▶Trio Motion Technology◀

Trio Motion Technology Ltd.
Shannon Way,
Tewkesbury
Glos. GL20 8ND U.K.

Tel: 01684 292333
Fax: 01684 297929
Email: apps@triomotion.com
Web: www.triomotion.com

Application Note

Doc No.: **TN20-17**
Version: **1.0**
Date: **18 October 2000**
Subject: **Using the CAN keyword to implement a simple network**

Introduction

It is possible to connect a number of controllers together on the CANbus and send data between them by using the CAN(…) command in TRIO BASIC. This application note describes how to implement a simple protocol for exchanging VR() variables between controllers. It is intended as an example of how to communicate only and does not attempt to implement any of the defined CANbus protocols, such as Canopen or Devicenet.

Minimum Requirements

Only Motion Coordinators type MC202, MC204, Euro205 and MC216 have a built-in CANbus port. The program described in this application note cannot be used at the same time as the CAN16I/O or the CAN Analogue modules.

Overview

The example program will cater for between 2 and 5 Motion Coordinators connected together on a single CANbus network. Network nodes are given 4 CAN addresses each to act as channels of communication from the other 4 controllers on the network. No Motion Coordinator is a master but the addressing rules of CANbus mean that the lowest address has the highest priority when two messages collide.

Each Motion Coordinator has the ability to set the value of a VR() variable in any of the other network nodes. Reading of another Motion Coordinator's variables is not supported at this level.

Motion Coordinator CAN addresses

The base addresses for the 5 CANbus nodes are 10, 20, 30, 40 and 50.

Each node sets up 4 receive channels at message numbers 1 to 4 and 4 transmit channels at messages 5 to 8. Each channel provides point to point message passing between two nodes.

Node Base address	Rx #1	Rx #2	Rx #3	Rx #4	
10	10	11	12	13	
20	20	21	22	23	
30	30	31	32	33	
40	40	41	42	43	
50	50	51	52	53	

Node Base address	Tx #5	Tx #6	Tx #7	Tx #8	
10	20	30	40	50	
20	31	41	51	10	
30	42	52	11	21	
40	53	12	22	32	
50	13	23	33	43	

For a node to send a message it must transmit to an address which is set up as a receive message buffer at another node. For example for node 1 to send to node 4 it must place the message in message buffer 7. (address 40) Node 4 then picks up the message from message buffer 1. Similarly for node 5 to send to node 2, it puts the message in buffer 6 (address 23) and node 2 picks up the message from buffer 4.

The following table lists the network's virtual connections in full

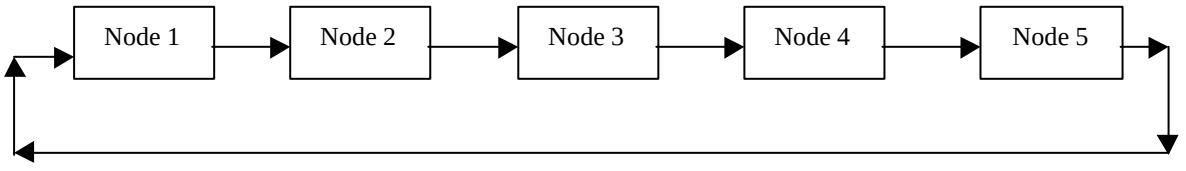
	Received FROM node				Transmit TO node			
Node	Rx #1	Rx #2	Rx #3	Rx #4	Tx #5	Tx #6	Tx #7	Tx #8
1	2	3	4	5	2	3	4	5
2	1	3	4	5	3	4	5	1
3	1	2	4	5	4	5	1	2
4	1	2	3	5	5	1	2	3
5	1	2	3	4	1	2	3	4

Program Operation

The example program simplifies the reception of messages by polling all 4 receive buffers and acting on the data received without regard to the source. The user then needs only to write a program which selects the node to transmit to and the variable number to update at that node.

```
data=2.345
n=1             ' send new value to VR(1) in Motion Coordinator at message channel 5
c=5
GOSUB send_vr
```

Note that the actual destination will depend on which node is sending the message. The general rule is that channel 5 sends to the next node in the system and if the sender is node 5 the message goes to node 1. I.e. the channel destination nodes are configured in a "ring".



In the same way, message channel 6 sends to the next node+1, 7 sends to next node+2 and 8 sends to next node+3.

Program Routines

All of the program sub-routines are described here for reference.

Init_can_port:	Sets up the CANbus baud rate and initialises the CAN chip message buffers. (i.e. assigns a CAN address to each buffer)
Send_vr:	Assembles an 8 byte message ready to be sent over the CANbus. The message format is: VR#, SGN, byte1, byte2, byte3, byte4, byte5, byte6. Byte1 to byte4 contain the integer part of the number as 32 bits. Byte5 and byte6 contain the fractional part as 16 bits. (only 4 decimal places are supported)
Get_vr:	Polls the 4 receive buffers, translates the incoming message and places the decoded value in the required VR().
Set_tx_id:	Sets up a transmit message buffer as defined by: VR(200) = message buffer number VR(201) = identifier (CANbus address)
Set_rx_id:	Sets up a transmit message buffer as defined by: VR(200) = message buffer number VR(201) = identifier (CANbus address)
Send_msg:	Sends a message through message buffer set by VR(200) The 8 bytes of the message are taken from VR(201) to VR(208)
Rx_msg:	Poll the message buffer set by VR(210). If a message has been received then place the 8 bytes of the message in VR(211) to VR(218). If no message then return with negative value in VR(210).