Motion Control Products

Application note
CI521 and CI522 Modbus TCP I/O expansion

AN00225

Rev E (EN)

Extend your MicroFlex e190 or MotiFlex e180 digital and analog I/O capabilities via Modbus TCP and the CI521-MODTCP or CI522-MODTCP modules. Increase I/O count or add enhanced functionality such as temperature / thermocouple sensor inputs

Overview

The introduction of the 58xx firmware series for both MicroFlex e190 and MotiFlex e180 servo drives brings with it the inclusion of Modbus TCP client functionality. This allows these already powerful and flexible drives to connect to a variety of Modbus TCP server devices such as barcode scanners, PLCs, HMIs and ABB's own Modbus TCP I/O expansion modules, CI521-MODTCP and CI522-MODTCP.

The drives can connect up to 2 CI521 or CI522 modules, each of which can be fitted with up to 10 further S500 I/O expansion modules which are simply plugged onto the side of the preceeding module.

If the reader is not already familiar with the e190 and e180 drive range it is recommended that they familiarise themselves with the contents of installation manuals 3AXD50000037326 (MicroFlex e190 installation manual) and 3AXD50000019946 (MotiFlex e180 installation manual).

For full details on the CI521 and CI522 modules please refer to the Automation Builder help system. Connection information for the S500 expansion modules is also contained within the Automation Builder help system. These modules can be used together with the following base units:

- TU507-ETH (screw terminal connections)
- TU508-ETH (spring/cage clamp connections)

Prerequisites

For correct operation of the Modbus TCP I/O modules please ensure the following firmware is installed:

MicroFlex e190: 5868 or later
MotiFlex e180: 5811 or later (although 5868 is recommended to take advantage of the latest features)
CI521 / CI522: V3.1.7 or later (the firmware version installed on these modules can be determined by reading Modbus registers 6 and 7 from the device. For example with the module connected to the drive and configured as server device 0 enter…

```
? Hex MODBUSTCP16(0,6)
? Hex MODBUSTCP16(0,7)
```

…at the Workbench command line. These commands should return the values 5603 and 107 for example (56 = ASCII V, followed by the version 03 01 07). If the Modbus I/O base modules need updating please contact your local ABB support office (an AC500 PLC program is available to perform this update in the field but it may be necessary to return them for upgrade).

CI521-MODTCP

This (base) module provides the following I/O points:

- 4 analog inputs (12 bit resolution + sign bit)
- 2 analog outputs (12 bit resolution + sign bit)
- 8 digital inputs (24Vdc, current sinking)
- 8 digital outputs (24Vdc, 0.5A, current sourcing)

The analog inputs are configurable for use in a wide range of modes including 0-10Vdc, +/-10Vdc, 4-20mA, 2 and 3 wire Pt100, Pt1000 and Ni1000 thermocouples.
The analog outputs are configurable for operation in three different modes; +/-10Vdc, 0-20mA and 4-20mA.

The module is provided with rotary switches to set the last octet of its IP address (192.168.0.rotaryswitches). Two Ethernet sockets allow the module to be daisy-chained to additional CI521-MODTCP or CI522-MODTCP modules if the system is fitted with a number of distributed I/O modules/racks.
Status LEDs on the front of the module indicate the general operating status of the module, the Modbus network and the state of each I/O point. The analog input and output levels are indicated via LEDs of varying brightness.

CI522-MODTCP

This (base) module provides the following I/O points:

- 8 digital inputs (24Vdc, current sinking)
- 8 digital outputs (24Vdc, 0.5A, current sourcing)
- 8 configurable I/O (inputs or outputs, 24Vdc current sinking inputs, 24Vdc 0.5A current sourcing outputs)

The module is provided with rotary switches to set the last octet of its IP address (192.168.0.rotaryswitches). Two Ethernet sockets allow the module to be daisy-chained to additional CI521-MODTCP or CI522-MODTCP modules if the system is fitted with a number of distributed I/O modules/racks.
Status LEDs on the front of the module indicate the general operating status of the module, the Modbus network and the state of each I/O point.

S500 I/O modules

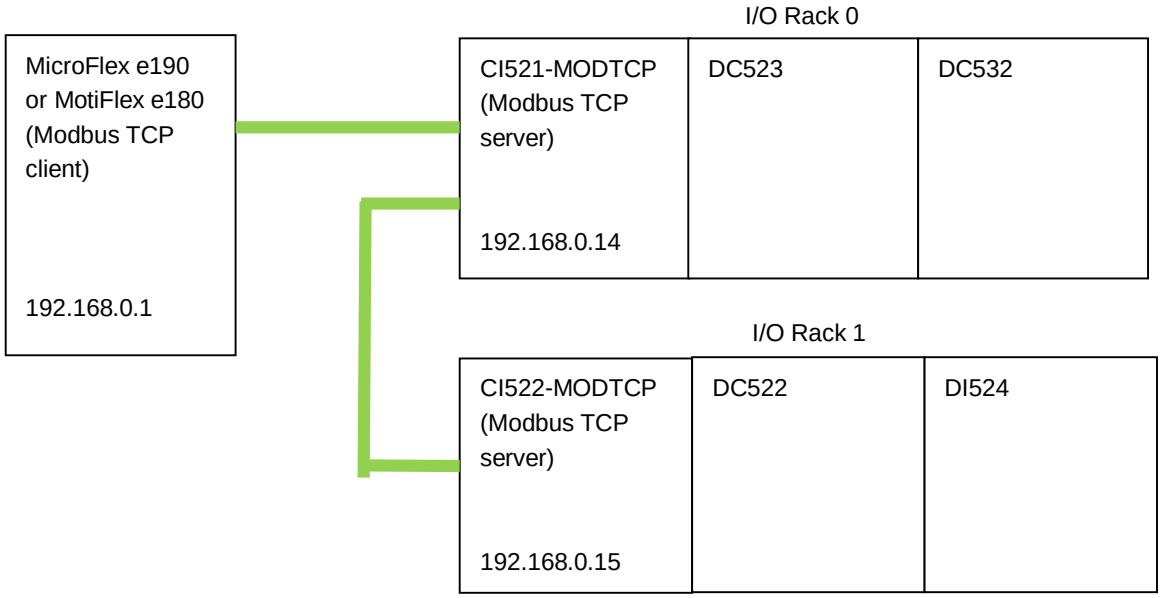
Each Modbus TCP base module can be extended via the use of ABB's S500 range of expansion I/O modules. Up to 10 additional modules can be fitted to each rack (and up to 2 racks can be connected to each drive).

S500 expansion modules supported by the example code provided with this application note are:

- DI524 (32 digital inputs)
- DC522 (16 configurable digital I/O)
- DC523 (24 configurable digital I/O)
- DC532 (16 digital inputs, 16 configurable digital I/O)
- DX522 (8 digital inputs, 8 relay outputs, fast counter *)
- DX531 (8 digital inputs [120/230VAC], 4 relay outputs)
- AX521 (4 configurable analog inputs, 4 configurable analog outputs)
- AX522 (8 configurable analog inputs, 8 configurable analog outputs)
- AI523 (16 configurable analog inputs)
- AO523 (16 configurable analog outputs)
- AI531 (8 configurable analog inputs)

Example I/O configuration

For our example we will configure a remote I/O system that looks like this:

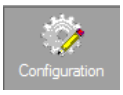


We will daisy chain the Ethernet cables between the drive and the I/O modules making use of the built in switch on the CI521 module, but it would be equally valid to create a star type Ethernet network using an additional switch between the drive and the I/O racks.

Modbus TCP client configuration

To perform configuration of the MicroFlex e190 or MotiFlex e180 drive it is necessary to connect your PC to the drive via Mint Workbench (Build 5814 or later). Connect your PC via the Ethernet socket on the front face of the drive, this socket is reserved for use with standard Ethernet networks. It will probably be necessary to include a standard Ethernet switch to allow all Ethernet devices to be connected at the same time.

Once Workbench is running and the Mint HTTP server has located the drive(s) as part of the scanning process, connect to the required drive (if you are using an Ethernet connection and Workbench does not find your drive please check via the HTTP server tray application that the Discovery check box has been selected for the specific network adaptor being used on your PC – again, refer to the appropriate drive installation manual for further details).

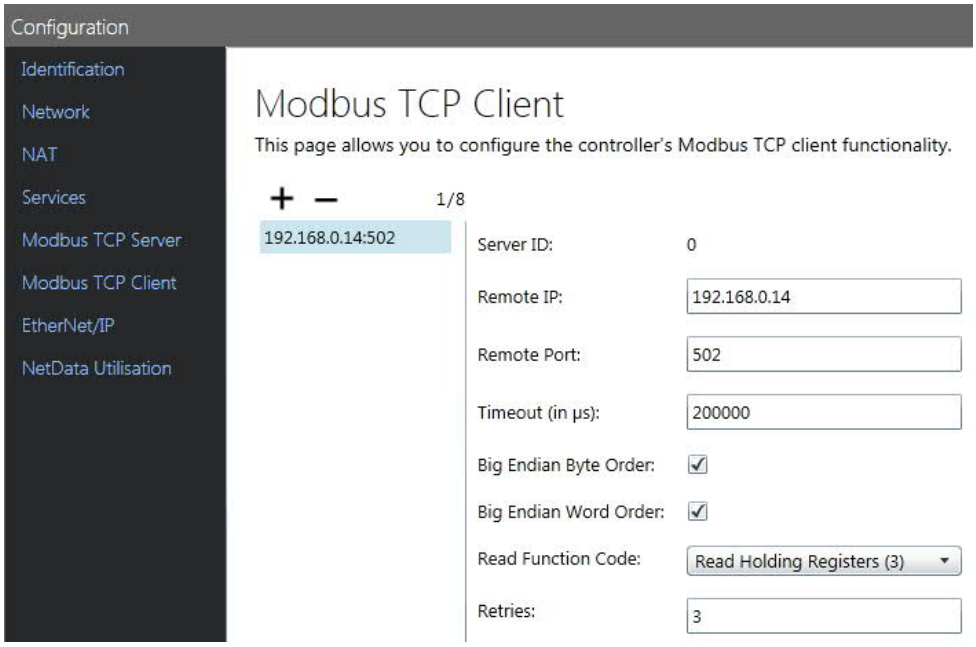
Once online, select the Configuration option from the Toolbox in Workbench. 

When the Configuration screen completes loading select ‘Start New Configuration’ from the list of options. Select the Network screen to view the drive’s IP configuration. By default MicroFlex e190 and MotiFlex e180 drives are set for an address of 192.168.0.1. This can be changed if necessary, but the drive must remain on the same subnet as the I/O modules if the use of an additional router is to be avoided. The Modbus TCP I/O modules use addresses in the range 192.168.0.x (where x is set by two rotary switches on the front of the modules) by default. Again this can be changed if required (using the IPConfig tool built into Automation Builder). If the last octet of the modules’ IP address is set to 0 via IPConfig then the rotary switches set the last digit of the address, otherwise the address set by IPConfig is used. For our example we will leave the modules set for 192.168.0.{switches} and we can therefore leave the drive configured as 192.168.0.1.

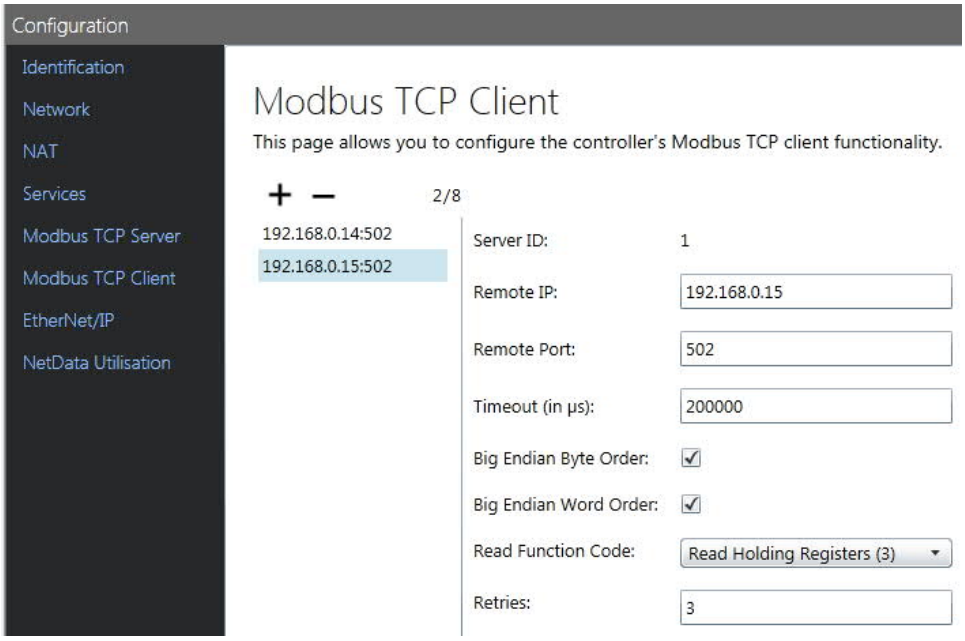
Now select the Modbus TCP client screen and click on the ‘+’ button to add a new server to the configuration. The first server is identified as ‘server ID 0’ and a dialog appears allowing the user to enter the IP address of this device. The server ID (0) must tie up with the logical rack number in the system – in our case logical I/O rack 0 is the rack headed by the CI521 module and so the IP address for server ID 0 is 192.168.0.14 (see the example I/O configuration diagram earlier in this document – 192.168.0.14 is the address because we set the rotary switches on this module to 0E hex).

Modbus TCP uses port number 502 so there is no need to change this setting. Similarly all ABB devices use Big Endian byte and word orders so these two check boxes can be left ticked. If using firmware version 5868 or later it is also possible to set the

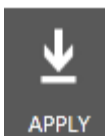
number of automatic retries the client can make in the event of a communication error (here we have set the maximum value of 3):



Click on the '+' button to add the second logical I/O rack to the configuration (this is I/O Rack 1 so the server ID for this rack must also be 1). This rack is headed by the CI522 module and the IP address for this module has been set (via its rotary switches) to 192.168.0.15 so this is the value we enter for the Remote IP address…



Once both I/O racks have been added to the list of servers click the Apply button at the bottom of the screen to save this configuration in the drive



This completes the configuration of the Modbus TCP client operation on the drive.

Mint program configuration

This application note comes complete with an example Mint program containing all the elements needed for any combination of Modbus TCP I/O racks (i.e. 1 or 2 racks, each with up to 10 S500 expansion modules connected to the base CI52x module). The program requires no changes to accommodate IP address changes (this is all handled by the Modbus TCP client configuration detailed in the previous section). It is only necessary to edit the Initialise subroutine within the ModbusIO task to suit the configuration of the I/O racks actually present in the system.

As previously illustrated, our I/O configuration is as follows:

Rack number	0	1
Base Module	CI521	CI522
Expansion 1	DC523	DC522
Expansion 2	DC532	DI524

As a result the code we need to add/edit in the Initialise routine is as follows:

```
'-----------------------------------------------
' Configure the Modbus remote IO racks
' Edit this section to suit the application
'-----------------------------------------------

'First rack (server 0)
Rack(0).Module(0).nType = _nCI521
Rack(0).Module(0).tAnalogs.InMode(0)  = _nPlusMinus10vIn
Rack(0).Module(0).tAnalogs.InMode(1)  = _nPlusMinus10vIn
Rack(0).Module(0).tAnalogs.InMode(2)  = _nPlusMinus10vIn
Rack(0).Module(0).tAnalogs.InMode(3)  = _nPlusMinus10vIn
Rack(0).Module(0).tAnalogs.OutMode(0) = _nPlusMinus10vOut
Rack(0).Module(0).tAnalogs.OutMode(1) = _nPlusMinus10vOut
Rack(0).Module(1).nType = _nDC523
Rack(0).Module(2).nType = _nDC532

'Second rack (server 1)
Rack(1).Module(0).nType = _nCI522
Rack(1).Module(1).nType = _nDC522
Rack(1).Module(2).nType = _nDI524
```

Rack(0) refers to I/O rack 0, Rack(1) refers to I/O rack 1 (and Rack(2) would refer to I/O rack 2 if there was one etc…).
Module(0) refers to the base (CI52x) module for the rack (so in the case of rack 0 this is a CI521 module for example).
Module(1) refers to the 1st S500 expansion module fitted to the base module etc…
In our example we happen to have 2 expansion modules fitted to each base module, this is not a requirement, you can have as many modules as you like fitted to the base module (from 0 to 10).

Pre-defined constants have been included in the program to declare each module type (e.g _nCI521, _nDC532, _nDI524 etc…See the list provided in the Overview section for the full list of possible S500 modules that can be used and refer to the beginning of the ModbusIO task for the corresponding program constants).

For any base or expansion module just comprising digital I/O (e.g. CI522, DC522) it is only necessary to define the module type. For any module (including the CI521 base module) comprising analog I/O it is also necessary to define the input and/or output operating modes for each available analog channel (via tAnalogs.InMode and tAnalogs.OutMode respectively). Again, pre-defined program constants are defined for each possible analog operating mode (in our example the 4 analog inputs on the CI521 are configured for +/-10Vdc operation and the 2 analog outputs are also configured for +/-10Vdc operation).

Advanced Mint program configuration

For a large proportion of applications all that is needed to allow the Modbus TCP I/O to operate is the configuration of the Modbus TCP client settings and the editing / creation of the I/O racks in the Initialise subroutine. However, for some applications it may prove necessary to also edit the setModuleParameters routine to tailor the behaviour of some modules to suit the application requirements.

Possible reasons for needing to edit setModuleParameters are:

- Behaviour of digital outputs in the event of an error with the Modbus TCP module need adjusting (by default the outputs remain in their last state)
- Filter time constant for digital inputs needs adjusting (by default this is set to 0.1ms)
- Operating mode for analog inputs needs to be set to something other than +/-10Vdc (this has to match the configuration set in the Initialise routine). +/-10Vdc is the default mode but most modules support thermocouple inputs, 0-10Vdc etc…
- Configuration of plausibility settings for digital and analog inputs - e.g. checking for short circuits (this is turned off by default)
- A requirement for two modules of the same type to operate in different ways – e.g. there could be two CI521 modules, one where the analog inputs need to operate in +/-10Vdc mode and one where the analog inputs need to be connected to thermocouples. In this case the user should copy the relevant section from the setModuleParameters routine, paste it into the routine for a second time and then create a new module type for this section of code (e.g. Case _nCI521b), remembering to also include a declaration for this new constant and allocating it a unique value. The new case can then be edited to suit the required operation and the Initialise routine can be edited to define the appropriate module type (e.g. if the new CI521b instance is at Rack 7 then the Initialise routine would include Rack(7).Module(0).nType = _nCI521b)

The setModuleParameters includes comments to highlight where edits may be necessary and illustrates the possible values that can be used.

Example:

```
'-------------------------------------------------------------------------------------------------------------
' Set Supervision timeout > 00 to allow outputs to automatically switch off or to predefined state in the
' event of communication loss. Timeout is in 10ms units (e.g. 64 hex = 100 x 10 = 1000ms)
'-------------------------------------------------------------------------------------------------------------
MODBUSTCP16(nModbusServer, 0x3012) = Int16(0x0000)     'Master IP 7 byte 3 (byte) ; Supervision timeout (byte)
```

If we wanted to change the supervision timeout so digital outputs turned off if there was a network error or break in Modbus communication for more than 1s we would edit the code to read…

```
MODBUSTCP16(nModbusServer, 0x3012) = Int16(0x0064)     'Master IP 7 byte 3 (byte) ; Supervision timeout (byte)
```

Using the I/O in the Mint application

The Mint program sample included with this application note provides a task that continually cycles and reads the remote I/O physical input image into an array of data structures. Similarly, this task reads the local output image from the array of data structures and writes this to the physical remote I/O racks.

The priority of this task can be adjusted if necessary once it has been embedded into the main application code using the TASKPRIORITY keyword, but in most cases it is expected that the default priority should provide adequate performance.

Digital I/O is accessed by addressing the appropriate element of the remote rack array. It is best illustrated with some examples. In our example application (see earlier diagram and table) we defined that a DC532 (16 digital inputs and 16 configurable digital I/O) module was fitted as module 2 on rack 0. Therefore to access input 5 on this module Mint code in any of our other Mint tasks in the application would use code of the form…

```
x = ModbusIO::Rack(0).Module(2).bfDigInputs.Bit5
```

Channel.

Digital I/O type:
bfDigInputs
bfDigOutputs

Module no.

Rack/Server ID

Task name (always ModbusIO::)

As another example, to turn on output C19 on this same module the Mint code would be of the form:

```
ModbusIO::Rack(0).Module(2).bfDigOutputs.Bit19 = _On
```

The Mint DEFINE keyword can of course be used to give more meaningful names to I/O channels if needed, for example…

```
Define opRUN_LAMP = ModbusIO::Rack(0).Module(2).bfDigOutputs.Bit19
```

…and then the Mint code to use this channel could be as simple as…

```
opRUN_LAMP = _On
```

Analog I/O is addressed in a very similar fashion, the user should be aware that all voltage/current/50k resistance I/O is scaled to operate as a percentage (in the same way standard Mint controller analog I/O operates). Temperature inputs are scaled to read in degrees C directly.

In our example application a CI521 module is used as the base module for rack 0. This has 4 analog inputs (channels 0 to 3) and 2 analog outputs (channels 0 and 1). To read analog input channel 1 from this rack/module we would use code of the form…

```
x = ModbusIO::Rack(0).Module(0).tAnalogs.InValue(1)
```

Analog channel.

Access type:
InValue
Outvalue

Module no.

Rack/Server ID

Task name (always ModbusIO::)

Analog identifier (always tAnalogs)

…and to write to analog output 0 on this rack/module (e.g. to set 52.3%) we would use code of the form…

```
ModbusIO::Rack(0).Module(0).tAnalogs.OutValue(0) = 52.3
```

Other program features

The example program assumes the user would want to initialise the Modbus TCP I/O racks as part of the Mint Startup module (i.e. before the main Mint program executes). Because there is no ONERROR event associated with the Startup module the example application makes use of the Mint Shutdown block to detect errors in the Startup routine. The Shutdown code detects if the error occurred during the Modbus I/O initialisation, and if it did it attempts to restart the program 10 times before finally giving up and letting the program terminate.
This is necessary for instances where the drive has been communicating with the Modbus TCP I/O and is then reset independently. If the drive reboots in less time than is needed for the previous Modbus connection to close an error may occur so the code provides a retry mechanism that lasts at least as long as the maximum connection timeout (30 seconds).

Note also that if the Modbus I/O racks are power cycled independently then it may eventually become necessary to wait for the drive to reset its TCP connections (approximately 2 minute timeout).

The ParentTask also includes a line of code that ensure the ModbusIO task is only executed if racks and modules have been defined in the Initialise routine. In most cases the example code wouldn't be used if there was no remote I/O at all, but this check is included should the user want to include all the code anyway in readiness for addition of expansion I/O modules at some point in the future.

```
If ModbusIO::nNoOfRacks > 0 Then Run(ModbusIO)
```

If using the supervision timeout to instigate the digital / analog output behaviour under a fault condition (e.g. turn off the outputs automatically) be aware that the modules will time out on any delay in communication, not just a physical break in the connection. So for example, holding up execution of the ModbusIO task because the Mint program has jumped to (and is sat in) a Mint event or downloading a new Mint program could result in the outputs being forced to their "fault behaviour" state.

Contact us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC