SUBJECT: SMC PRODUCT TO WAGO I/O CONNECTIVITY
Product: SMC based products (SMC3010, SMC4000)

Summary:
This document describes how to setup and configure a distributed I/O system using the SMC products that feature Ethernet ports and products from Wago. Both an Ethernet coupler (non-intelligent) and an Ethernet Programmable Fieldbus Controller (intelligent) I/O device are used. Timing differences between direct I/O vs. Ethernet I/O are measured. A PLS-type application is configured with the PFC.

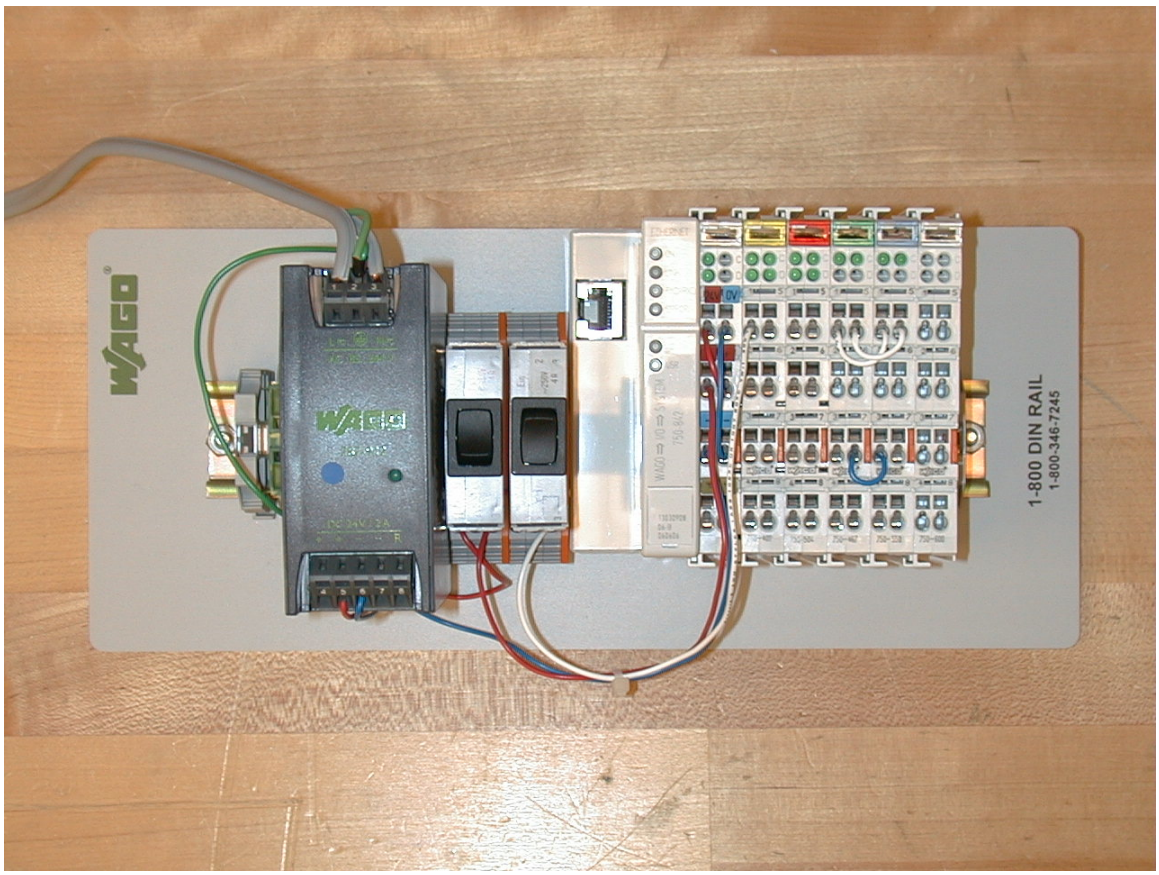
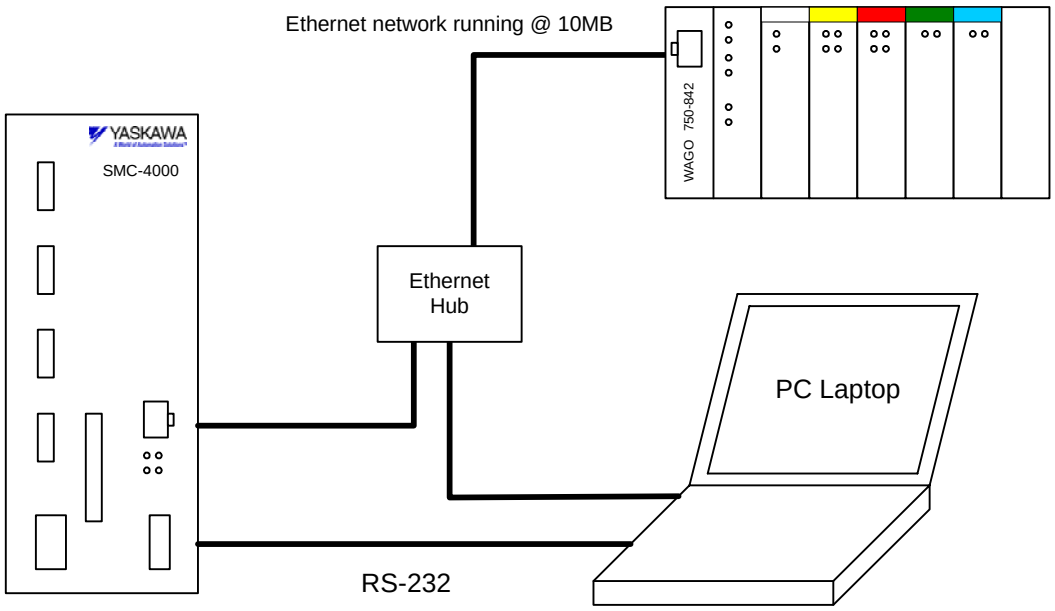


Table Of Contents:

THE APPLICATION:

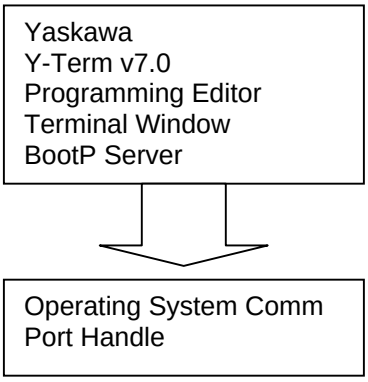
The SMC controller features connections to direct on-board I/O, Ethernet distributed I/O, and an Ethernet Programmable Fieldbus Controller (PFC). The response differences between the direct vs. Ethernet I/O is measured by an oscilloscope. The SMC will send motor position information continuously to the PFC which will run its own ladder logic program to perform a PLS-type operation. In this manner, 1 of 8 outputs will be assigned to a 45° quadrant.

HARDWARE LAYOUT:



SOFTWARE LAYOUT:

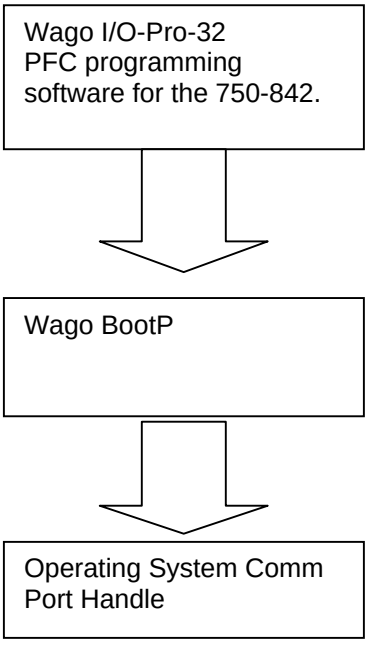
Programming the SMC product



Y-term is used to program all of the Yaskawa SMC products.

Y-term can connect to multiple SMCs over RS232 or Ethernet. Initial IP address is set via serial communication or Y-term BootP. After that, all communication can be via Ethernet.

Programming the Wago Ethernet Coupler



I/O-Pro-32 is used to program the Wago 842 PFC. It conforms to IEC-61131. Programs can be developed in Ladder, Structured text, Instruction list, Sequential Function Chart or Function Block. This is not used for the –342 Ethernet coupler since there is no logic processor.

BootP is a program that configures the IP address of a Wago Ethernet coupler based on its MAC address.

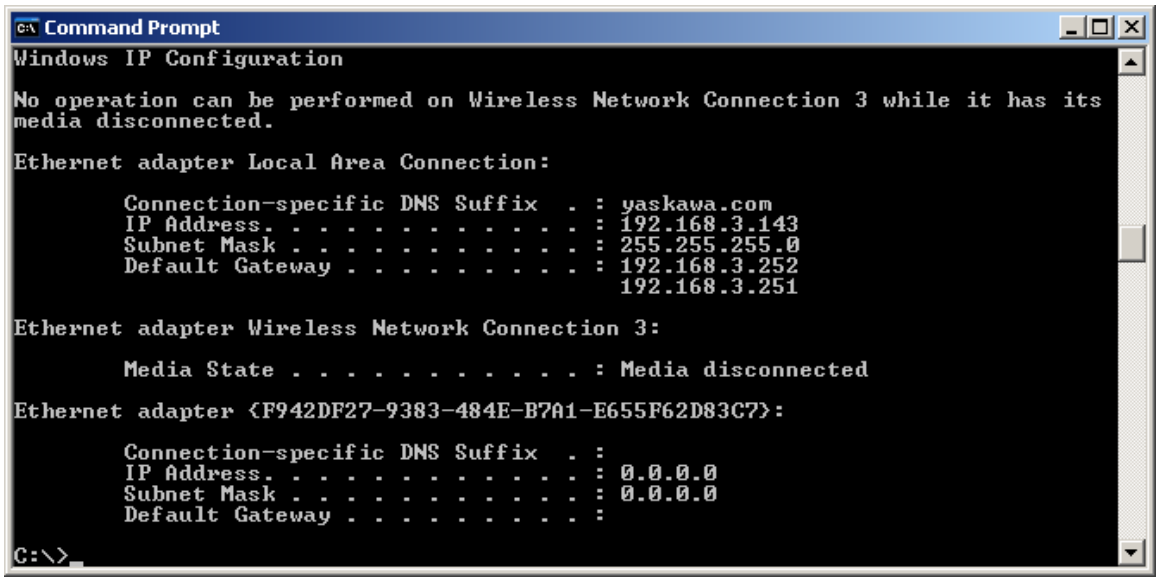
SETTING UP THE APPLICATION:

Below is a high-level checklist of the steps necessary to set up the application a more detailed description follows.

1. Configure the SMC IP address using Y-Term
2. Configure the Wago Ethernet Coupler IP address using Wago BootP
3. Wago I/O configuration and addressing.
4. Program the SMC to establish an Ethernet connection to the Wago I/O
5. Program the SMC to directly manipulate the distributed I/O
6. Program the Wago PFC to perform a PLS function based on motor position.

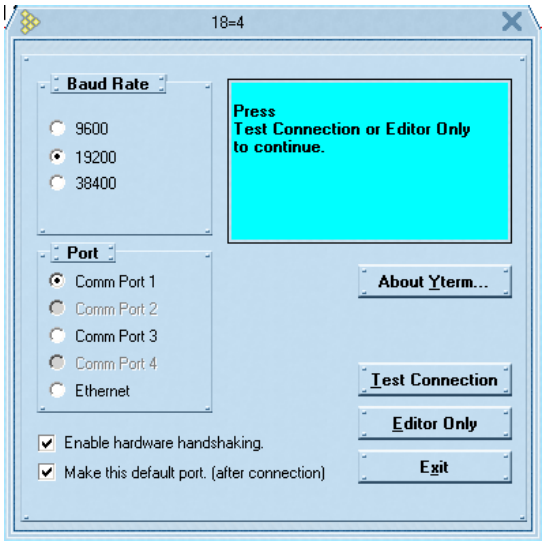
Configure SMC IP address using Y-Term:

First, the IP address of the computer needs to be determined in order to define an appropriate IP address for the SMC. If the computer IP address is unknown, it can be displayed by entering c:\ipconfig using the command prompt:



An IP address for the SMC must contain the same address in the first 3 IP fields and a unique address in the 4th.

Choose Serial port and Click Test Connection.



Once serial communication is established, click on the Terminal Tab and enter the IP address with the IA command.

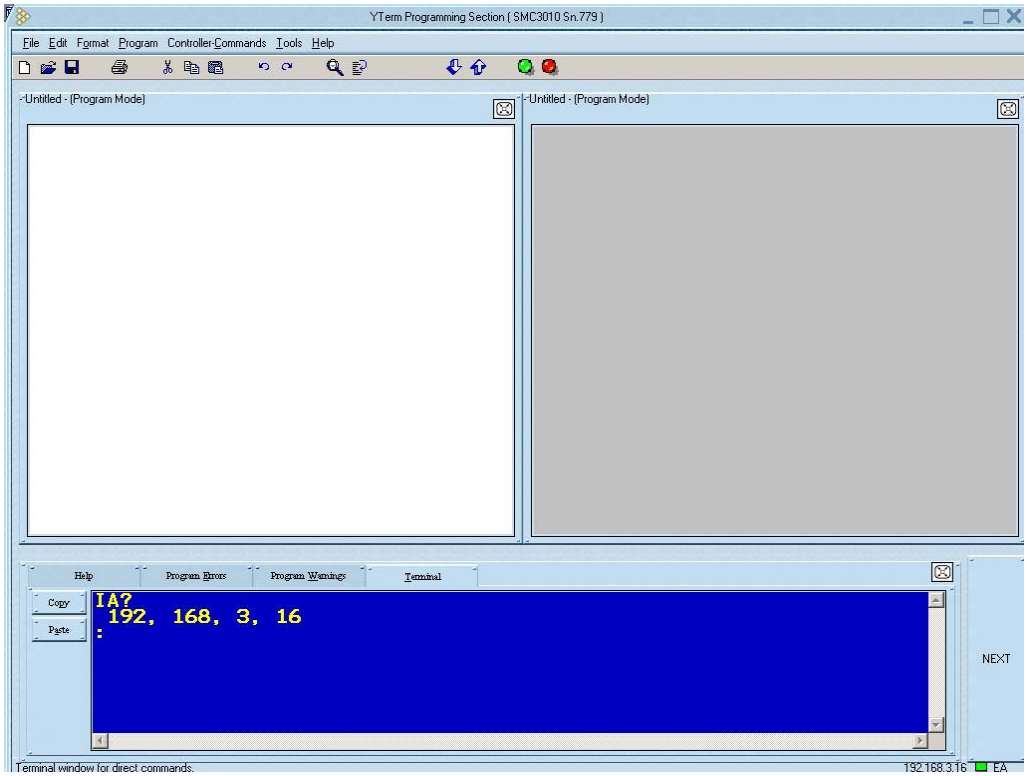
Ex: **IA 192,168,3,16**

Then issue the command to burn the address to non-volatile memory.

Ex: **BN**

The IP address setting can always be interrogated by the command

: **IA?**



Issue the reset command

:**RS**

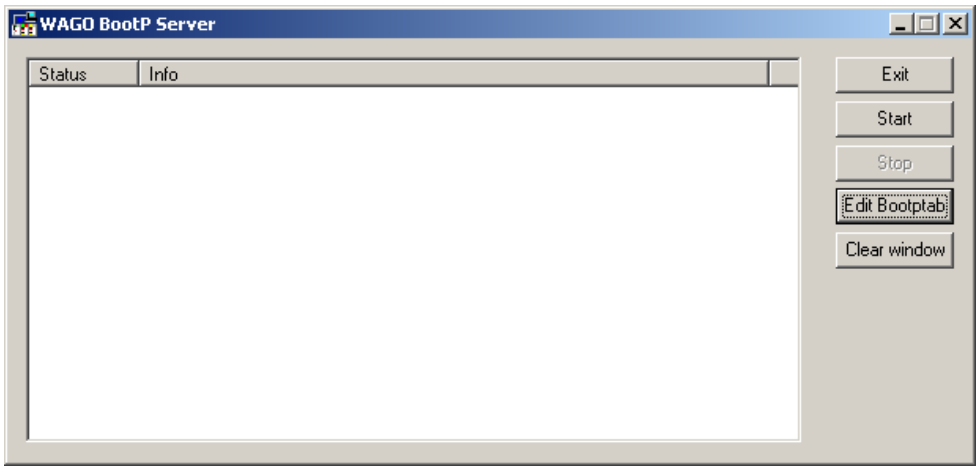
Close Y-Term and re-launch. This time, choose Ethernet connection and enter the IP address you just configured.

You should now be able to connect to the SMC unit over Ethernet.

Configure Wago IP address using Wago BootP:

Install the BootP program from the PFC Tools CD ROM or from web site download.

Once installed, it can be found at Start, Programs, Wago Software, Wago BootP Server.



Click on Edit Bootptab to modify the BootP initialization file. This will allow you to assign an IP address and a network name to a device at a particular MAC address. This procedure will overwrite an existing IP address for the chosen device.

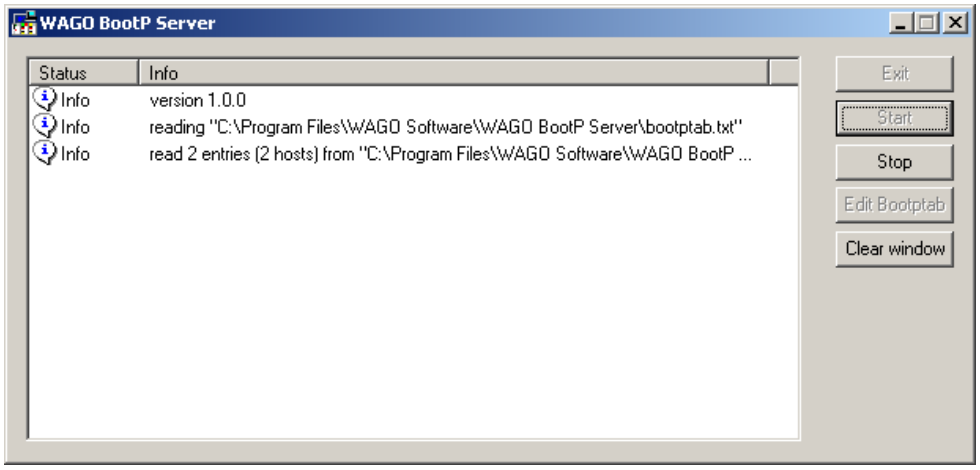
Modify the last line only of the file with the desired information.

'device name':ht=1:ha= 'MAC address':ip= 'IP address':sm= 'subnet mask'

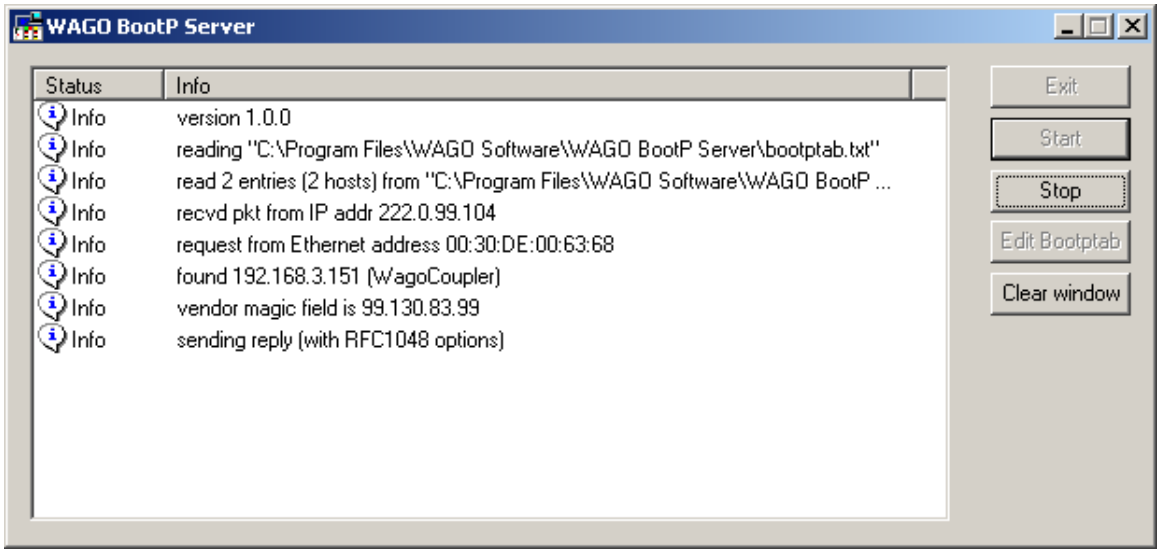
Ex: **WagoCoupler:ht=1:ha=0030DE006368:ip=192.168.3.151:sm=255.255.255.0**

Configures Ethernet coupler with MAC address 00:30:DE:00:63:68, to IP address 192.168.3.151 with subnet mask 255.255.255.0 and names it "WagoCoupler".

Save and close the file and click on the Start Button. The program will prepare the I/O device to re-ARP.



Cycle the power on the I/O device. The IP configuration will be completed.



Wago Configuration and Addressing:

Wago I/O configures and maps itself automatically upon power up. Understanding how the Wago I/O is addressed is the key to proper application.

On power up, the I/O coupler or PFC will address the I/O according to what it finds to be present. Addresses are assigned in the following order:

Analog Inputs
Digital Inputs
Analog Outputs
Digital Outputs

The coupler follows this hierarchy no matter what order the modules are physically arranged in the rack.

Each analog channel consumes 1 word of memory. Each digital channel consumes 1 bit.

	PFC Word Addressing	PFC Byte Addressing	PFC Bit Addressing	Memory Map Usage	Network/PFC Accessibility
0	%IW0	%IB0, %IB1	%IX0.0 to %IX0.15		
	%IW1	%IB2, %IB3	%IX1.0 to %IX1.15	Real World	Network Access
	%IW2	%IB4, %IB5	%IX2.0 to %IX2.15	Analog Inputs	is Read Only
	.	.	.	&	
	thru	thru	thru	Real World	PFC Access
	.	.	.	Digital Inputs	is Read Only
255	%IW255	%IB510, %IB511	%IX255.0 to %IX255.15		
256	%QW256	%QB512, %QB513	%QX256.0 to %QX256.15		
	%QW257	%QB514, %QB515	%QX257.0 to %QX257.15	Network Variable	Network Access
	%QW258	%QB516, %QB517	%QX258.0 to %QX258.15	Memory	is Read Only
	.	.	.		
	thru	thru	thru	(PFC to Network)	PFC Access
	.	.	.		is Read/Write
511	%QW511	%QB1023, %QB1024	%QX511.0 to %QX511.15		
512	%QW0	%QB0, %QB1	%QX0.0 to %QX0.15		
	%QW1	%QB2, %QB3	%QX1.0 to %QX1.15	Real World	Network Access
	%QW2	%QB4, %QB5	%QX2.0 to %QX2.15	Analog Outputs	is Read/Write
	.	.	.	&	
	thru	thru	thru	Real World	PFC Access
	.	.	.	Digital Outputs	is Read/Write
767	%QW255	%QB510, %QB511	%QX255.0 to %QX255.15		
768	%IW256	%IB512, %IB513	%IX256.0 to %IX256.15		
	%IW257	%IB514, %IB515	%IX257.0 to %IX257.15	Network Variable	Network Access
	%IW258	%IB516, %IB517	%IX258.0 to %IX258.15	Memory	is Read/Write
	.	.	.		
	thru	thru	thru	(Network to PFC)	PFC Access
	.	.	.		is Read Only
1023	%IW511	%IB1023, %IB1024	%IX511.0 to %IX511.15		

For the 750-842 Programmable Fieldbus Coupler, the physical Inputs are mapped starting at Word 0. The physical Outputs are mapped starting at Word 512. Variable memory that the logic processor can use are mapped starting at Word 256. And finally, variable memory registers that an external device (like the SMC) can write to begin at Word 768.

Opening SMC Ethernet connections to a Wago I/O block:

Ethernet connections with the SMC products are called “Handles”. The SMC-3010 (also called the Legend-MC) can open up to 16 handles at one time. The SMC-4000 can open 8. Handles are opened with the ‘IH’ command.

IH

FUNCTION: Open Internet Handle

Description:

The IH command is used when the SMC product is operated as a network master. This command opens a handle and connects to a slave.

Each controller may have several handles open at any given time. They are designated by the letters A through P. To open a handle, the user must specify:

The IP address of the slave

The type of session TCP/IP or UDP/IP

The port number of the slave. TCP protocol uses port 502.

Arguments: IHh = ip0, ip1, ip2, ip3 <p >q OR IHh = n <p >q OR IHh = >r Where:

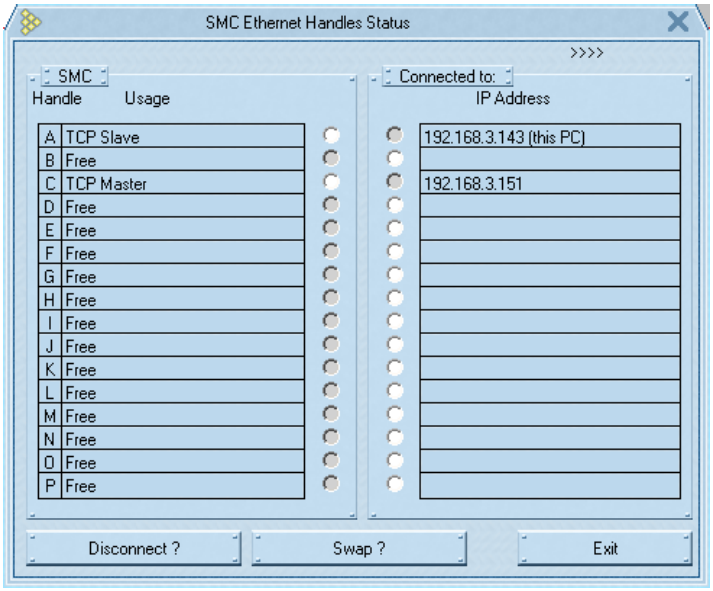
Argument	Minimum	Maximum	Note
h	A	H	Internet handle
ip0 - ip3	0	255	Four bytes of IP address separated by commas
n	-2147483648	2147483647	32 bit address alternative to ip0 - ip3
S=>C	-1 (UDP)	-2 (TCP)	Close the handle that sent the command
T=>C	-1 (UDP)	-2 (TCP)	Close handles except the one sending the command
<p	0	65535	Specifies the port number of the slave, not required for opening a handle
>q	0	2	Set connection type; 0=none; 1=UDP; 2=TCP
>r	-1 (UDP)	-2 (TCP)	Terminate connection, and handle to be freed
?			Returns the IP address as four 1 byte numbers

For example:

To open a TCP/IP connection to the above configured Wago I/O block using Handle C, issue the command:

IHC = 192,168,3,151 <502 >2

The Ethernet handle connections are viewed by going under the menu Tools, Ethernet Status.



The IH command also contains status variables _IHh1 through _IHh4 where 'h' is the handle letter. These are useful for programmatically checking the status of the handle when implementing proper programming technique.

_IHh0 contains the IP address as a 32 bit number

_IHh1 contains the slave port number

_IHh2 contains a 0 if the handle is free

contains a 1 if it is for a UDP slave

contains a 2 if it is for a TCP slave

contains a -1 if it is for a UDP master

contains a -2 if it is for a TCP master

contains a -5 if attempting to connect by UDP

contains a -6 if attempting to connect by TCP

contains a 1 if it is for a UDP slave

_IHh3 contains a 0 if the ARP was successful

contains a 1 if it has failed or is still in progress

_IHh4 contains a 1 if the SA command is waiting for acknowledgement from a slave

contains a 2 if the SA command received a colon

contains a 3 if the SA command received a question mark

contains a 4 if the SA command timed out

Programming the SMC using I/O set, read and Modbus commands:

Distributed Digital I/O is extremely easy to accomplish with the SMC products. Digital outputs are controlled using the already familiar SB and CB commands. The Wago TCP/IP connection implements Modbus protocol.

SB

FUNCTION: Set Bit

Description:

The SB command sets a bit on the output port, slave controller or Modbus I/O

When using Modbus devices, the I/O points are calculated using the following formula:

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (BitNum-1)

Slave address is used when the Modbus device has slave devices connected to it, and specified as Addresses 0 to 255. Please note that the use of slave devices for Modbus are very rare and this number will usually be 0.

HandleNum is the handle specifier from A-P (1-16)

Module is the position of the module in the rack from 1 to 16

BitNum is the point in the I/O module from 1 to 4

Arguments: SB n

CB

FUNCTION: Clear Bit

Description:

The SB command clears a bit on the output port, slave controller or Modbus I/O

Same addressing as SB command

Arguments: CB n

EX: SB 3000 will turn on the first digital output on the above I/O device using Handle C.
CB 3000 will turn it off.

Distributed Digital input status is read with the @IN function just as it is with local inputs.

@IN

FUNCTION: Status of Digital Input

Description:

@IN returns the status of the digital input number or variable given in square brackets. Note that the @IN command is a *function* and does not follow the convention of the commands, and does not require the underscore when used as an operand.

When using Modbus devices, the I/O points are calculated using the following formula:

n = (SlaveAddress*1000) + (HandleNum*1000) + ((Module-1)*4) + (BitNum-1)

Slave address is used when the Modbus device has slave devices connected to it, and specified as Addresses 0 to 255. Please note that the use of slave devices for Modbus are very rare and this number will usually be 0.

HandleNum is the handle specifier from A-P (1-16)

Module is the position of the module in the rack from 1 to 16

BitNum is the point in the I/O module from 1 to 4

Arguments: @IN [n]

```
EX:   IF (@IN[3000])          //If the 1st digital input of the above I/O device is ON
        MG {EA} @IN[3000]     //print the status to the screen

      ENDIF
```

Analog I/O or Memory variables are implemented using the Modbus command to read and/or write registers (words) in the distributed device.

MB

FUNCTION: Modbus

Description:

The MB command is used to communicate to devices using the first 2 levels of the Modbus protocol.

The format of the command varies depending on each function code. The function code, -1, designates that the first level of Modbus is used (creates raw packets and receives raw data). The other codes are the 10 major function codesof the second level that the SMC products support.

Note: All formats contain an 'h' parameter. This designates the handle connection number.

Arguments: MBh = a, Function, …

Function	Meaning	Example
-1	Raw Packets	MBh = -1, y, array [ ]
1	Read Coil Status	MBh = a, 1, t, b, array [ ]
2	Read Input Status	MBh = a, 2 , t, b, array [ ]
3	Read Holding Registers	MBh = a, 3, e, r, array [ ]
4	Read Input Registers	MBh = a, 4, e, r, array [ ]
5	Write Single Coil	MBh = a, 5, t, c
6	Write Single Register	MBh = a, 6,g, s
7	Read Exception Status	MBh = a, 7, array [ ]
15	Write Multiple Coils	MBh = a, 15, t, b, array [ ]
16	Write Multiple Registers	MBh = a, 16, e, r, array [ ]
17	Report Slave ID	MBh = a, 17, array [ ]

Argument	Description	Argument	Description
a	Slave address	h	Connection handle number
array [ ]	Name of array containing data	r	Number of registers
b	Number of bits	s	16 bit value
c	0 or 1 (to turn coil OFF or ON)	t	Starting bit number
e	Starting register	y	Number of bytes
g	Register number		

EX: **MBC = 3,6,512, 1024**

This command will write a value of 1024 into the first analog output of the Wago I/O device on Handle C

EX:
```
DM WagoIn[2]              //Dimension an array named 'WagoIn" with 2 elements
MBC = 3,4,0,1,WagoIn[0]   //Read the input register at address 0
MG {EA} WagoIn[0]         //Display the value to the screen
```

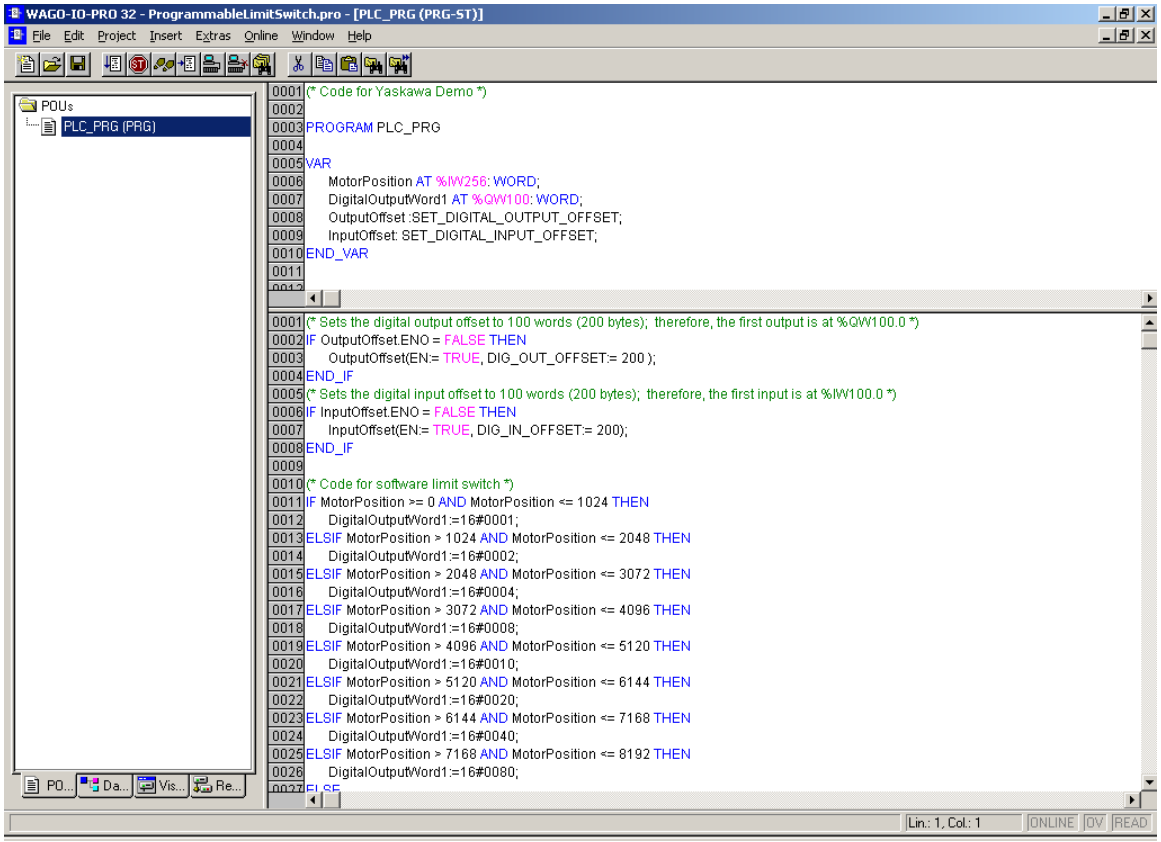
This command will read the value of the first analog input of the Wago I/O device. Analog output #1 has been wired to Analog Input #1. These values should be close to the same.

Programming the Wago PFC using Wago-I/O-PRO 32:

Wago-I/O-PRO 32 is used to program the 750-842 Ethernet Programmable Fieldbus Controller. This unit features not only the distributed I/O modules, but also a local logic processor. Note that the 750-841 is a similar PFC device, but is programmed with a different software package.

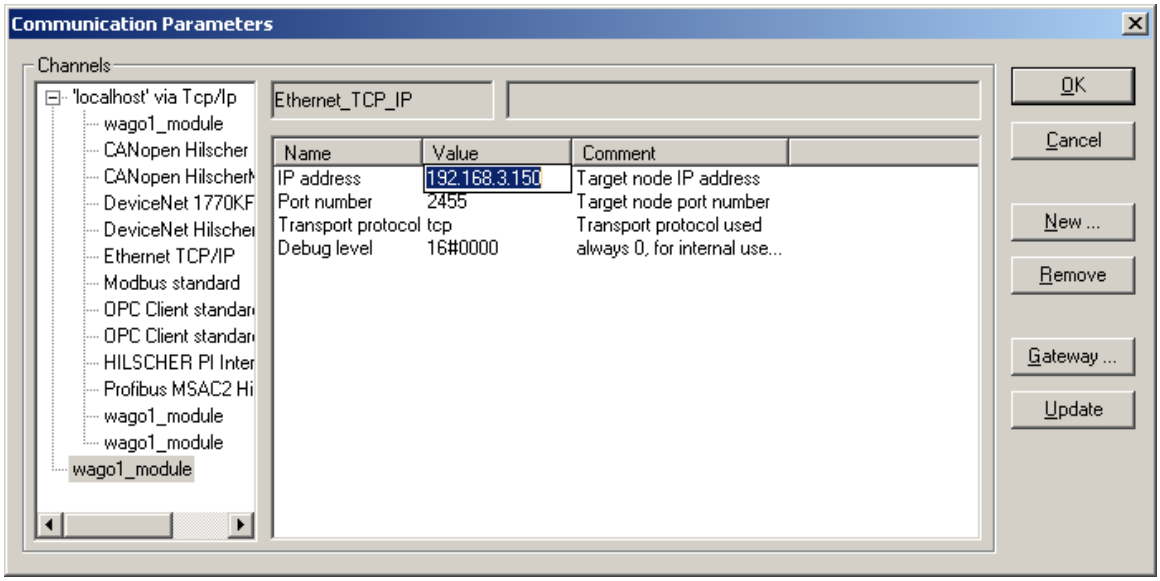
After Installation, start the program by clicking Start, Programs, Wago-I/O-PRO 32, Wago-I/O-PRO 32.

For this example, open the PLS program by choosing File, Open and locate "ProgrammableLimitSwitch.pro" The code listing, written in Structured Text will be displayed.

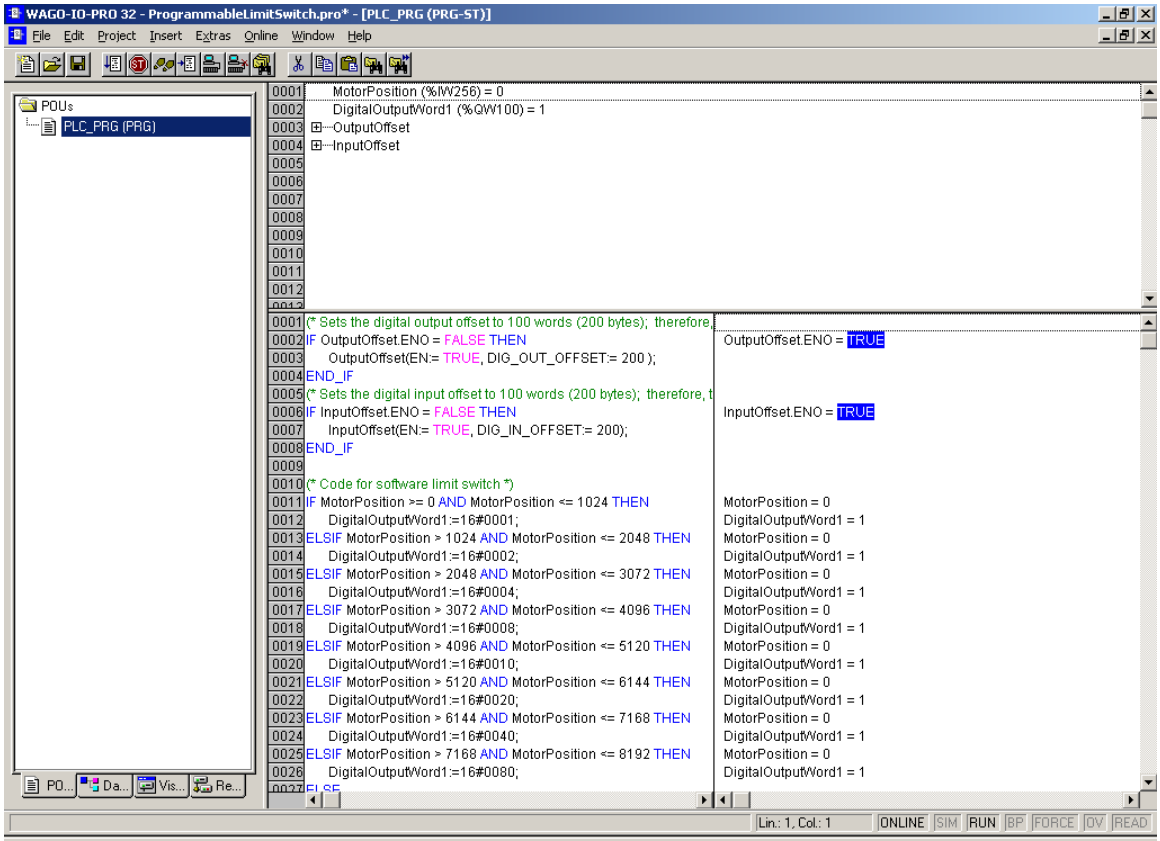


To connect to a particular PFC at a known IP address, choose Online, Communication Parameters…

Highlight and alter the TCP/IP address of the device. Don't forget to hit <enter> to accept the changes. Then click OK.



Next, go Online with the unit by choosing Online, Login. Start the logic processor by choosing Online, Run.



Note that the screen has changed and now shows actual program states and values.

For our PLS example, download the file “SMC-Wago Connectivity.smc” to the SMC controller using Y-Term. Run the PLS program by typing:

XQ #SETUP

Observe the status of the Wago PFC.

Keep in mind that if the PFC is used, the SMC would not normally write directly to the outputs. This could cause a conflict with the 2 logic programs trying to control the same I/O. The more common method would be to write to a memory register as demonstrated, and let the Wago PFC take control from there.

APPENDIX:

SMC Example Code Listing:

This program contains basic routines to demonstrate connectivity between the SMC products and the Wago Ethernet I/O.

#SETUP	- sets initial parameters and starts the homing routine
#RELEASE	- releases the Ethernet handles if they are open
#ASSIGN	- opens Ethernet handles
#INIT	- homes the motor to the C-Pulse
#OUTLOOP	- sequences On and Off the digital outputs on an –842 and –342 coupler
#DIO-TIM	- measures execution time in servo updates of direct I/O response
#EIO-TIM	- measures execution time in servo updates of Ethernet I/O response
#PLS	- jogs the motor and sends motor position info out to the PFC
#IOLOOP	- sets both direct and Ethernet outputs for response measurement with scope

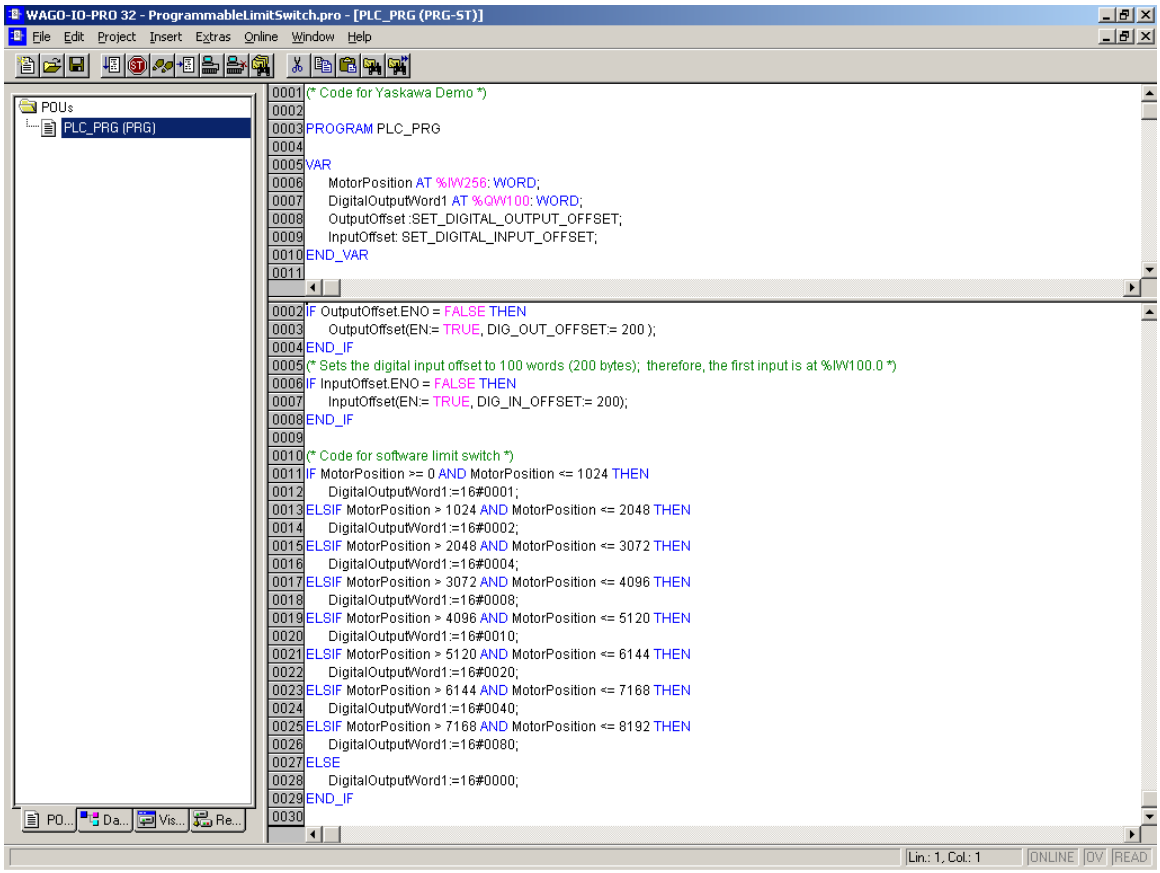
```
//---------------------------------------
#SETUP
 DM WAGO[20]
 UDP=1
 TCP=2
 UDPCONN=-5
 TCPCONN=-6
 ON = 0
 OFF = 1
 MW 1

 //Tuning Parameters
 KP 1.4
 KD 16
 KI 0

 //Enable the servo
 SH X

 //Scale factors
 mres = 8192
```

```
 JS #RELEASE
 JS #ASSIGN
 JS #INIT

EN

#RELEASE
 MG {EA} "RELEASING HANDLES..."
 IF(_IHB2<>0)
  #REL_B
  IHB=>@ABS[_IHB2]*-1
  WT 1000
  JP #REL_B, _IHB2<>0
 ENDIF

 IF(_IHC2<>0)
  #REL_C
  IHC=>@ABS[_IHC2]*-1
  WT 1000
  JP #REL_C, _IHC2<>0
 ENDIF
 MG {EA} "RELEASE DONE"

EN

#ASSIGN
 #ASSIGNB
  MG {EA} "ASSIGNING HANDLES..."
  IHB= 192,168,3,150 <502 >TCP    //PROGRAMMABLE FIELD COUPLER
  #ASGN_B;WT 10; JP #ASGN_B, _IHB2=TCPCONN; JP #ASSIGNB, _IHB2<>-TCP
  MG {EA} "HANDLE B ASSIGNED TCP"
 #ASSIGNC
  IHC= 192,168,3,151 <502 >TCP    //ETHERNET TCP/IP COUPLER
  #ASGN_C;WT 10; JP #ASGN_C, _IHC2=TCPCONN; JP #ASSIGNC, _IHC2<>-TCP
  MG {EA} "HANDLE C ASSIGNED TCP"
 MG {EA} "ASSIGN DONE"
EN

#INIT
 MG {EA} "Homing the motor..."

 HX 1               //Halt the PLS Task (thread)
 JG 1000
 FI X
 BG X
 AM X
 MG {EA} "Homing Complete"

 XQ #PLS, 1

EN
```

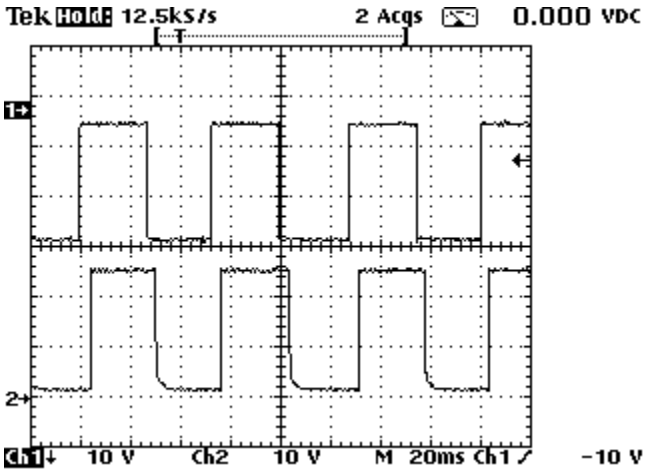
```
#OUTLOOP
 x=2000
 #OLOOP1
  SB x
  WT 250
  x=x+1
 JP #OLOOP1, x<=2007
 x=3000
 #OLOOP2
  SB x
  WT 250
  x=x+1
 JP #OLOOP2, x<=3003

 WT 500

 x=2000
 #OLOOP3
  CB x
  WT 250
  x=x+1
 JP #OLOOP3, x<=2007
 x=3000
 #OLOOP4
  CB x
  WT 250
  x=x+1
 JP #OLOOP4, x<=3003

EN


#DIO_TIM
 dstartt = TIME
 SB 3
 #WT_DI; JP #WT_DI, @IN[3]<>ON
 dstopt = TIME
 dtdiff = (dstopt-dstartt)*1000/1024

 MG {EA} "DELAY TIME = ", dtdiff

 CB 3

EN


#EIO_TIM
 estartt = TIME
 SB 2000
 #WT_EI; JP #WT_EI, @IN[2000]<>ON
 estopt = TIME
 etdiff = (estopt-estartt)*1000/1024

 MG {EA} "DELAY TIME = ", etdiff

 CB 2000

EN
```

```
//PLS Operation
#PLS
 //Start moving the motor
 JG 2*mres
 BG X

 //Send motor position out to the Wago PFC, this also adds
 //speed compensation by sending an advanced motor position
 //based on the motor velocity
 #SENDPOS
   SB3
   MBB=2,6,768,@FRAC[_TPX+(_TVX/200)/mres]*mres
   CB3
 JP #SENDPOS

EN

#IOLOOP
 SB3;SB2000;SB3000
 WT 20
 CB3;CB 2000;CB3000
 WT 20
 JP #IOLOOP

EN
```

Wago PFC Example Code Listing:



Oscilloscope Traces:

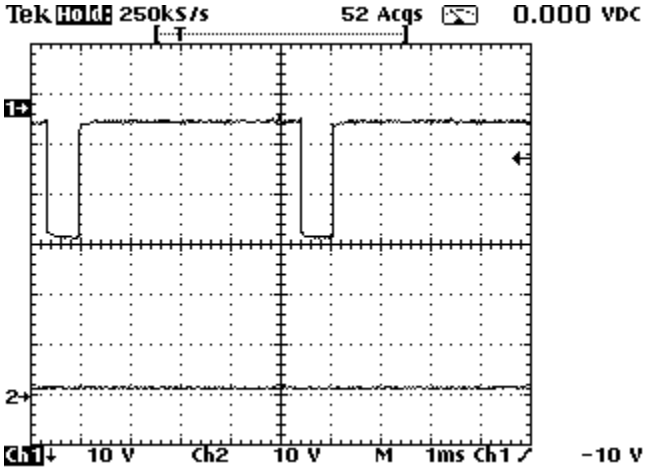


CH1 is Direct Output from SMC

CH2 is Ethernet output on Wago

SMC executing the #IOLOOP routine.

There is only about a 4 msec lag on the Ethernet outputs.

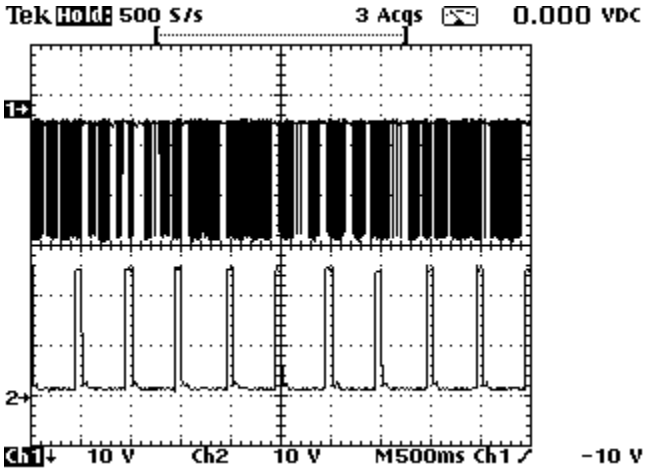


CH1 is Direct Output from SMC

CH2 is Ethernet output on Wago

SMC executing the #PLS routine

This identifies the loop time of the sample program used to output the motor position to the Wago PFC. From this we see that it takes about 5 msec to execute the Modbus Write (6) command.

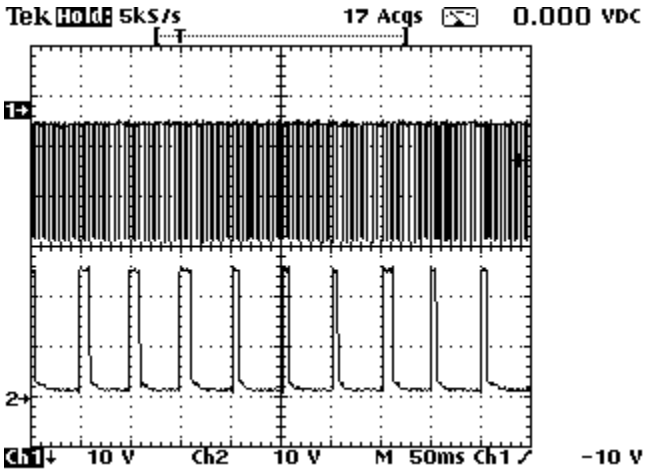


CH1 is Direct Output from SMC

CH2 is Ethernet output on Wago

SMC executing the #PLS routine

This shows the result of the PLS program with the motor running at 2 rev/sec. The SMC feeds the motor position to the Wago PFC. The Wago then executes its logic program to activate the outputs.



CH1 is Direct Output from SMC

CH2 is Ethernet output on Wago

SMC executing the #PLS routine

This shows the result of the same PLS program except with the motor running at 20 rev/sec. The frequency is very consistent even though the width is starting to vary.