CHAPTER

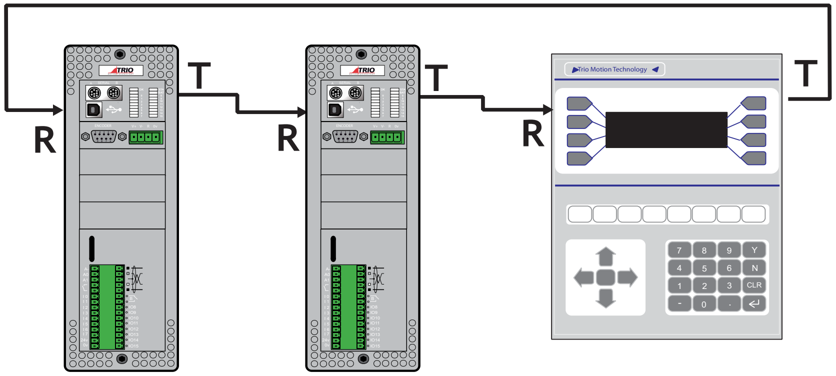
14

Fibre-Optic Network

General Description

The TRIO fibre optic network has been designed to link up to fifteen *Motion Coordinator* modules and membrane keypads. Any number of either type of module can be on the network up to the maximum of fifteen but at least one must be a *Motion Coordinator*.

The physical connection of the network takes the form of a ring. The interconnections between nodes being made with fibre optic cable.



Example of Network Connection

There are three fixed types of message that can be transmitted round the network. These are:

(i) Character transfer

- a string of characters is transmitted to a specified node e.g. a message to a display. This is performed with a PRINT# command. Reception of characters to a *Motion Coordinator* is carried out with the GET# command.

(ii) Direct variable transfer

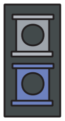
- a specified variable on a given *Motion Coordinator* node can be modified by another *Motion Coordinator* node independent of the program running on the receiving node. This is achieved with the SEND command.

(iii) Keypad offset

- a special message sent to a membrane keypad node to set it into network mode and to tell it where on the network to direct its key presses to. This message type is also transmitted by the SEND command.

Connection of Network

All the nodes have to be connected to form a ring. Single core fibre optic cable is used terminated with Hewlett Packard “Versatile Link” connectors. The fibre optic modules on the *Motion Coordinator* and membrane keypad are colour coded as follows:



Grey or Black	Transmitter
Blue	Receiver.

To form the ring the transmit of a node is connected to the receive of the next node, i.e. grey/black to blue. The transmit of the final node is connected to the receive of the first thus completing the ring.

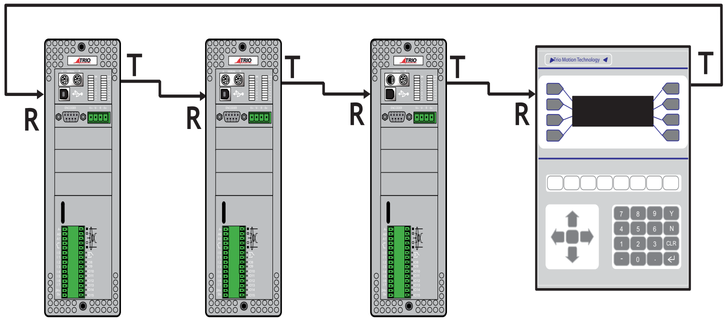
The order in which nodes are connected is not critical but the performance of the network can be affected by rearranging the order of the nodes. This is especially true if there are a large number of nodes in the network and a large amount of data being passed between two nodes. If these two nodes are not adjacent in the ring any nodes between them will be tied up re-transmitting messages and will not be able to transmit as priority is given to re-transmission of messages. This can be a particular problem if one of the nodes re-transmitting is a membrane keypad as this can cause poor response to key presses.

Fibre-optic cables

Trio can supply a fibre-optic connection kit (P570 Simplex Cable Kit) comprising a length of fibre-optic cable, connectors and assembly components.

Network Node Addressing

Nodes on the network are not given absolute addresses. Instead, when a message is sent it is labelled with the address of a destination node which defines the number of nodes along the ring from the sender that the message must travel. The addresses are in the range 10, denoting a message for the next node along the ring, to 24, denoting a message for the fifteenth node along the ring from the sender.



Node Addressing Example

The example above shows the destination node addresses for a network with respect to the transmitting *Motion Coordinator*. If a message was sent with the address 13 then the message would come back to the sender. Likewise if address 14 was used the message would completely traverse the ring once and finish up at node #10. If address 18 was used then the message would go twice round the ring and finish up at node #10.

Network Programming

Trio BASIC Commands

The transmission and reception of messages on the network is performed by four Trio BASIC commands.

GET #n

Type: Command.

Syntax: **`GET#n,VR(x)`**

Description: Waits for the arrival of a single character on the specified input device. The ASCII value of the received character is stored in the chosen variable. The characters received are held in a 256 character buffer.

Parameters:

n	Number to specify how the returned value generated.
3	value returned is defined by DEFKEY
4	value returned is character received
x	Variable number in the range 0 to 250.

KEY#n

Type: Function.

Syntax: **`IF KEY#n THEN GET #n,VR(x)`**

Description: Returns **TRUE** or **FALSE** depending on whether a character has been received on the specified input device or not. The character is not read nor the receive state reset.

Parameters:

n	Number to specify serial input device.
3	network input from fibre optic port via DEFKEY table
4	network input from fibre optic port

PRINT#n,

Type: Command.

Syntax: `PRINT #11, CURSOR(20);"Printed on Keypad 2"`

Description: The **PRINT#** command allows the program to output a series of characters to the specified output device. The **PRINT#** command can output parameters, variables, fixed ASCII strings and single ASCII characters. Multiple items to be printed can be put on the same **PRINT** line provided they are separated by a comma or semi-colon. The comma and semi-colon are used to control the format of strings to be output.

Parameters:

n: Number from 10 to 24 to specify number of nodes from transmitting node the message must be sent.

SEND

Type: Command.

Syntax: `SEND(n,type,data1[,data2])`

Description: Outputs a network message of a specified type to a given node.

Parameters:

n: Number from 10 to 24 to specify number of nodes from transmitting node the message must be sent.

type: Message type:

1 - Direct variable transfer
2 - Keypad offset

data1 if type=1: data1 is the variable number on the destination node to modify.

If type=2: data1 is in the range 10..24 to specify the number of nodes from the keypad that the key characters are sent.

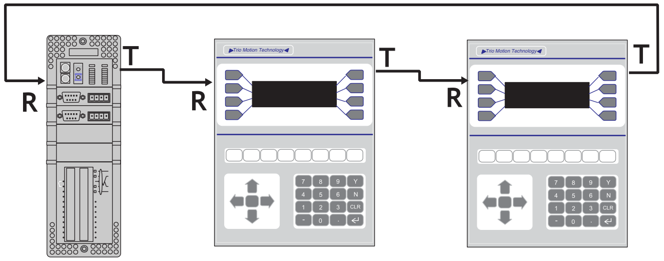
data2: If type=1: data2 is the value to change the specified variable to.

If type=2: data2 is not used.

Examples of network programming

Example 1 Consider a four axis machine which is 30m long. The operator has need to make machine adjustments at either end of the machine, thus requiring two stations for operator input. This can be achieved using a *Motion Coordinator*, four Servo Daughter Boards and two membrane keypads. The two keypads and the *Motion Coordinator* will be networked together so input from both keypads is given to the *Motion Coordinator* and any resulting change in machine parameters is displayed at either end of the machine.

The network should be connected as shown below.



Network Example 1

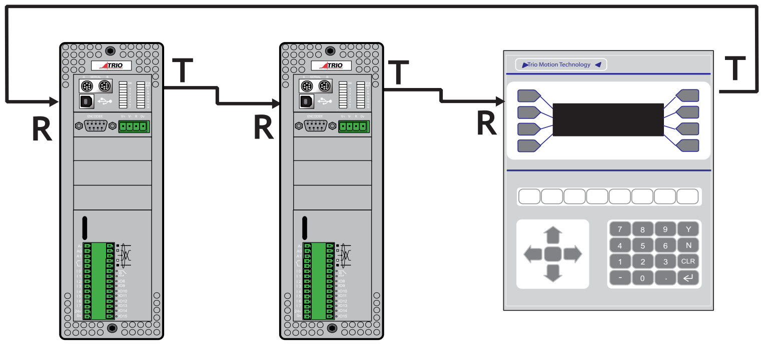
The program could look something like this:

```
length=2
' Set offset on first keypad to 2, i.e. send key press to MC
SEND(10,2,11)
' Set offset on second keypad to 1, i.e. send key press to MC
SEND(11,2,10)

' Clear first display & make cursor invisible
PRINT #10,CHR(12);CHR(14);CHR(20)
' Clear second display & make cursor invisible
PRINT #11,CHR(12);CHR(14);CHR(20)
' setup display
PRINT #10,"SPEED:":PRINT #10,"LENGTH:"
PRINT #11,"SPEED:":PRINT #11,"LENGTH:"

REPEAT
  IF KEY#3 THEN GOSUB read_key ELSE GOSUB do_motion
UNTIL FALSE
read_key:
  GET #3,k
  IF k>9 THEN GOTO not_num
  a=a*10+k
  IF a>9999 THEN a=0
  PRINT #10,CHR(27);CHR(72);CHR(7);a[6,0]
  PRINT #11,CHR(27);CHR(72);CHR(7);a[6,0]
not_num:
  IF (k=10)&(VR(length)<10000) THEN VR(length)=VR(length)+0.1
  IF (k=11)&(VR(length)>0.1) THEN VR(length)=VR(length)-0.1
  PRINT #10,CURSOR(28);VR(length)[8,1]
  PRINT #11,CURSOR(28);VR(length)[8,1]
RETURN

do_motion:
' Main motion routine
```

Example 2 Consider a five axis machine with one membrane keypad. The axes can be divided into two distinct blocks. Two axes are concerned with the feeding of the material and the other three with the cutting and packing of the material.

This could also be achieved with an MC206X or an MC224 + Axis expander, but in this example because each block can be built as a stand-alone unit, each requires it's own *Motion Coordinator*. In this example our system uses two MC224 controllers.

Our network will consist of the two *Motion Coordinators* and the membrane keypad. It should be connected as shown in below.



Network Example 2

As there are now two *Motion Coordinator*s in the network it is necessary to have a program on each.

For the purposes of this example the three axis node will act as the master and issue instructions to the two axis node. The three axis node will also receive all input from the keypad.

The program for the three axis node could be as follows:

```
init:  SEND(11,2,10)'     Set offset on keypad
     PRINT #11,CHR(12);CHR(14);"INITIALISING......."
     PRINT #10,"I";'     Send command to two axis node to
initialise
     GOSUB initax'     Initialisation routine
     WAIT UNTIL VR(200)=99'Two axis node signals ready by
setting variable 200
     PRINT #11,CHR(12);CHR(14);"MACHINE READY"

     'Get required parameters
setlen:
   PRINT #11,"FEED LENGTH:";VR(0)[6.1];
   GET #3,VR(100)
   IF VR(100)=13 THEN GOTO setsp
   IF (VR(100)=10)AND(VR(0)<1000000) THEN VR(0)=VR(0)+0.1
   IF (VR(100)=11)AND(VR(0)>0.1) THEN VR(0)=VR(0)-0.1
   PRINT#11,CHR(27);CHR(72);CHR(32);VR(0)[6.1];
GOTO setlen
setsp:
   PRINT #11,"SPEED:";VR(1)[2];
   GET #3,VR(100)
   IF VR(100)=13 THEN GOTO initnet
   IF (VR(100)=10)AND(VR(1)<100) THEN VR(1)=VR(1)+0.01
   IF (VR(100)=11)AND(VR(1)>0.01) THEN VR(1)=VR(1)-0.01
   PRINT #11,CHR(27);CHR(72);CHR(46);VR(1)[2];
GOTO setsp

initnet:
   'Update variable values on two axis node
   SEND(10,1,0,VR(0))
   SEND(10,1,1,VR(1))

start:
   PRINT #11,CHR(12);CHR(14);"PRESS START TO RUN"
```

```
readkey:
  WAIT UNTIL KEY#3
  GET #3,VR(100)
  IF VR(100)<>20 THEN GOTO readkey
  PRINT #10,"G";
  WAIT UNTIL VR(200)=999

moves:  'Main motion routine
initax: 'Initialisation routine
```

The program on the two axis node could be as follows:

```
readinit:
  IF NOT(KEY#4) THEN GOTO readinit
  GET #4, VR(100)
  IF VR(100)<>73 THEN GOTO readinit
  GOSUB initax
  SEND(11,1,200,99)
readsp:
  IF NOT(KEY#4) THEN GOTO readsp
  SPEED=VR(1)
  GET#4,VR(100)
  IF VR(100)<>71 THEN GOTO readsp
  SEND(11,1,200,999)
moves:    'Main motion routine
initax:   'Initialisation routine
```

The previous example highlights a couple of useful points:

1) The *Motion Coordinator* that is not controlling the membrane keypad must not transmit anything to or via the keypad until the keypad has been set into network mode by issuing the **SEND** command to give its keypad offset. This example uses a simple form of handshaking by setting variables and sending characters to prevent this situation but another alternative is for the controlling *Motion Coordinator* to issue the **SEND** command on the first line of its program and for all other *Motion Coordinator*s in the network to have a delay at the start of their program to allow time for the membrane keypads to be initialised.

2) When using the **GET#** command it is a good idea to test for the buffer being empty with the **KEY#** command, especially if reading input from a keypad, because it is possible for the buffer to overflow with unpredictable results if it is not emptied by reading from it with the **GET#** command.

Network Specification

Message Format

Each message is constructed of a header character, the message, a cwt. check (on direct variable transfers only) and an end of message character.

Bit	7	6	5	4	3	2	1	0
Header	1	<Address of Receiver>				<Message Type>		
Message (n bytes)	0	<----------- Character Information ---------->						
CRC (byte 0)	0	<--------------- Bits 6..0 of CRC -------------->						
CRC (byte 1)	0	<--------------- Bits 14..8 of CRC ------------>						
End of Message	1	<-Address of Receiver>				0	0	0

Note:
- Address of receiver is in the range 1 (for next node) to 15 (15th node on network). This is different to the node addressing used by Trio BASIC and is for internal use only.
- The number of message bytes (n) is determined by the message type as described below
- CRC bytes are used for direct variable transfers (message type 2)

Network Protocol

The sender places the message on the ring and assumes it arrives at its destination. Any packet checking must be done by the user because the nodes have no knowledge of the size of the net. Each successive node decrements the address by 1 and if the result is greater than zero it re-transmits the message with the decremented address in the header and end of message. If the result is zero then the message is for that node and is processed as necessary. The re-transmission is invisible to the user.

Message Types

The message types described here are for internal use only and are different to those used by the SEND command

0 End of message character.

1 Character transfer. Message can be 1 byte long, e.g. a key press from a keypad, or several bytes long, e.g. PRINT statement.

2 Direct variable transfer. The value of the variable specified is modified directly, without needing to be processed by the user program. Message length is 2 bytes (14 bits) for the variable number and 10 bytes (32 bit integer + 32 bit fraction) for the variable value. Ignored by membrane keypads.

3 Keypad offset. Message is 1 byte long and tells the keypad the number of nodes along the ring to send key presses.

4 Keypad Mode
Message is 1 byte long and controls the response of the keypad to key presses.

0 = Character sent on Key ON only
127 = Key ON/Key OFF each send seperate characters

Network Buffers

The transmit and receive on all nodes is buffered. As far as the user is concerned the buffers in the membrane keypad do not exist as all the necessary housekeeping is performed by the keypad itself. The only concern to the user is the receive buffer on the *Motion Coordinator*. This is 256 bytes long and is used to store the message characters only. The header and end of message bytes are stripped off as they are received so it is not necessary to do this through the Trio BASIC program.

Network Status

The NETSTAT common parameter shows any errors that have occurred on the network since the last time this parameter was cleared. The errors are displayed as follows:

Bit	Value	Error Type	Description
0	1	TX Timeout	Shows that a problem has occurred while trying to put information into the transmit buffer.
1	2	TX Buffer Error	indicates an error on transmission from the buffer. This part of the operation is invisible to the user and if this error ever occurs please contact TRIO for further advice.
2	4	RX CRC Error	the received variable has become corrupt. The error flag is set but the variable isn't.
3	8	RX Frame Error	an incomplete or corrupt message has been received.