program. This is sometimes useful during program testing or debugging for temporarily overriding an I/O that does not respond as predicted.

Please see the Internet TRiLOGI Programmer’s Reference for more details.

2.5.4 Suspending PLC's Ladder Program

You can suspend the operation of the controller at any time by pressing the <P> key or by clicking the [Pause] button. A warning message will appear and the PAUSE button will be displayed in RED color. When the controller is suspended, its program will not be executed until it is resumed by pressing the <P> key again. At this time you can force set or reset any relay or output bits. This is convenient during programming or debugging as you can control the output driver to bring any physical component to any desired locations effortlessly.

2.6 Ladder Monitoring

When the “Full Screen Online Monitoring” window is opened, you can also monitor the logic states of I/Os directly on the ladder logic editor itself. TRiLOGI will continuously update the controller's I/O logic states and display any "ON" I/O bit with highlighted label names on the ladder diagram.

Note: On-Line Monitoring action is achieved by continuously sending host link commands to the PLC and analyzing the response strings immediately in order to update the screen. Since the controller must spare some time to process the host-link commands, the overall scan time will slow down during on-line monitoring. So please take precaution that programs which require fast scan-time, such as counters fed by the 0.01s and 0.02s clock sources, may lose some accuracy during online monitoring. Inputs based on interrupts, such as the High Speed Counters however will not be affected.

2.7 Get PLC's Hardware Info

You can find out the PLC's firmware version number, the maximum of input, outputs, relays, timers and counters supported on this PLC as well as the total amount of program memory available. The same info will be displayed when you try to transfer a program to the PLC.

2.8 Set PLC's Real Time Clock

This command lets you set the time and date of the PLC's built-in Real Timer Clock (RTC). When you execute this command, It will open up a table pre-filled with time and date data based on your PC's current date and time, but you can change the date and time value before updating to the PLC. The special bit "RTC.Err" will be turned OFF after you have executed this command.

MX-RTC Module

When the PLC power is turned off, the built-in RTC will stop operating and the date and time setting will be lost. When the power is re-applied to the PLC, the RTC must be reset to some factory pre-determined date and time values. In order to maintain the clock settings (non-volatility), you can purchase the MX-RTC option. The MX-RTC module is a special socket attached to the T100MD's data RAM and provides a Lithium battery-backed real time clock that continues to run even when the PLC power is turned off. The "Set PLC's Real Time Clock" command will also set the date and time within the MX-RTC module if installed.

The MX-RTC module also maintains the contents of all the I/Os and internal variables stored in the PLC's data RAM in the event of power lost. The DIP-switch SW1-1 can be set to avoid clearing of the variables when power on (please refer to section 1.10 for details). This may be useful for control systems that must maintain the contents of all data in the event of a power failure.

2.9 Trouble-Shooting Communication Errors

If you keep encountering the "Communication Error" message when you execute any command under the “Controller” menu, the following are some possible causes:

1) The T100MD is not connected to the cable.
2) The host computer COM port is not connected to the cable.
3) Wrong COM port number is specified for the PC. Try another one.
4) Power to PLC is not turned on or an inadequate power supply has been used. Make sure that the CPU power supply is within specifications. Try another power supply.
5) Faulty serial port of host computer. Try another computer with a good working COM port.
6) Faulty serial cable. Try another cable.
7) Faulty PLC. Return the unit to authorized dealer for servicing.

Communication Errors After Transferring A User Program

If you have been able to communicate with the PLC, but all a sudden, after transferring a new TRiLOGI program into the PLC you keep encountering the "Communication Error" messages, then the most likely causes are:

1) Your program has changed the serial port setting to other than 8 data bit, 1 stop bit and no parity. Or you have change the baud rate to values not supported by TLServer.

2) You are executing PRINT #, OUTCOMM, NETCMD$, READMODBUS, or WRITEMODBUS on the same COMM port which TRiLOGI connects to. TRiLOGI reports a comm error when it receives data that is different from the expected response from the slave.

To fix the above situation, turn ON DIP Switch SW1-4 and reset the PLC. change the Baud rate setting in TLServer to 9600 and attempt to communicate with the PLC again. If you are able to communicate with the PLC then the problem must definitely be caused by some offending codes in your TRiLOGI program. Correct the error and re-transfer the program before turning OFF DIP SW1-4 and change TLServer baud back to default 38400 bps.

Chapter 3 Host Communication

While a T100MD+ or T100MX+ PLC is running, a host computer or another T100M+ PLC (this abbreviation is used to refer to both the T100MD+ and T100MX+ in this manual) may send ASCII string commands to it to read or write to its inputs, outputs, relays, timers, counters and all the internal variables. These ASCII commands are known as the "host-link commands" and are to be serially transmitted (via RS232C or RS485 port) to and from the controller. The default serial port settings of T100M+ PLC for host-link communication are: *38400 baud, 8 data bit, 1 stop bit, no parity*. The baud rate and the communication format may be changed using the "SetBAUD" TBASIC command described in the Programmer's Reference Part II - TBASIC.

Multiple Communication Protocols

The competent T100M+ family of PLCs supports many different communication protocols to allow maximum application flexibility. In addition to its own native set of communication protocols, the T100M+ PLC also understands and speaks the following protocols:

1. ***MODBUS**™ ASCII mode compatible communication protocol.
2. ***MODBUS**™ RTU mode compatible communication protocol.
   (For Rev D board with Firmware revision r32 and above only)
3. ***OMRON**™ Host Link Commands for the C20H PLC family.

*Note: all trademarks belong to their respective owners.

The native host link command protocol will be described in detail in this chapter as well as in Chapter 4. The MODBUS and OMRON compatible protocols will be described in Chapter 5.

Native Mode Communication Protocols

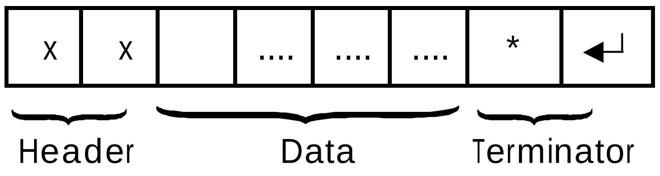
When a T100M+ PLC receives a native host-link command from COMM1 or COMM3, it will automatically send a response string corresponding to the command. This operation is totally transparent to the user and need not be handled by the user's program.

All T100M+ PLCs support both point-to-point (one-to-one) and multi-point (one-to-many) communication protocols. Each protocol has a different command structure as described below:

3.1 POINT-TO-POINT COMMUNICATION

In a point-to-point communication system, the host computer's RS232C serial port is connected to the PLC's COMM1. At any one time, only one controller may be connected to the host computer. The host-link commands do not need to specify any controller ID code and are therefore of simpler format, as shown below:

Command/Response Block Format (Point to Point)



Each command block starts with a two-byte ASCII character header, followed by a number of ASCII data and ends with a terminator which comprises an '*' character and a carriage return (ASCII value = 13_{10}). The header denotes the purpose of the command. For example, RI for Read Input, WO for Write Output, etc. The data is usually the hexadecimal representation of numeric data. Each byte of binary data is represented by two ASCII characters (00 to FF).

To begin a communication session, the host computer must first send one byte of ASCII character: Ctrl-E (=05Hex) via its serial port to the controller. This informs the controller that the host computer wishes to send a (point-to-point) host-link command to it. Thereafter, the host computer must wait to receive an echo of the Ctrl-E character from the controller. Reception of the echoed Ctrl-E character indicates that the controller is ready to respond to the command from the host computer. At this moment, the host computer must immediately send the *command block* to the controller and then wait to receive the *response block* from the controller. The entire communication session is depicted in Figure 2-1.

After the controller has received the command, it will send a response block back to the host computer and this completes the communication session. If the controller accepts the command, the response block will start with the same header as the command, followed by whatever information that has been requested by the command and the terminator.

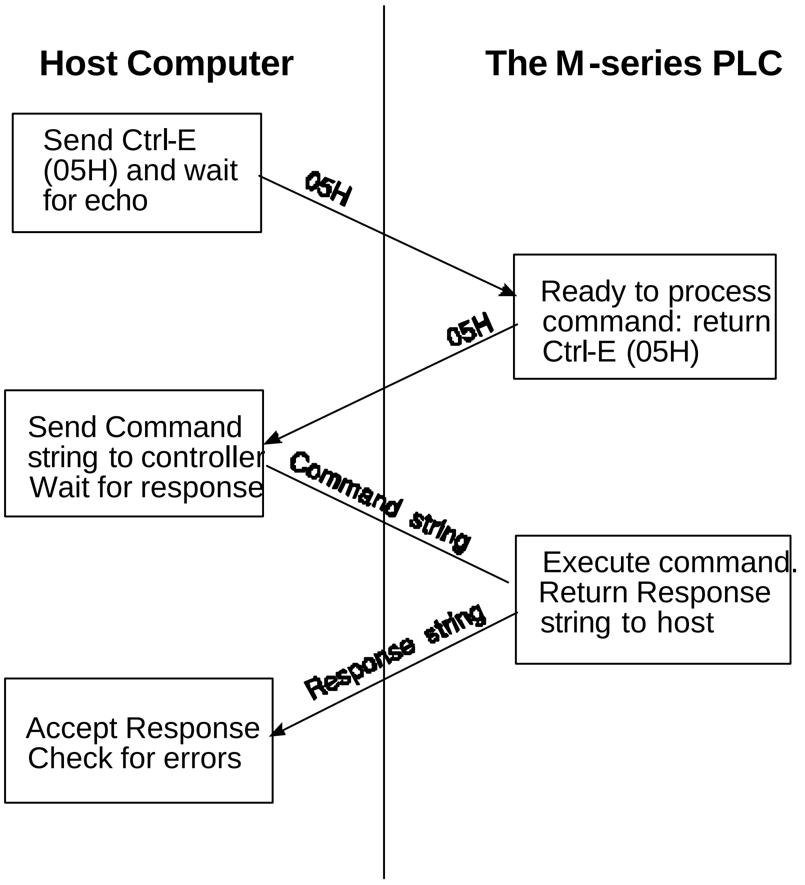

Host Computer
The M-series PLC
Send Ctrl-E (05H) and wait for echo
05H
Ready to process command: return Ctrl-E (05H)
05H
Send Command string to controller Wait for response
Command string
Execute command. Return Response string to host
Response string
Accept Response Check for errors

Figure 3.1

If an unknown command is received or if the command is illegal (such as access to an unavailable output or relay channel), the following **error response** will be received:

Error Response Format

E	R	*	◄┘

The host computer program should always check the returned response for possibilities of errors in the command and take necessary actions.

3.2 MULTI-POINT COMMUNICATION SYSTEM

In this system, one host computer may be connected to either a single T100M+ (via either RS232 or RS485) or multiple T100M+ PLCs on an RS485 network.

3.2.1 RS485 Network Interface Hardware

The built-in RS-485 interface allows the T100M+ controllers to be networked together using very low cost twisted-pair cables. Standard RS-485 allows up to 32 controllers (including the host computer node) to be networked together. When fitted with 1/8 power RS485 driver such as the 75HVD3082, up to 256 devices can be connected together. The twisted-pair cable goes from node to node in a daisy chain fashion and should be terminated by a 120 ohm resistor as shown below.

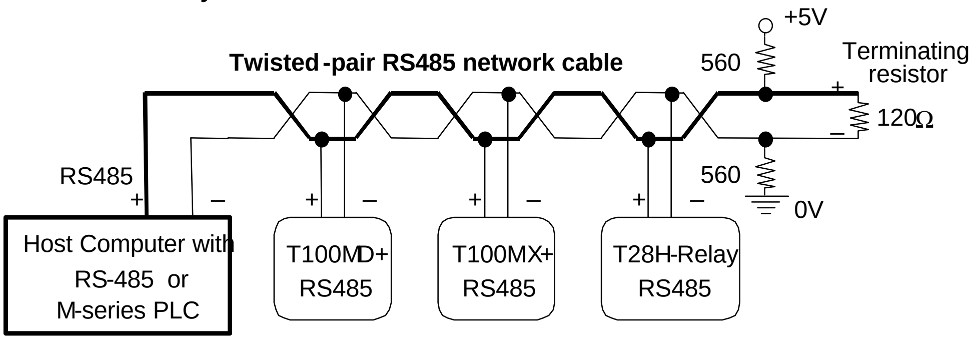


Figure 3.2

Note that the two wires are not interchangeable so they must be wired the same way to each controller. The maximum wire length should not be more than 1200 meters (4000 feet). RS-485 uses balanced or differential drivers and receivers, this means that the logic state of the transmitted signal depends on the differential voltage between the two wires and not on the voltage with respect to a common ground.

As there will be times when no transmitters are active (which leaves the wires in "floating" state), it is a good practice to ensure that the RS-485 receivers will indicate to the CPUs that there is no data to receive. In order to do this, we should hold the twisted pair in the logic '1' state by applying a differential bias to the lines using a pair of 560Ω to 1KΩ biasing resistors connected to a +9V (at least +5V) and 0V supply as shown in Figure 3-2. Otherwise, random noise on the pair could be falsely interpreted as data.

The two biasing resistors are necessary to ensure robust data communication in actual applications. Some RS485 converters may already have biasing built-in so the biasing resistors may not be needed. However, if the master is an M-series PLC then you should use the biasing resistor to fix the logic states to a known state. Although in lab environment the PLCs may be able to communicate without the biasing resistors, their use is strongly recommended for industrial applications.

3.2.2 Protection of RS485 Interface

The simple, direct multi-drop wiring shown in Figure 3-2 will work well if all the networked PLCs are in close proximity and they all share a common power supply. They will even work for long distance as long as no wiring error ever occurred. However, in an industrial environment, the PLCs are most likely far apart and they each may have their own power supply. Since processes are often modified regularly and if one day somebody by mistake shorts one of the PLC's RS485 to high voltage, **all the PLCs connected to the same RS485 wiring will be fried simultaneously.** This can result in very costly down time for the whole process, since all the PLCs connected to the network will need to be repaired.

Hence, for networking over long distances and involving more than a few PLCs, it is important to either strengthen or protect the RS485 interface, as described below:

1) You can replace the standard RS485 driver (75176) on the PLC by a fault-tolerant RS485 driver IC with part number LT1785AIN8. This 8 pin IC is made by Linear Technology and can withstand wrong voltages of up to $\pm$60V! As an added bonus, the LT1785AIN8 is a 1/4 power RS485 driver, which means up to 128 PLCs can be connected together.

   Unfortunately this IC is much more expensive than 75176 and hence it is not provided as standard component on the T100M+ PLC. You can purchase the IC from any major electronic catalog company or contact sales@tri-plc.com for a quotation of this IC driver.

2) When using non fault-tolerant RS485 driver such as SN75176 or SN75HVD3082, we strongly recommend the following protection circuit to be added between every PLC's RS485 and the twisted pair multi-drop network cable:

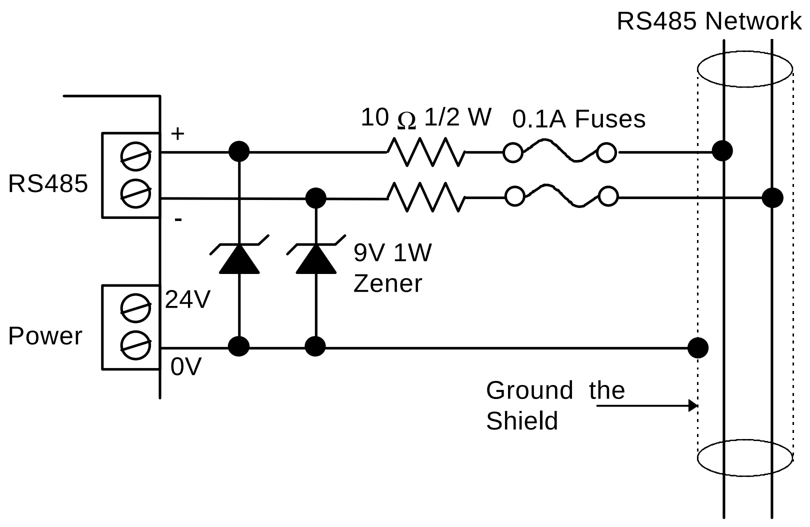


Figure 3.3

Note:

- As can be seen from the circuit, the two 9V Zener diodes clamp the signal voltage to the PLC's RS485 interface to between +9V and - 0.7V. If the high voltage persists, the 0.1A fuse will blow, effectively disconnecting the PLC from the offending network voltage.

- Even if you choose to replace the RS485 driver by LT1785AIN8 IC instead of using the zener/fuse pair wiring, you should still use shielded twisted pair cables as the multi-drop network "backbone" and connect the shield to the 0V (DC ground) power terminal of every PLC. The grounded shield then provides a common ground reference for all the different PLCs' power supplies. Even though the RS485 network may still work without a common ground reference because the signal wire pair will somehow "pull" all the RS485 to some reference point. **Failure to provide a common ground is a potential source of serious trouble** as signal wires with a floating ground easily induce large voltage differences between nodes when subjected to electromagnetic interference. Hence for reliable operation it is important to provide the common ground. A grounded shield also has the additional advantage of shielding the electrical signals from EMI.

3.2.3 Single Master RS485 Networking Fundamentals

RS485 is a half-duplex network, i.e., the same two wires are used for both transmission of the command and reception of the response. Of course, at any one time, only one transmitter may be active. The T100M+ PLCs implement master/slave network protocol. The network requires a master controller, which is typically a PC equipped with an RS485 interface. In the case of a PC, you can purchase an RS-485 adapter card or an RS232C-to-RS485 converter and connect it to the RS232C serial port. A T100M+ PLC can also be programmed to act as the master, it can communicate with other PLCs by executing the "NETCMD$" function or the "READMODBUS" or the "WRITEMODBUS" commands (the latter two are for communicating using MODBUS protocols only).

Only the master can issue commands to the slave PLCs. To transmit a command, the master controller must first enable its RS-485 transmitter and then send a multi-point command to the network of controllers. After the last stop bit has been sent, the master controller must relinquish the RS485 bus by disabling its RS485 transmitter and enabling its receiver. At this point the master will wait for a response from the slave controller that is being addressed. Since the command contains the ID of the target controller, only the controller with the correct ID would respond to the command by sending back a response string. For the network to function properly, <u>it is obvious that no two nodes can have the same ID</u>. You can use the "Setup Serial Port" command in TLServer to set the ID for each M-series PLC. You can also use the "IW" command to set the device ID. Also, all nodes must be configured to the same baud rate and communication format.

Also, care should be taken to ensure that the power supplies for all the controllers are properly isolated from the main so that no large ground potential differences exist between any controllers on the network.

3.2.4 Multi-Masters RS485 Networking Fundamentals

Since any T100MD or T100MX is capable of sending out network commands, the obvious question is whether multiple masters are allowed on the RS485 network? It is possible to have multiple masters on a single RS485 network provided the issues of collision and arbitration are taken care of. There are several means to achieve these objectives:

1) <u>Multiple Access with Collision Detection</u>

   There is nothing to stop any PLC from sending out host-link commands to other PLCs. However, If more than one PLC simultaneously enables their transmitters and send out host-link commands, then the signals will conflict and the messages will be garbled up. If the network traffic is low, then the solution may be a matter of having the master check for the correct response after sending out a command string. If there is error in the response string, the master should back off the network for a short while (use different timing for different PLCs) and then re-send the command until a correct response string is obtained. This scheme is similar to the CSMA/CD (Carrier Sensing Multiple Access/Collision-Detection) commonly used in Ethernet.

Fortunately, the "NETCMD$" function of T100M+ PLC automatically senses the RS485 lines until they are free before sending out the command string to reduce the chance of a collision. It also checks the integrity of the response string for correct FCS (Frame Check Sequence) characters before returning the string (Please refer to the Programmer's Reference for detail description of the NETCMD$( ) function).

However, the program must still check the following items in the response string to verify that the string returned from NETCMD$( ) function indeed comes from the PLC that it had talked to and not from another PLC (which tries to send a command to someone else):

i) The ID is correct
ii) The header is identical to the command string
iii) The length of response string is correct.

<u>*Pros and Cons:*</u> This method does not incur any hardware cost, but it requires careful programming and strict checking of the response string and hence requires more effort to program. It is also the least desirable if the network traffic is moderately high as many collisions will occur and there is danger of some undetected error being allowed to pass through.

2) <u>Token Awarding Scheme</u>

A "token" is a software means of telling a PLC that it has been given the right to temporarily act as the master. A T100MD+ PLC or a host PC can serve as the token master. An internal relay bit or a variable of the PLC can be defined as the token. The token master will begin by giving the token (i.e., by setting the token relay bit to '1' or the token variable to some fixed value) to the first PLC on the list. The PLC that has the token can then send host-link commands to other PLCs. When it has finished the job it can then send a command to the token master to relinquish its token. If it is based on a fixed timing scheme the master can assume that the PLC will complete its job after a fixed time (say 0.1 seconds) and turn off its corresponding token relay bit.

The token master then passes the token to the next PLC on the list and so on until the last PLC has relinquished its token, and the token is passed back to the first PLC on the list again. This way at any one time there will only be one active network master (the one with the token) and hence there is no danger of conflicting signals or garbled messages to handle.

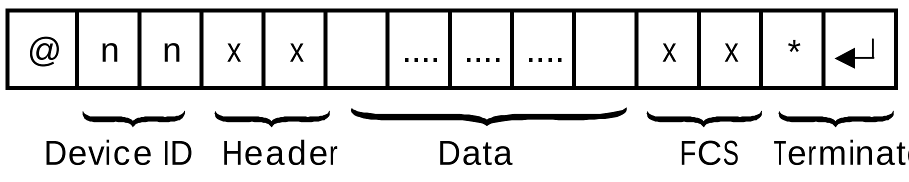
Pros and Cons: This method also does not incur any hardware cost, but it requires the programmer to draw up a plan on what internal relay or variable to use as the token and how the PLC can relinquish its token to the token master. (It could be by fixed timing or by returning a message to relinquish the token) It is a challenging job for programmers unfamiliar with networking scheme, but with some experimentation it can be achieved readily.

3) Rotating Master Signal

In this scheme we make use of the digital inputs of the T100M+ PLCs to grant the PLC the right to act as the network master. Lets call this input the "Be the Master" input. We can use a low cost H-series PLC running a sequencer to activate the "Be the Master" input line of each PLC one at a time. Each PLC is given a fixed amount of time to be the master (e.g. 0.1s each). Only when the "Be the Master" input is ON can the T100M+ PLC start sending out host-link commands to other PLCs. So at any one time there will only be one master on the network and no conflict will occur as a result.

Pros and Cons: This method is the easiest to program since there is no need to handle the token with the token master or perform extensive error check on the response string. However, this method uses one input of each PLC and as many outputs on the master-signal generator PLC as there are PLC masters. It also requires wiring the PLCs to the master-signal generator PLC and hence is the most costly method of all.

3.2.5 Command/Response Block Format (Multi-point)

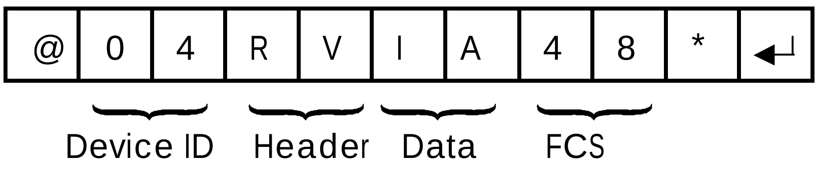


Each command block starts with the character "@" and two-byte hexadecimal representation of the controller's ID (00 to FF), and ends with a two-byte "Frame Check Sequence" (FCS) and the terminator. FCS is provided for detecting communication errors in the serial bit-stream. If desired, the command block may omit calculating the FCS simply by putting the characters "00" in place of the FCS.

Note: we call “00” the “wildcard” FCS, which is available when the PLC is in “auto protocol” mode. This is to facilitate easy testing of multi-point protocol. However, the wildcard FCS is disabled if the PLC has executed the SETPROTOCOL *n*, 5 to put it’s COMM port *n* into pure native mode. In that case you will have to supply the actual FCS to your command string.

Calculation of FCS

The FCS is 8-bit data represented by two ASCII characters (00 to FF). It is a result of Exclusive OR sequentially performed on each character in the block, starting from @ in the device number to the last character in the data. An example is as follow:



@	0100 0000
	XOR
0	0011 0000
	XOR
4	0011 0100
	XOR
R	0101 0010
	XOR
V	0101 0110
	XOR
I	0100 1001
	XOR
A	0100 0001

0100 1000 = 48_{16}

Value 48_{16} is then converted to ASCII characters '4' (0011 0100) and '8' (0011 1000) and placed in the FCS field.

FCS calculation program example

The following C function will compute and return the FCS for the "string" passed to it.

```
unsigned char compute_FCS(unsigned char *string){
   unsigned char result;
   result = *string++;     /*first byte of string*/
   while (*string)
      result ^= *string++; /* XOR operation */
   return (result);
}
```

A **Visual Basic** routine for FCS computation is included in the source code of a sample communication program you can download from:

http://www.tri-plc.com/applications/SerialComm.zip.

3.2.6 Communication Procedure

Unlike the point-to-point communication protocol, the host computer must NOT send the CTRL-E character before sending the command block. After the host computer has sent out the multi-point host-link command block, only the controller with the correct device ID will respond. Hence it is essential to ensure that every controller on the RS485 network assumes a different ID. Otherwise, contention may occur (i.e., two controllers simultaneously sending data on the receiver bus, resulting in garbage data being received by the host). On the other hand, if none of the controller IDs match that specified in the command block, then the host computer will receive no response at all.

The PLC automatically recognizes the type of command protocols (point-to-point or multi-point) sent by the host computer and it will respond accordingly. If a multi-point command is accepted by the controller, the response block will start with a character '@', followed by its device ID and the same header as the command. This will be followed by the data requested by the command, a response block FCS and the terminator.

Framing Errors

When the controller receives a multi-point host-link command block, it computes the FCS of the command and compares it with the FCS field received in the command block. If the two do not match, then a "framing error" has occurred. The controller will send the following Framing Error Response to the host:

Framing Error Response Block (Multi-point only)

@	x	x	F	E	x	x	*	↵
	Device ID		Header		FCS		Terminator	

Command Errors

If an unknown command is received or if the command is illegal (such as an attempt to access an unavailable channel), the following **error response** will be received:

Error Response Format

@	x	x	E	R	x	x	*	↵
	Device ID		Header		FCS		Terminator	

The host computer program should always check the returned response for possibilities of errors in the command and take necessary action.

3.3 SHOULD YOU USE POINT-TO-POINT OR MULTI-POINT PROTOCOL?

Although at first the point-to-point protocol appears simpler in format (having no ID and no FCS computation), the communication procedure is actually more complex since it involves the need to synchronize the two communicating devices by exchanging the Control-E character. The lack of error checking also makes the protocol less reliable especially in noisy environment.

In fact, the TLServer software as well as the Ethernet XServer will only accept multi-point communication protocol from the client software with the exception of the “IR*” command, which is needed to obtain the ID of a PLC with unknown ID.

Hence, if you were to write your own communication program to talk to the PLCs, we would strongly recommend using only the multi-point protocol exclusively due to its simplicity and built-in error checking capability.

3.4 TROUBLE-SHOOTING AN RS485 NETWORK

a) Single faulty device

If a single device on the RS485 network becomes inaccessible, problems can be isolated to this particular device. Check for loose or broken wiring or wrong DIP switch settings. Also double check the device ID using the host-link command "IR*" sent via the RS232C port of the PLC. If all attempts fail, either replace the entire PLC or the SN75176 chip that handles the RS485 interfacing and try again.

b) Multiple faulty devices

If all the PLCs are inaccessible by the host computer, it may possibly be due to a faulty RS232C-to-RS485 converter at the PC. If this is the case, disconnect the RS485 converter from the network and check it using a single PLC. Replace the converter if it is confirmed to be faulty. Next check the wire from the converter to the beginning of the network. A broken wire here can lead to the failure of the entire network.

Since an RS485 network links many PLCs together electrically and in a daisy chain fashion, problems occurring along the RS485 network sometimes affect the operation of the entire network. For example, a broken wire at the terminal of one node may mean that all the PLCs connected after this node become inaccessible by the master. If the RS485 interface of one of the PLCs has short-circuited because of component failure, then the entire network goes down with it too. This is because no other node is able to assert proper signals on the two wires that are also common to the shorted device.

Hence when trouble-shooting a faulty RS485 network, it may be necessary to isolate all the PLCs from the network. Thereafter, reconnect one PLC at a time to the network, starting from the node nearest to the host computer. Use the TRiLOGI program to check communication with each PLC until the faulty unit has been identified.

This page is intentionally left blank.

Chapter 4 Command/Response Format

This chapter describes the detail formats of the command and response blocks for all M-series PLC host link commands. Only the formats for the point-to-point communication protocol are presented, but all these commands are available to the multi-point protocol as well. To use a command for multi-point system, simply add the device ID (@nn) before the command header and the FCS at the end of the data (See Chapter 3 for detailed descriptions of multi-point communication command format).

4.1 Device ID Read

Command Format

I	R	*	↵

Response Format

I	R	16^1	16^0	*	↵

Device ID (00 to FF)

The device ID is to be used for multi-point communication protocol where the host computer can selectively communicate with any controller connected to a common RS485 bus (see Chapter 3 for details). The ID has no effect for point-to-point communication.

The device ID is stored in the PLC's EEPROM and therefore will remain with the controller until it is next changed.

4.2. Device ID Write

Command Format

I	W	16^1	16^0	*	↵

Device ID (00 to FF)

Response Format

I	W	*	↵

E.g. To set the PLC's ID to 0A, send command string "`IW0A*`" to PLC.

4.3 Read Digital Input Channels

Command Format

R	I	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	I	16^1	16^0	*	↵

8-bit Data (Hex)

Definition of Input Channels

The following table shows the input numbers as defined in TRiLOGI's Input entry table corresponding to the input channel number.

	Bit7			Input/Output Numbers				Bit0
CH00:	8	7	6	5	4	3	2	1
CH01:	16	15	14	13	12	11	10	9
CH02:	24	23	22	21	20	19	18	17
CH03:	32	31	30	29	28	27	26	25
CH04:	40	39	38	37	36	35	34	33
CH05:	48	57	56	45	44	43	42	41
CH06:	56	55	54	53	52	51	50	49
CH07:	64	63	62	61	60	59	58	57
CH08:	72	71	70	69	68	67	66	65
CH09:	80	79	78	77	76	75	74	73
CH0A$_{16}$:	88	87	86	85	84	83	82	81
CH0B$_{16}$:	96	95	94	93	92	91	90	89
CH0C$_{16}$:	104	103	102	101	100	99	98	97
....	..	..	..	..	..	..	..	..
CH1E$_{16}$:	248	247	246	245	244	243	242	241
CH1F$_{16}$:	256	255	254	253	252	251	250	249

The 8-bit inputs of each channel is represented by two bytes ASCII text expression of its hexadecimal value. For example: if inputs 1 to 3 are logic '0's, inputs 4 to 10 are logic '1's and all other inputs are logic '0's, then if you send command "`RI00*`", you will get response "RIF8`*`" (F8$_{16}$ =1111 1000$_2$).

4.4 Read Digital Output Channels

Command Format

R	O	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	O	16^1	16^0	*	↵

8-bit data (Hex)

Please refer to the Input/Output vs Channel Number table described in the section "4.3. Read Digital Input Channels" for details.

4.5 Read Internal Relay Channels

Command Format

R	R	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	R	16^1	16^0	*	↵

8-bit data (Hex)

Definition of Internal Relay Channel Numbers

All M-series PLC supports 512 internal relays, the channel definition of the first 256 internal relays is the same as the inputs and the outputs. The remaining relays and their assigned channels are shown in the following table:

	bit7			Relay numbers				bit0
CH20$_{16}$:	264	263	262	261	260	259	258	257
CH21$_{16}$:	272	271	270	269	268	267	266	265
CH22$_{16}$:	280	279	278	277	276	275	274	273
CH23$_{16}$:	288	287	286	285	284	283	282	281
CH24$_{16}$:	296	295	294	293	292	291	290	289
CH25$_{16}$:	304	303	302	301	300	299	298	297
CH26$_{16}$:	312	311	310	309	308	307	306	305
CH27$_{16}$:	320	319	318	317	316	315	314	313
CH28$_{16}$:	328	327	326	325	324	323	322	321
CH29$_{16}$:	336	335	334	333	332	331	330	329
CH2A$_{16}$:	344	343	342	341	340	339	338	337
CH2B$_{16}$:	352	351	350	349	348	347	346	345
..	..	..	..	..	..	..	..	..
CH3E$_{16}$:	504	503	502	501	500	499	498	497
CH3F$_{16}$:	512	511	510	509	508	507	506	505

4.6 Read Timer Contacts

Command Format

R	T	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	T	16^1	16^0	*	↵

8-bit data in Hex

Definition of Timer-Contact Channel Numbers

A timer contact is a single bit of memory and 8 timer contacts are grouped into one 8-bit channel similar to that of the inputs, outputs etc.

The following table shows the timer numbers defined in TRiLOGI's Timer entry table and their corresponding channel numbers.

CH0:	8	7	6	5	4	3	2	1
CH1:	16	15	14	13	12	11	10	9
CH2:	24	23	22	21	20	19	18	17
CH3:	32	31	30	29	28	27	26	25
CH4:	40	39	38	37	36	35	34	33
CH5:	48	57	56	45	44	43	42	41
CH6:	56	55	54	53	52	51	50	49
CH7:	64	63	62	61	60	59	58	57

4.7 Read Counter Contacts

Command Format

R	C	n	n	*	↵

8-bit channel # (Hex)

Response Format

R	C	16^1	16^0	*	↵

8-bit data in Hex

Definition of Counter-Contact Channel Numbers:

The 64 counter contacts are assigned channel # in exactly the same way as the 64 timers. Please refer to last section :"4.6. Read Timer Contacts" for details.

4.8 Read Timer Present Value (P.V.)

Command Format

R	M	N	n	*	↵

nn: Timer1=00, ... Timer16=0F.... Timer64=3F

Response Format

R	M	10^3	10^2	10^1	10^0	*	↵

Timer present value in Decimal

4.9 Read Timer Set Value (S.V.)

Command Format

R	m	n	n	*	↵

nn: Timer1=00, ... Timer16=0F.... Timer64=3F

Response Format

R	m	10^3	10^2	10^1	10^0	*	↵

Timer Set Value in Decimal

The Set Value (S.V.) of the specified timer is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this command header contains **small letter "m"** instead of "M" in the "RM" command.

4.10 Read Counter Present Value (P.V.)

Command Format

R	U	n	n	*	↵

nn: Counter1=00, ... Counter16=0F.... Counter64=3F

Response Format

R	U	10^3	10^2	10^1	10^0	*	↵

Counter present value in Decimal

The Present Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999.

4.11 Read Counter Set Value (S.V.)

Command Format

R	u	n	n	*	↵

nn: Counter1=00, ... Counter16=0F.... Counter64=3F

Response Format

R	u	10^3	10^2	10^1	10^0	*	↵

Counter Set Value in Decimal

The Set Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this header contains **small letter "u"** instead of "U" in the "RU" command.

4.12 Read Variable - Integers (A to Z)

Command Format

R	V	I	*alphabet*	*	↵

A,B.C....Z

Response Format

R	V	I	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

8 Hexadecimal Digit for 32-bit integer

E.g. To read the value of the variable "K", send host-link command "RVIK*". If variable K contains the value 123456_{10} (=$1E240_{16}$), PLC will send the response string as "RVI0001E240*".

4.13 Read Variable - Strings (A$ to Z$)

Command Format

R	V	$	*alphabet*	*	↵

A,B.C....Z

Response Format

R	V	$	*a*	*a*	*a*	...	...	*a*	*a*	*a*	*	↵

ASCII characters of the string (variable length)

E.g. To read the value of the string variable "M$", send host-link command "RV$M*". If variable M$ contains the string "Hello World", the PLC will send the response string as "RV$Hello World*".

4.14 Read Variable - Data Memory (DM[1] to DM[4000])

Command Format

R	V	D	16^3	16^2	16^1	16^0	*	↵

0001 to 0FA0 (4000_{10})

Response Format

R	V	D	16^3	16^2	16^1	16^0	*	↵

4 Hexadecimal Digit for 16-bit integer

E.g. To read the value of DM[3600], send host-link command "RVD0E10*". If variable DM[3600] contains the value 12345_{10} (=3039_{16}), PLC will send the response string as "RVD3039*".

4.15 Read Variable - System Variables

This command allows you to read all the M-series PLC's 16-bit system variables such as the inputs[ ], outputs[ ], relays[ ], counters[ ], timers[ ], timers' P.V., counters' P.V., CLK[ ] and DATE[ ]. Although inputs, outputs etc. are also accessible via the "RI", "RO", "RR"... commands, the RVS command can access them as 16-bit words instead of as 8-bit bytes in those commands. For the 32-bit system variable HSCPV[ ], use the "RVH" command described in the next section to access it. It may be more conventional for some SCADA software driver to use a single header command "RVS" to access all the I/O, varying only the "type" number to access different I/O types.

The RVS command also can be used to access the internal variables used to store ADC, DAC and PWM values obtained during the latest execution of the ADC(), setDAC or setPWM statement. These are however not system variables in TBASIC sense. E.g. it is illegal to use ADC[2] to access the ADC channel #2 in TBASIC (you have to use the ADC(2) function instead). An 8-bit hexadecimal number is used to denote the "type" of system variable, as shown in the following table:

System Variable	type
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	type
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

* Not a system variable in TBASIC

Command Format

R	V	S	n	n	16^1	16^0	*	↵

$\underbrace{n\ n}_{\text{type}}\quad\underbrace{16^1\ 16^0}_{\text{Index}}$

type (01 to 0D) - denote the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

Response Format

R	V	S	16^3	16^2	16^1	16^0	*	↵

$\underbrace{16^3\ 16^2\ 16^1\ 16^0}_{\text{4 Hexadecimal Digit for 16-bit integer}}$

Example: To read the value of DATE[2] (which represents the month of the RTC), send command “RVS0902*” and if the PLC responds with “RVS0005” it means the month is May.

4.16 Read Variable - High Speed Counter HSCPV[ ]

Command Format

R	V	H	n	*	↵

Channel: 1 or 2

Response Format

R	V	H	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

8 Hexadecimal Digit for 32-bit integer

E.g. To read the value of HSCPV[2], send hostlink command “RVH2*”. If variable HSCPV[2] contains the value 123456_{10} (=$1E240_{16}$), PLC will send the response string as “RVH0001E240*”.

4.17 Write Inputs

Command Format

W	I	n	n	16^1	16^0	*	↵

Channel # (00 to 0F) Data

Response Format

W	I	*	↵

4.18 Write Outputs

Command Format

W	O	n	n	16^1	16^0	*	↵

Channel # (00 to 0F) Data

Response Format

W	O	*	↵

4.19 Write Relays

Command Format

$$\boxed{\text{W}}\boxed{\text{R}}\underbrace{\boxed{\text{n}}\boxed{\text{n}}}_{\text{Channel \#}}\underbrace{\boxed{16^1}\boxed{16^0}}_{\text{Data}}\boxed{*}\boxed{\circlearrowright}$$

Response Format

$$\boxed{\text{W}}\boxed{\text{R}}\boxed{*}\boxed{\circlearrowright}$$

4.20 Write Timer-contacts

Command Format

$$\boxed{\text{W}}\boxed{\text{T}}\underbrace{\boxed{\text{n}}\boxed{\text{n}}}_{\substack{\text{Channel \#}\\ \text{(00 to 07)}}}\underbrace{\boxed{16^1}\boxed{16^0}}_{\text{Data}}\boxed{*}\boxed{\circlearrowright}$$

Response Format

$$\boxed{\text{W}}\boxed{\text{T}}\boxed{*}\boxed{\circlearrowright}$$

4.21 Write Counter-contacts

Command Format

$$\boxed{\text{W}}\boxed{\text{C}}\underbrace{\boxed{\text{n}}\boxed{\text{n}}}_{\substack{\text{Channel \#}\\ \text{(00 to 07)}}}\underbrace{\boxed{16^1}\boxed{16^0}}_{\text{Data}}\boxed{*}\boxed{\circlearrowright}$$

Response Format

$$\boxed{\text{W}}\boxed{\text{C}}\boxed{*}\boxed{\circlearrowright}$$

4.22 Write Timer Present Value (P.V.)

Command Format

$$\boxed{\text{W}}\boxed{\text{M}}\underbrace{\boxed{\text{n}}\boxed{\text{n}}}_{\substack{\text{Timer1=00,}\\ \text{........}\\ \text{Timer64=3F (Hex)}}}\underbrace{\boxed{10^3}\boxed{10^2}\boxed{10^1}\boxed{10^0}}_{\text{New timer PV}}\boxed{*}\boxed{\circlearrowright}$$

Response Format

W	M	*	↵

Please note that the timer number starts from 00 which represent timer #1, 01 represents timer #2... and so on.

4.23 Write Timer Set Value (S.V.)

Command Format

W	m	n	n	10^3	10^2	10^1	10^0	*	↵

Timer1=00, New timer SV
....
Timer64=3F (Hex)

Response Format

W	m	*	↵

Note: the 2nd character is a lower case “m” instead of the upper case “M” of “WM” command.

4.24 Write Counter Present Value (P.V.)

Command Format

W	U	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New PV
....
Counter64=3F (Hex)

Response Format

W	U	*	↵

4.25 Write Counter Set Value (S.V.)

Command Format

W	u	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New Counter SV
....
Counter64=3F (Hex)

Response Format

W	u	*	↵

Note: the 2nd character is a lower case “u” instead of the upper case “U” of the “WU” command.

4.26 Write Variable - Integers (A to Z)

Command Format

W	V	I	*alphabet*	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

A,B.C....Z 8 Hexadecimal Digit for 32-bit integer

Response Format

W	V	I	*	↵

E.g. To assign variable “K” to number 56789_{10}(=$0DD5_{16}$), send hostlink command “`WVIK00000DD5*`”.

4.27 Write Variable - Strings (A$ to Z$)

Command Format

W	V	$	*alphabet*	*a*	*a*	...	...	*a*	*a*	*	↵

A,B.C....Z ASCII characters of the string (variable length)

Response Format

W	V	$	*	↵

E.g. To assign the string “T100MD+ Super PLC” to the string variable P$, send hostlink command “`WV$PT100MD+ Super PLC*`”.

4.28 Write Variable - Data Memory (DM[1] to DM[4000])

Command Format

W	V	D	16^3	16^2	16^1	16^0	16^3	16^2	16^1	16^0	*	↵

16-bit Index to array 0001 to 0FA0 (4000_{10}) 16-bit Integer Data

Response Format

W	V	D	*	↵

E.g. To write the value 1234_{10} (=$4D2_{16}$)to DM[1000], send hostlink command "`WVD03E804D2*`". (1000_{10} = $3E8_{16}$)

4.29 Write Variable - System Variables

System Variable	type
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	type
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

* Not a system variable in TBASIC

Command Format

W	V	S	n	n	16^1	16^0	16^3	16^2	16^1	16^0	*	↵

type Index 16-bit Integer Data

type (01 to 0D) - denote the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

Response Format

W	V	S	*	↵

Example: To set clk[1] (which represents the hour of the RTC) to 14, send the command "`WVS0801000E*`" to the PLC.

4.30 Write Variable - High Speed Counter HSCPV[ ]

Command Format

W	V	H	n	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

1 or 2 8 Hexadecimal Digit for 32-bit integer

Response Format

W	V	H	*	↵

E.g. To clear the value of HSCPV[2], send hostlink command "`WVH200000000*`".

4.31 Update Real Time Clock Module

Command Format

W	r	*	↵

Response Format

W	r	*	↵

If the battery-backed MX-RTC module is installed, this command forces he PLC to write the values of the TIME[ ] and DATE[ ] variables into the RTC module. This command will be ignored by a PLC without the RTC module.

4.32 Halting the PLC

Command Format

C	2	*	↵

Response Format

C	2	*	↵

When the PLC receives this command, it temporarily halts the execution of the PLC's ladder program after the current scan. However, the PLC continues to scan the I/Os and processes host link commands sent to it and will report the current I/O data and internal variables to the host computer.

4.33 Resume PLC Operation

Command Format

C	1	*	↵

Response Format

C	1	*	↵

When the PLC receives this command, it will resume execution of the ladder program if it has been halted previously by the "C2" command. Otherwise, this command has no effect.

Important Note
The following Host Link Commands: RA, RXI, RX$, WA, WXI, WX$ and Wb are available only on newest M-series PLCs installed with CPU firmware version r47 & above. You can check your CPU firmware version by using the "Controller-> Get PLC Hardware Info" on the TRiLOGI software.

4.34 Read Analog Input *(r47 Firmware Only)*

This command forces the PLC to refresh the value of its ADC data at the analog channel before returning its value in the response string (i.e. no need for PLC to execute ADC(n) function to refresh the analog input)

Command Format

R	A	n	n	c	c	*	↵

$\underbrace{n\ n}_{\text{Starting Analog Channel \# (01-08h)}}$ $\underbrace{c\ c}_{\text{Channel count (01 to 08h)}}$

Starting Analog Channel # (01-08h) Channel count (01 to 08h)

Response Format

R	A	16^3	16^2	16^1	16^0	…	16^2	16^1	16^0	*	↵

Starting channel 16-bit Data (Hex) … Ending channel 16-bit Data (Hex)

E.g. To read 4 channels of Analog starting from Ch #2, Send "RA0204*". The response string will contain 4 sets of data for channel 2, 3, 4 and 5.

4.35 Read EEPROM Integer Data *(r47 Firmware Only)*

Command Format

R	X	I	n	n	n	n	c	c	*	↵

EEPROM starting Address (Hex) Word Count (01 to 20h)

Response Format

R	X	I	16^3	16^2	16^1	16^0	…	16^2	16^1	16^0	*	↵

1st EEPROM Integer 16-bit Data (Hex) … Last EEPROM 16-bit Data (Hex)

Maximum allowable word count per command is 32 (01 to 20 Hex). If "count" is > 32, only the first 32 words will be returned.

E.g. To read the 10 words of EEPROM data starting from address 100, send host-link command "`RXI00640A*`". The response string will contain 10 sets of 16-bit data (4 ASCII hex digit per set).

4.36 Read EEPROM String Data *(r47 Firmware Only)*

Command Format

R	X	$	n	n	n	n	*	↵

EEPROM String starting Address (Hex)

Response Format

R	X	$	*a*	*a*	*a*	...	...	*a*	*a*	*a*	*	↵

E.g. To read the string data stored at EEPROM address 10, send host-link command "`RX$000A*`". The response string will contain string data stored in the EEPROM (maximum 40 characters).

4.37 Write Analog Output *(r47 Firmware Only)*

Upon receiving this command, the PLC updates the value of its DAC data at the analog output channel (i.e. no need for PLC to execute SETDAC to update the analog output) .

Command Format

W	A	n	n	c	c	16^3	16^2	16^1	16^0	…	16^1	16^0	*	↵

Starting Analog channel # (01-02h) channel count DAC output data for 1st channel DAC output data for subsequent ch

Response Format

W	A	c	c	*	↵

channel count (Hex)

4.38 <u>Write EEPROM Integer Data</u> *(r47 Firmware Only)*

Command Format

W	X	I	n	n	n	n	c	c	16^3	16^2	16^1	16^0	…

Starting EEPROM Address (0001-xxxx) — count (01-20h) — Hex data for starting EEPROM address

16^3	16^2	16^1	16^0	…	16^1	16^0	*	↵

data for subsequent EEPROM addresses

Response Format

W	X	I	*	↵

Maximum allowable word count per command is 32 (01 to 20 Hex).

4.39 <u>WRITE EEPROM String Data</u> *(r47 Firmware Only)*

Command Format

W	X	$	n	n	n	n	*a*	*a*	...	...	*a*	*	↵

EEPROM String Address (Hex) — ASCII characters (max. 40 characters)

Response Format

W	X	$	*	↵

E.g. To write the string data "Hello TRi" at EEPROM String address 12, send host-link command "`RX$000CHello TRi*`".

4.40 <u>Force Set/Clear Single I/O Bit</u> *(r47 Firmware Only)*

This new "Wbnnnnxx" command allows you to change a single I/O bit on the PLC. You can force set or clear any single input, output, relay, timer or counter bit. This has advantage over other write commands such as WI, WO, etc that affects the entire group of 8 or 16-bits organized into "channels".

Command Format

$$\text{W}\ \text{b}\ \underbrace{\text{n}\ \text{n}\ \text{n}\ \text{n}}_{\text{I/O Bit address (Hex)}}\ \underbrace{\text{x}\ \text{x}}_{\substack{\text{00 – Clear I/O bit (OFF)}\\ \text{FF – SET I/O bit (ON)}}}\ *\ \hookleftarrow$$

Response Format

W	b	*	↵

I/O Type	Bit address nnnn (Hex)
Input #1 to #256	0000 to 00FF
Output #1 to #256	0100 to 01FF
Timer #1 to #256	0200 to 02FF
Counter #1 to #256	0300 to 03FF
Relay #1 to #256	0400 to 04FF
Relay #257 to #512	0500 to 05FF

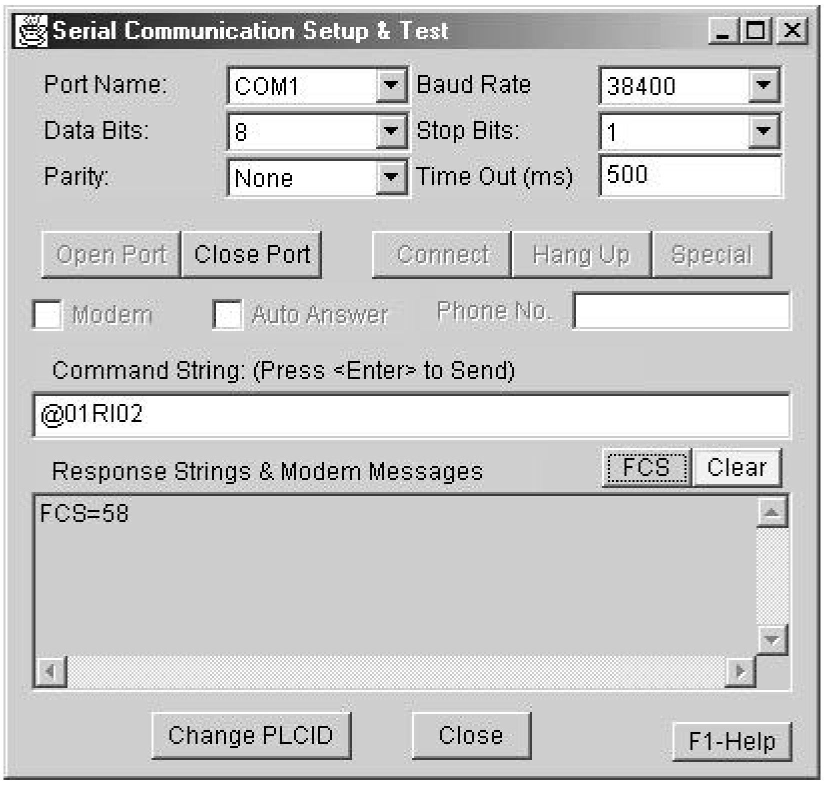
E.g. to force output 1 to ON, send “`Wb0100FF*`”. To turn it OFF, send “`Wb010000*`”

4.41 Testing of Host Link Commands

You can try out all the hostlink commands using the TLServer’s “Serial Communication Setup”. However, the TLServer is designed to accept only multi-point protocol except the “IR*” command (which is necessary to obtain the device ID from the PLC). So you have to enter all your host link commands in multi-point format.

Since the multi-point protocol requires an FCS (frame check sequence) character to be appended to the end of the command string, you may be able to get around it by using the “wildcard” FCS “00” in place of the actual FCS. E.g. To read input channel 02 from PLC with ID = 01, you can enter the command string as “`@01RI0200*`”.

For TLServer version 2.1 and above, there is an “FCS” button that let you compute the actual FCS for the string in the command string text field. You can then use the actual FCS with the command string to completely test your command. E.g. If you type in the string “`@01RI02`” in the command string (but do not press Enter), then click on the “FCS” button, the FCS for this string will be computed and shown as “FCS = 58”, as shown in the following figure:


Serial Communication Setup & Test

Port Name: COM1
Baud Rate: 38400
Data Bits: 8
Stop Bits: 1
Parity: None
Time Out (ms): 500

Command String: (Press <Enter> to Send)

@01RI02

Response Strings & Modem Messages

```
FCS=58
```

You now can enter the complete command string as "`@01RI0258*`" and it will be accepted by TLServer. (Note: If the PLC has executed a `SETPROTOCOL` *n*,5 to configure its serial port into pure native mode, then wildcard FCS will not be accepted and you must use the actual FCS with your command. The FCS button makes it much easier than computation by hand).

If you have changed some data using the write command, then activate On-Line Monitoring and examine the changes made using the "View Variables" window.

4.42 Visual Basic Sample Program

To help users get started writing their own Visual Basic program to communicate with the PLC, we have created a sample Visual Basic program with full source code listing. Please visit the following web page to download the visual basic sample program.

http://www.tri-plc.com/applications/VBsample.htm

4.43 Inter-PLC Networking Using NETCMD$ Command

All M-series PLCs are able to send out host link commands to other M-series or H-series PLCs using the built-in TBASIC function **NETCMD$()**. This function accepts host link commands in multi-point format and automatically computes the Frame Check Sequence (FCS) characters, append them to the command string and send out the whole command string together with the terminators. The function then waits for a response string and checks the integrity of the received response string for error. This function returns a string only if a proper response string has been received. Please refer to the TBASIC Reference for detailed explanation of this command.

The **NETCMD$()** function therefore greatly simplifies the programming tasks for handling networking between PLCs. The programmer only needs to construct the correct command string according to the formats described in this chapter, pass the formatted string to the **NETCMD$()** function and then check for the response string. An M-series PLC may use the NETCMD$ to map the I/O of another PLC into its internal relays and use the other PLC as its remote I/O.

There are two programming examples in your "TRILOGI\TL4" directory which illustrate the use of **NETCMD$()** to map I/Os of slave PLC to the master. Please study the two examples: "REMOTE-H.PC4" and "REMOTE-M.PC4" carefully to understand the mechanism of mapping I/Os between the PLC. The TRiLOGI program "REMOTE-H.PC4" will work on both H- and M-series PLCs as slaves , whereas the program "REMOTE-M.PC4" will only work with M-series slave PLC. This is because the M-series host link command set is a superset of H-series host link command set, and this example uses the more efficient M-series host link commands to read/write 16-bit data for networking between M-series PLC.

4.44 Inter PLC Networking Using MODBUS Protocols

The T100M+ PLCs may also pass data to each other using special MODBUS commands which are even simpler to use than NETCMD$ but are restricted to accessing variables that are mapped into MODBUS address structure. Please refer to the next chapter as well as the TBASIC Reference manual for details on using the READMODUS and WRITEMODBUS as well as the READMB2 and WRITEMB2 commands.

4.45 Using OMRON Host Link Commands

Since the T100M+ PLCs also support OMRON C20H Host Link commands, which are very similar in construct to our multi-point command/response format, you can also make use of OMRON commands to supplement the native host link commands.

We will only discuss four of the OMRON host link commands "RR", "WR", "RD" and "WD" in this section because these commands can be used by users to read/write to **multiple** I/O registers and data memory in a single command.

Note: Since the M-series native protocol command set typically only supports read/write of single variable and data memory, if you want to read/write multiple memory location in a single command you can make use of these OMRON host link commands.

I. Read IR Registers

This command refers to Table 5.1 in Chapter 5 to map the PLC's I/Os to OMRON IR register space from IR0 to IR519

Command Format

$$\boxed{@}\underbrace{\boxed{d}\boxed{d}}_{\text{Device ID}}\underbrace{\boxed{R}\boxed{R}}_{\text{Header}}\underbrace{\boxed{n}\boxed{n}\boxed{n}\boxed{n}}_{\text{IR Address (Dec)}}\underbrace{\boxed{c}\boxed{c}\boxed{c}\boxed{c}}_{\text{IR count (Hex)}}$$

$$\underbrace{\boxed{f}\boxed{f}}_{\text{FCS}}\boxed{*}\boxed{\circlearrowleft}$$

Response Format

$$\boxed{@}\underbrace{\boxed{d}\boxed{d}}_{\text{Device ID}}\underbrace{\boxed{R}\boxed{R}}_{\text{Header}}\underbrace{\boxed{s}\boxed{s}}_{\substack{\text{Status}\\ \text{00 – OK}\\ \text{15 - Bad}}}\underbrace{\boxed{16^3}\boxed{16^2}\boxed{16^1}\boxed{16^0}}_{\text{1st Data (Hex)}}\boxed{\dots}\boxed{\dots}$$

$$\underbrace{\boxed{16^3}\boxed{16^2}\boxed{16^1}\boxed{16^0}}_{\text{Last data}}\underbrace{\boxed{f}\boxed{f}}_{\text{FCS}}\boxed{*}\boxed{\circlearrowleft}$$

E.g. To read Timer PV #1 to #7 using this command, send:

"@01RR012800074D*"

The PLC will send return a response "@01RR00xxxxyyyyzzzz…. *"

II. WRITE IR Registers

This command refers to Table 5.1 in Chapter 5 to map the PLC's I/Os to OMRON IR register space from IR000 to IR519

Command Format

@	d	d	W	R	n	n	n	n	16^3	16^2	16^1	16^0	….
	Device ID		Header		IR Start Addr(Dec)				1st data				

16^3	16^2	16^1	16^0	f	f	*	↵
Last data				FCS			

Response Format

@	d	d	W	R	s	s	f	f	*	↵
	Device ID		Header		Status 00 – OK		FCS			

E.g. To Write to CtrPV #1 to #2 using this command, send:

“@01WR0256xxxxyyyyff*”

where xxxx and yyyy are the hex values to be written to CtrPV 1 & 2.

III. Read Data Memory DM[1] to DM[4000]

Command Format

@	d	d	R	D	n	n	n	n	c	c	c	c
	Device ID		Header		DM Address (Dec)				DM count (Hex)			

f	f	*	↵
FCS			

Response Format

@	d	d	R	D	s	s	16^3	16^2	16^1	16^0	…	…
	Device ID		Header		Status 00 – OK		1st Data (Hex)					

16^3	16^2	16^1	16^0	f	f	*	↵
Last data				FCS			

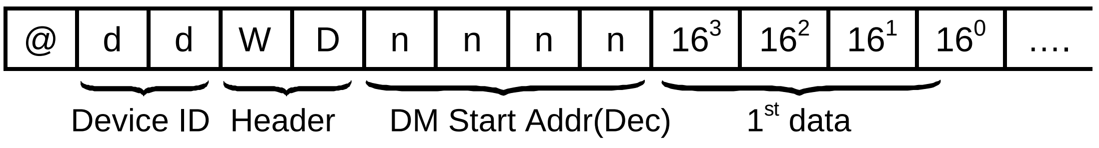
E.g. To read DM#112 to #130 (19 words), send:

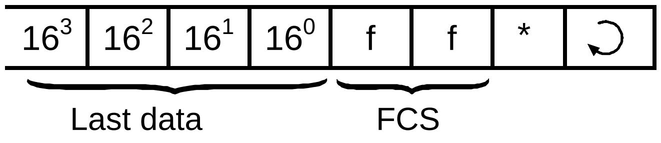
“@01RD0112001357*”

The PLC will send return a response “@01RD00xxxxyyyyzzzz…*”

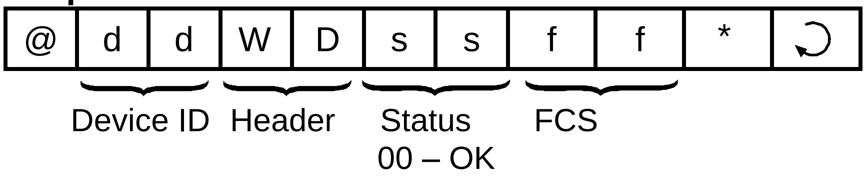
IV. <u>WRITE Data Memory DM[1] to DM[4000]</u>

Command Format


@ | d | d | W | D | n | n | n | n | 16³ | 16² | 16¹ | 16⁰ | ….
Device ID
Header
DM Start Addr(Dec)
1st data



Response Format



E.g. To Write to DM#1200 to #1201 using this command, send:

“@01WD1200xxxxyyyyff*”

where xxxx and yyyy are the values to be written to DM[1200] & DM[1201].

Chapter 5 MODBUS /OMRON Protocols Support

The T100M+ PLC supports a subset of the OMRON™ and MODBUS™ (Both ASCII and RTU modes are supported) compatible communication protocols so that it can be easily linked to third-party control software/hardware products such as SCADA software, touch panels etc. The PLC automatically recognizes the type of command format and will respond with the correct response. These are accomplished without any user intervention and without any need to configure the PLC at all!

Both MODBUS and Omron protocols use the same **device ID address** (00 to FF) as the native protocol described in Chapter 3. Since the addresses of I/O and internal variables in the T100M+ PLC are organized very differently from the OMRON or Modicon PLCs, we need to **map** these addresses to the corresponding memory areas in the other PLCs so that they can be easily accessed by their corresponding protocols. All I/Os, timers, counters, internal relays and data memory DM[1] to DM[4000] are mapped to Modbus Holding Registers space. The Inputs, Outputs, Relays, Timers and Counters bits are mapped to MODBUS Bit address space as shown in Table 5.1. Note that inputs and outputs bits are always mapped according to Table 5.1 whether it is MODBUS function 01, 02 or 05.

However, 32 bit variables and string variables are not mapped since they are fundamentally quite different in their implementation among different PLCs. Internal variables which are not mapped can be still be accessed by copying the contents of these variables to unused data memory DM[n] (these can be easily accomplished within a CusFn) so that they can be accessed by these third party protocols.

5.1 MODBUS ASCII Protocol Support

T100M+ supports MODBUS ASCII protocols with the following command and response format:

START	Address	Function	Data	LRC Check	CRLF
:	2 chars	2 chars	# chars	2 chars	2 chars

The following Function Codes are supported:

01/02	Read I/O bit (Use Bit Address Mapping in Table 5.1)
03/04	Read I/O Word registers
05	Force I/O Bit (Use Bit Address Mapping in Table 5.1).
06	Preset Single Word Register
16	Preset Multiple Word Registers

The exact command/response format of the MODBUS protocol can be found at http://www.modbus.org. However, if your only purpose is to interface the PLC to other MODBUS host such as LCD touch panel or SCADA software then there is no need to know the underlying protocol command format. All you need to know is which PLC's system Variable is mapped to which MODBUS register, as shown in Table 5.1.

BIT ADDRESS MAPPING

All the M-series I/O bits are mapped identically to both the MODBUS "0x" and 1x space. The bit register offset is shown in the last column of Table 5.1. Although MODBUS name the 0x address space as "Coil (which means output bits) and the "1x" address space as "Input Status" (which means input bits only), the T100M+ PLCs treat both spaces the same. Some MODBUS drivers only allow "read" from 0x space and "write" to 1x space but you still use the same offset shown on Table 5.1.

Example:

1) To map a lamp symbol to PLC Input 5, you select the MODBUS register address 0-0005. You can also map a lamp symbol to the PLC's output #2. In that case, you should map it to MODBUS register address 0-0258.

2) To map a toggle switch symbol to the PLC input #5, if you are restricted to select only MODBUS 1x address space, then you will have to map the switch to 1-0005, and likewise you can map the switch to output #2 using MOBDUS address 1-0258. But if the driver allows the switch to be mapped to 0x space then you can use MODBUS register space 0-0005 and 0-0258 for the mapping with identical result.

WORD ADDRESS MAPPING

As shown in Table 5.1, to access DM[1] from the PLC, you use MODBUS address space 4-1001 and so on. To access the Real Time Clock Hour data (TIME[1]), use 4-0513. The I/O channels can also be read or written as 16-bit words by using the addresses from 4-0001 to 4-0320.

Some MODBUS drivers (such as National Instrument "Looktout" software) even allow you to manipulate individual bit within a 16-bit word. So it is also possible to map individual I/O bit to "4x" address space. E.g. Input bit #1 can be mapped to 4-0001.1 and output bit #2 is mapped to 4-0257.2, etc. This is how it is shown in Table 5.1. However, if you do not need to manipulate the individual bit then you simply use the address 4-0001 to access the system variable INPUT[1]. and address 4-0257 to access the system variable OUTPUT[1]. Note that INPUT[1] and OUTPUT[1] are TBASIC system variables and they each contain 16 bits that reflect the on/off status of the actual physical input and output bits #1 to #16.

Table 5.1: Memory Mapping of T100M+ to other PLCs

T100M+ I/O #		**OMRON**	**MODBUS Word Addr. mapping**	**MODBUS Bit Addr. Mapping**
Input	**n**			**n**
	1 to 16	IR00.0 to IR00.15	40001.1 to 40001.16	1 to16
	17 to 32	IR01.0 to IR01.15	40002.1 to 40002.16	17 to 32
	33 to 48	IR02.0 to IR02.15	40003.1 to 40003.16	33 to 48
	49 to 64	IR03.0 to IR03.15	40004.1 to 40004.16	49 to 64
	65 to 80	IR04.0 to IR04.15	40005.1 to 40005.16	65 to 80
	81 to 96	IR05.0 to IR05.15	40006.1 to 40006.16	81 to 96
Output	**n**			**256 + n**
	1 to 16	IR16.0 to IR16.15	40017.1 to 40017.16	257 to 272
	17 to 32	IR17.0 to IR17.15	40018.1 to 40018.16	273 to 288
	33 to 48	IR18.0 to IR18.15	40019.1 to 40019.16	289 to 304
	49 to 64	IR19.0 to IR19.15	40020.1 to 40020.16	305 to 320
	65 to 80	IR20.0 to IR20.15	40021.1 to 40021.16	321 to 336
	81 to 96	IR21.0 to IR21.15	40022.1 to 40022.16	337 to 352
Timer	**n**			**512+n**
	1 to 16	IR32.0 to IR32.15	40033.1 to 40033.16	513 to 528
	17 to 32	IR33.0 to IR33.15	40034.1 to 40034.16	529 to 544
	33 to 48	IR34.0 to IR34.15	40035.1 to 40035.16	545 to 560
	49 to 64	IR35.0 to IR35.15	40036.1 to 40036.16	561 to 576
Counter	**n**			**768 + n**
	1 to 16	IR48.0 to IR48.15	40049.1 to 40049.16	769 to 784
	17 to 32	IR49.0 to IR49.15	40050.1 to 40050.16	785 to 800
	33 to 48	IR50.0 to IR50.15	40051.1 to 40051.16	801 to 816
	49 to 64	IR51.0 to IR51.15	40052.1 to 40052.16	817 to 832
Relay	**n**			**1024 + n**
	1 to 16	IR64.0 to IR64.15	40065.1 to 40065.16	1025 to 1040
	17 to 32	IR65.0 to IR65.15	40066.1 to 40066.16	1041 to 1056
	33 to 48	IR66.0 to IR66.15	40067.1 to 40067.16	1057 to 1072
	49 to 64	IR67.0 to IR67.15	40068.1 to 40068.16	1073 to 1088
	65 to 80	IR68.0 to IR68.15	40069.1 to 40069.16	1089 to 1104
	81 to 96	IR69.0 to IR69.15	40070.1 to 40070.16	1105 to 1120
	97 to 112	IR70.0 to IR70.15	40071.1 to 40071.16	1121 to 1136
	113 to 128	IR71.0 to IR71.15	40072.1 to 40072.16	1137 to 1152
	129 to 144	IR72.0 to IR72.15	40073.1 to 40073.16	1153 to 1168
	145 to 160	IR73.0 to IR73.15	40074.1 to 40074.16	1169 to 1184
	161 to 176	IR74.0 to IR74.15	40075.1 to 40075.16	1185 to 1200
	177 to 192	IR75.0 to IR75.15	40076.1 to 40076.16	1201 to 1216
	193 to 208	IR76.0 to IR76.15	40077.1 to 40077.16	1217 to 1232
	209 to 224	IR77.0 to IR77.15	40078.1 to 40078.16	1233 to 1248
	..	..	..	..
	497 to 512	IR96.0 to IR96.15	40097.1 to 40097.16	1521 to 1536

* MODBUS is a registered trademark of Groupe Schneider.
OMRON is a registered trademark of OMRON Corporation.

T100M+ Variables		OMRON	MODBUS
Timer Present Values	1 to 64	IR128 to IR191	40129 to 40192
Counter Present Values	1 to 64	IR256 to IR319	40257 to 40320
Clock	TIME[1] TIME[2] TIME[3]	IR512 IR513 IR514	40513 40514 40515
Date	DATE[1] DATE[2] DATE[3] DATE[4]	IR516 IR517 IR518 IR519	40517 40518 40519 40520
Data Memory	DM[1] DM[2] …. DM[4000]	DM[1] DM[2] …. DM[4000]	41001 41002 …. 45000

5.2 MODBUS RTU Protocol Support

The new Rev D of the T100MD+ or T100MX+ PLCs also supports the MODBUS RTU protocol. The difference between the ASCII and RTU protocols is that the latter transmits binary data directly instead of converting one byte of binary data into two ASCII characters. A message frame is determined by the silent interval of 3.5 character times between characters received at the COMM port. Other than that, the function codes and memory mappings are identical to the MODBUS ASCII protocol. Table 5.1 therefore applies to MODBUS RTU protocol as well.

MOBBUS RTU has following command and response format:

Start	Address	Function	Data	CRC 16	END
Silence of 3.5 char times	1 byte	1 byte	# byte	2 bytes	Silence of 3.5 char times

The following Function Codes are supported:

01/02	Read I/O bit (Use Bit Address Mapping in Table 5.1)
03/04	Read I/O Word registers
05	Force I/O Bit (Use Bit Address Mapping in Table 5.1).
06	Preset Single Word Register
16	Preset Multiple Word Registers

5.3 OMRON Host Link Command Support

Command Type	Header	Level of Support
a) TEST	TS	Full support
b) STATUS READ	MS	Full support
c) ERROR Read	MF	Dummy (always good)
d) IR Area READ	RR	Full support (0000 to 1000)
e) HR, AR, LR Area & TC Status READ	RH	Dummy (always returns "0000")
f) DM AREA READ	RD	Full support
g) PV READ	RC	Dummy (always returns "0000")
h) Status Write	SC	Dummy (always OK)
I) IR Area WRITE	WR	Full Support
j) HR, AR, LR Area & TC Status WRITE	WH, WJ, WL, WG	Dummy (always OK)
k) DM Area WRITE	WD	Full Support (from DM0001-DM4000)
l) FORCED SET	KSCIO KRCIO	Full Support for IR Area only Dummy for other areas.
m) Registered I/O Read for Channel or Bit	QQMR/ QQIR	Full Support for IR and DM only Dummy for other areas (always 0000)

Some OMRON host link commands are described in Section 4.45. For other commands please refer to Omron C20H/C40H PLC Operation manual published by OMRON Corporation. If your purpose is only to use the PLC's OMRON mode with SCADA or HMI then there is no need to learn the actual command/response format.

5.4 Application Example: Interfacing to SCADA Software

SCADA software or MMI systems (also known as LCD Touch Panels) normally use object-oriented programming method. Graphical objects such as switches indicator lights or meters, etc., are picked from the library and then assigned to a certain I/O or internal data address of the PLC. When designing a SCADA system, first you need to define the PLC type. You can choose the MODBUS ASCII, MODBUS RTU or OMRON C20H. Once a graphical object has been created, you will need to edit its connection and at this point you will be presented with a selection table that correspond to the memory map of that PLC type.

<u>Example 1: To connect an indicator lamp to Input #9 of the PLC.</u>

You will need to program the switch to connect to IR00.8 for OMRON protocol. However, If you have defined the PLC as MODBUS type then this indicator lamp should be connected to bit address 1-0265. In either case there is no need to learn about the actual command format of the protocol itself, as the SCADA software will automatically generate the required commands to access the input address that has been chosen for the object.

Example 2: To display reading from ADC #3 as a bar graph on SCADA.

Since the data from ADC #3 is not directly mapped to MODBUS or OMRON in Table 5.1, you need to add a statement in the custom function that reads the ADC #3 and copy it into a data memory, e.g.,

DM[100] = ADC(3)

Now you can program the bar graph on the SCADA screen to be connected to DM[100] if you use OMRON protocol. For MODBUS protocol the object should be connected to the address: 41100

5.5 Using The T100M+ PLC as MODBUS Master

The T100M+ PLCs supports for MODBUS protocol goes beyond being a MODBUS slave only. You can use the TBASIC READMOBUS and WRITEMODBUS commands, as well as READMB2 and WRITEMB2 (for PLC with firmware revision r44 and above) to send out MODBUS ASCII or RTU commands to access any other T100M+ PLC or any third party MODBUS slave devices. The READMODBUS or READMB2 command use MODBUS Function 03 to read from the slave, and WRITEMODBUS or WRITEMB2 use MODBUS Function 16 to write to the slave.

Note that when using READMODBUS or WRITEMODBUS command, the 40001 address stated in Table 5.1 should be interpreted as address 0000, and 40002 as address 0001 …. 41001 as address 1000, etc. This is in accordance with the specifications stated in MODBUS protocol. MODICON defined zero offset address for the MODBUS protocol, yet in their holding register definition these are supposed to start from address 40001 - hence the unusual correspondence. But to maintain compatibility with the MODBUS specifications we have to adhere to their definitions.

M+ PLC As MODBUS RTU Master

The new Rev. D T100M+ PLC can also act as a MODBUS RTU master! The same READMODBUS and WRITEMOBUS commands can be used to send and receive MODBUS RTU commands. What you need to do is to add 10 (decimal) to the COMM port number to signal to the processor that you wish to use MODBUS RTU instead of ASCII to talk to the slaves. I.e. you should specify port #11 to use RTU commands on COMM1, and specify port #13 to use RTU commands on COMM3. E.g. the statement DM[10] = READMODBUS (13, 8, 16) will access via COMM3 the slave with ID = 08 and read the content of register #16. This register corresponds to MODICON address 40017 and is the OUTPUT[1] of the slave PLC. The ability to speak MODBUS RTU greatly extends the type of peripherals that can be used with a T100M+ PLC. You can now make use of many off-the-shelf, third party RTU devices to extend the PLC capability, making the M-series truly super PLCs!