NGM EVO

www.promax.it

VTB Software Resources

Rev. 2.0.0

© Promax s.r.l. – Via Newton, 5/G – Z.I. Malacoda – CastelFiorentino (Fi) ITALY

email:info@promax.it - internet:www.promax.it

1 Preface

This document is refered to NGMEVO Board HARDWARE resources usable with VTB language
For more details to VTB language, see the following links:

Programming Guide
Objects Guide

The following examples, are not refered to real applications

2 Ethernet Port

The TCP/IP STACK is managed, by operating system. The management protocols that using TCP/IP, is delegated to VTB APPLICATION. For example, the TCP/IP MODBUS, is managed by a OBJECT in VTB language. In the same mode, is possible, management the propretary protocols.

2.1 SET_IP

Sets the parameters of TCP/IP protocol.

Syntax

```
SET_IP(ip as *char, sm as *char, gw as *char) as void
```

Parameters

ip NGMEVO IP Addres
sm Subnet mask
gw Gateway (normally not used)

WARNING
THIS FUNCTION MUST BE CALLED IN THE INIT SECTION OF THE MAIN OR PLC TASK

2.2 PXETH_ADD_PROT

Adds a custom protocol to a specific port of TCP/IP. A custom function to process the new protocol must be written and its pointer must be pass to this function.

Syntax

```
PXETH_ADD_PROT(port as long, fun as delegate) as void
```

Parameters

port TCP port on which the new protocol is added
fun Pointer to the custom process function

2.2.1 PROTOCOL PROCESS FUNCTION

This function isn't defined by system but it must be written in the application. The system will call this function, by the pointer passed with ***pxeth_add_prot,*** each time a data packet is received from the port associated to this protocol. To read the received data the function **pxeth_rx** have to be call while to send the response data they must be written in the transmit buffer (buftx) and return from the function the number of bytes we want to send.

Syntax

```
MY_PROTOCOL(len as long, buftx as *char) as long
```

Parameters

len Length of data packet received

buftx Pointer to the transmit buffer

Return value
long Number of bytes to be send

2.3 PXETH_RX

Read a single byte from the TCP/IP receive buffer. It is called by the protocol process function to read the received data.

Syntax

```
PXETH_RX() as char
```

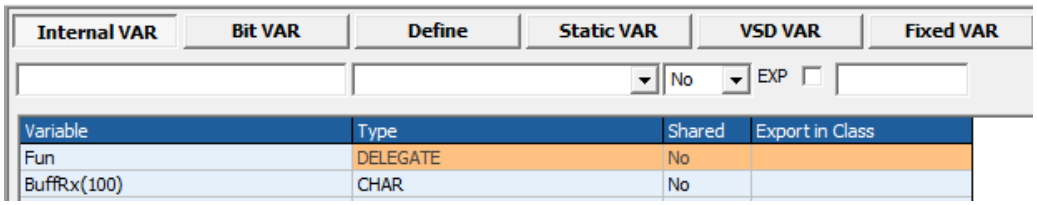
Return Value
Char Data read from the receive buffer

2.4 Example

In the following example, when is received a TCP/IP block, are checked the **first 3 characters** in the buffer. If these are equal to the string **"VTB"**, the reply is **"YES"**, otherwise the reply is **"NO"**
Is checked the ASCII code:

V = 86 **Y** = 89 **N** = 78
T = 84 **E** = 69 **O** = 79
B = 66 **S** = 83

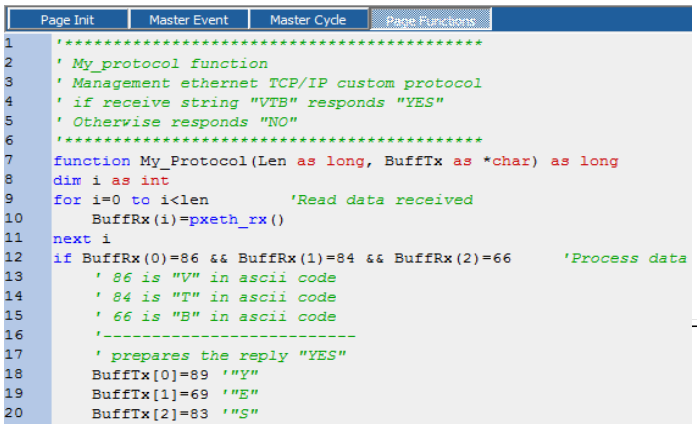
Variables used



Code in Init Main


```
Set_ip("10,0,0,15","255,255,255,0",0)      'IP = 10,0,0,15
                                           'SUBNET = 255,255,255,0
                                           'GATEWAY = none
Fun=my_protocol
pxeth_add_prot(502,Fun) 'Add Function my_protocol to 502 port
```

Code in Main Page Functions



```
'*****************************************
' My_protocol function
' Management ethernet TCP/IP custom protocol
' if receive string "VTB" responds "YES"
' Otherwise responds "NO"
'*****************************************
function My_Protocol(Len as long, BuffTx as *char) as long
dim i as int

for i=0 to i<len        'Read data received
      BuffRx(i)=pxeth_rx()
next i

if BuffRx(0)=86 && BuffRx(1)=84 && BuffRx(2)=66       'Process data
      ' 86 is "V" in ascii code
      ' 84 is "T" in ascii code
      ' 66 is "B" in ascii code
      '--------------------------
      ' prepares the reply "YES"
      BuffTx[0]=89 '"Y"
      BuffTx[1]=69 '"E"
      BuffTx[2]=83 '"S"
      My_Protocol=3 ' Data len for YES 3 Chars
else
      ' prepares the reply "NO"
      BuffTx[0]=78 '"N"
      BuffTx[1]=79 '"O"
      My_Protocol=2 ' Data len for NO 2 Chars
endif
endfunction
```

Example Download

3 Modbus TCP/IP

The Ethernet Port, can be configured with TCP(/P MODBUS Protocol
The TCP/IP STACK, can supporting, multi protocols connection.
The TCP/IP MODBUS, is managed by VTB OBJECT

3.1 Modbus TCP/IP OBJECT

This object, manages, the TCP/IP Modbus protocol

Property

Nodo	Node slave
IpAddress	Slave IP Adrress ex. “10.0.0.80”
Service Port	Slave IP Port (default 502)
PtData()	Array Data Register
Max Len Data	Data Register dimension

Methods

No

The following requests are handled:

Function Code 3	Read Multiple Registers
Function Code 4	Read Input Registers
Function Code 6	Preset Single Registers
Function Code 16	Preset Multiple Registers

Events

No

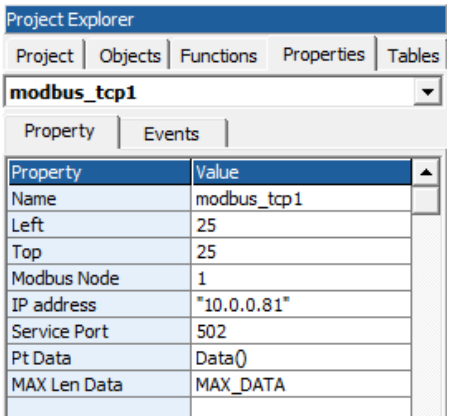
3.2 Example

In the following example, are read and written the 16 bit registers in NGMEVO memory.
The Array data, is named – **Data,** and the maximum number register, is in the DEFINE **MAX_DATA** (100 Register in the example)

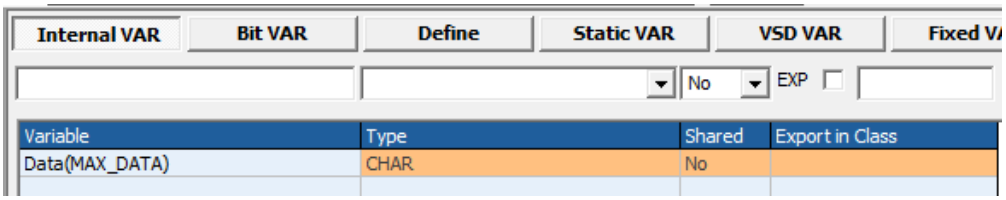
Where :
Read/Write from Modbus register Nr.1 → Data(0)
Read/Write from Modbus register Nr.2 → Data(1)
etc.
In the example, is read the register Nr.2 – Data(1), and written the register Nr. 1 - Data(0)

Objects used:

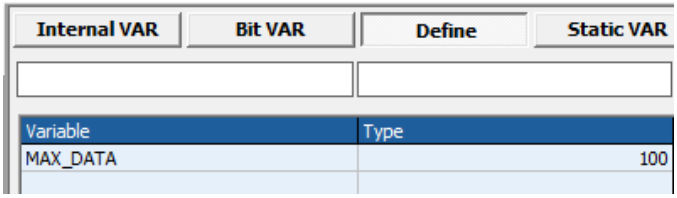
Modubus → CModbus → Modbus protocol TCP



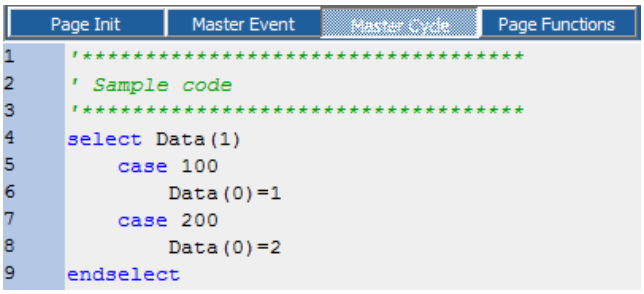
Variables used



DEFINE used



Code in Master Ciclo – Main




```
'*******************************
' Sample code
'*******************************
select Data(1)
      case 100
            Data(0)=1
      case 200
            Data(0)=2
endselect
```

Example Download

4 RS232/RS485 Port

The NGMEVO allows to use 1 RS232/485 port, with a custom or standard (MODBUS RTU) protocols.

4.1 SER_SETBAUD

Programming the BaudRate of the second SERIALE PORT - SER2.

Syntax

SER_SETBAUD (Baud as long) as void

Parameters

Baud Value of Baud Rate. The standard value are:
1200-2400-4800-9600-19200-38400-57600-115200

4.2 SER_MODE

Programming the mode of the second SERIAL PORT. If this function is never called, by default the port is programmed with:

No parity , 8 bit per character , 1 bit stop.

Syntax

SER_MODE(par as char, nbit as char, nstop as char) as void

Parameters

par Parity (0=no parity, 1=odd parity, 2=even parity)
nbit Number of bits per character (7 or 8)
nstop Number of stop bits (1 or 2)

4.3 SER_GETCHAR

Reads the receive buffer of the serial port. It doesn't wait for the presence of a character. This function, must be calling, in POLLING by VTB application. The operating System, manages the INTERRUPT BUFFER

Syntax

SER_GETCHAR () as int

Return Value

int ***-1*** No character is in the buffer
>=0 Code (0 to 255) of the character read from the buffer

4.4 SER_PUTCHAR

Sends a character to the serial port.

Syntax

SER_PUTCHAR (Car as int) as void

Parameters

Car Code (0 to 255) of the character to send

4.5 SER_PUTS

Sends a string of characters to the serial port. The string must be ended with the character 0 (NULL).

Syntax

SER_PUTS (str as *char) as void

Parameters

***str** String Pointer

4.6 SER_PRINTL

SFormatting print of an INTEGER value.

Syntax

SER_PRINTL (format as *char,val as long) as void

Parameters

Format	String corresponding to the format to be printed
Val	Any integer value or expression

Avalaible formats

######	Print a fixed number of characters	23456
###.###	Force the print of decimal point	123.456
+####	Force the print of the sign	+1234
#0.##	Force the print of a ZERO	0.12
X####	Print in HEXADECIMAL format	F1A3
B####	Print in BINARY format	1011

4.7 SER_PRINTF

Formatting print of a FLOAT value. It is the same as ***ser_printl*** but use a float value

Syntax

SER_PRINTF (const char *format, val as float) **as void**

Parameters

Format	String corresponding to the format to be printed
Val	Any integer value or expression

4.8 SER_PUTBLK

Sends a precise number of characters to the serial port. Unlike the function ***ser_puts*** it allows to send also the character with 0 code enabling the managing of binary protocols, furthermore it starts the background transmission setting in appropriate mode the RTS signal useful to work with RS485 lines.

WARNING
This function allows to manage BINARY and RS485

Syntax

SER_PUTBLK (Buffer as *char, Len as int) as void

Parameters

***Buffer**	Pointer to the data buffer to send
Len	Number of bytes to send

4.9 SER_PUTST

Reads the state of background transmission started by ***ser_putblk.***

Syntax

SER_PUTST () as int

Return Value

int	***-1***	Transmit error
	>=0	Number of characters to be transmitted

4.10 Example

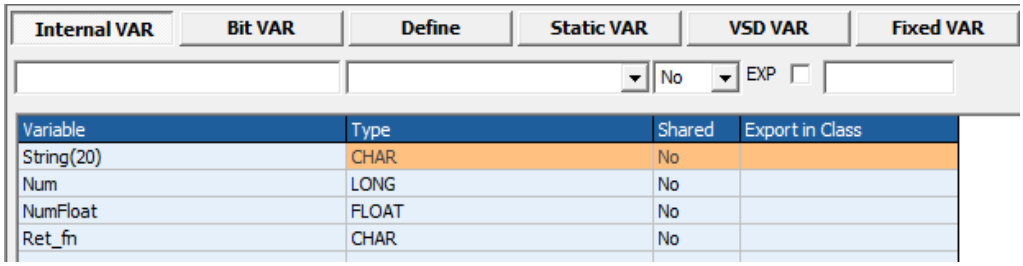
In the following example, is call the Read_Data() function, in polling in the Task Main
SER2 Setting:

Baud rate → ***115,200***
Nr. bit dati → ***8***
Nr. bit Stop → ***1***
Parità → ***NO***

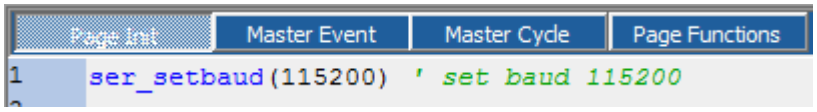
Response value:

Character received =1 → Echo charatcer reveived (1) with ser_putchar
Character received =2 → Send Text "Test String" with Ser_puts
Character received =3 → Formatted Print Variable Num (number of characters received)
Character received =4 → Formatted Print Variable NumFloat (Float random)
Character received =5 → Send in Binary mode Nr. 789488 with Ser_putblk
Character received =6 → Test state Ser_putblk - reply:
255 send data error
Nr characters in the transmission buffer
Character received=Others → Response 254 - Error unknown command

Variables used

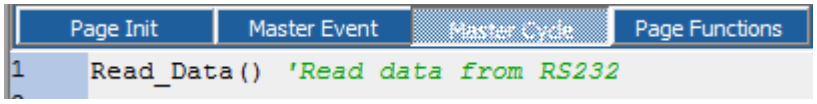


Code in Init Main




```
ser_setbaud(115200) ' set baud 115200
```

Code in Master Ciclo Main




```
Read_Data() 'Read data from RS232
```

Code in Page Functions Main

Page Init | Master Event | Master Cycle | Page Functions

```
'****************************
'Read Data From RS232
'****************************
function Read_Data() as void
Ret_fn=Ser_getchar()     ' Read one char from RS232 buffer
if Ret_fn=-1 ' none
    return ' return
endif
```

```
'****************************
'Read Data From RS232
'****************************
function Read_Data() as void

Ret_fn=Ser_getchar()     ' Read one char from RS232 buffer
if Ret_fn=-1 ' none
      return ' return
endif
inc Num      ' increases the received chars
NumFloat=Num*2.13 'random number
'process data received
select Ret_fn
      case 1 ' ------ echo char with send_putchar
            Ser_putchar(Ret_fn)      ' send reply echo char
      case 2 ' ------ send string with ser_puts
            strcpy(String(),"Test String") ' Copy in array string text
            ser_puts(String()) ' put data
      case 3 ' ------ print a long formatted with ser_printl
            ser_printl("###.##",Num) ' print ex: 123.45 format
      case 4 ' ------ print a float formatted with ser_printf
            ser_printf("####.###",NumFloat) ' print NumFloat
      case 5 ' ------ put a block with ser_putblk
            'Send a number 789488
            String(0)=0xF0 'LSB
            String(0)=0xOB
            String(0)=0x0C
            String(0)=0 'MSB
            Ser_putblk(String(),4) ' Data len 4 byte
      case 6 ' ------ test if ser_putblk is busy
            Ret_fn= Ser_putst() ' check if function ser_putblk is busy
            if Ret_fn=-1
                  Ser_putchar(255)  ' send error
            else
                  Ser_putchar(Ret_fn)      ' send number of chars
            endif
      case else
            Ser_putchar(254)  ' send error no char
endselect

endfunction
```

Example Download

5 Modbus RTU

The SER2 port, is able to manage the RTU MODBUS protocol.
The protocol MODBUS RTU is available in two configuration:
Master
Slave

5.1 Modbus RTU Slave Object

Thsi Object, manage, the RTU MODBUS SLAVE protocol.

Property

Nodo	Node slave
BaudRate	baud rate
PtData()	Array Data Register in the NGMEVO memory
Max Len Data	Data Register dimension
TimeOut	Master Time Out (millisecond) This must be smallest by a MASTER TimeOut

Methods

No

The following requests are handled MODBUS RTU:

Function Code 3	Read Multiple Registers
Function Code 6	Preset Single Registers
Function Code 16	Preset Multiple Registers

Events

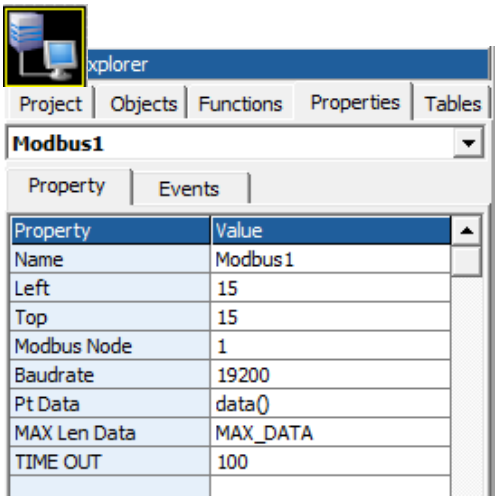
No

5.2 Example ModBus slave

In the next example,are read and written, some registers 16 bit declared in the NGMEVO memory.
The registers array is named **Data,** and the maximum dimension, is in the DEFINE **MAX_DATA**

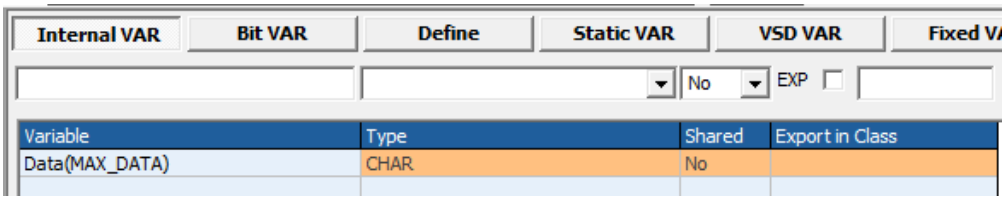
Where :
Read/Written from Modbus register Nr.1 → Data(0)
Read/Written from Modbus register Nr.2 → Data(1)
etc.
The example, Read the data register Nr. 2 - Data(1) and written the Data register Nr1 - Data(0)

Objects used:

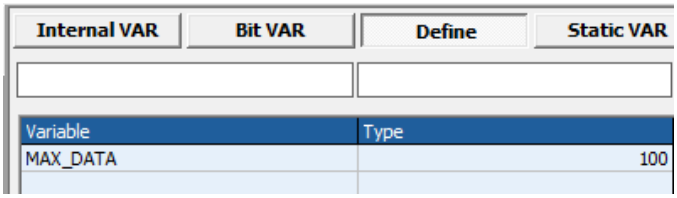


Modbus → Cmodbus → ModBus Protocol

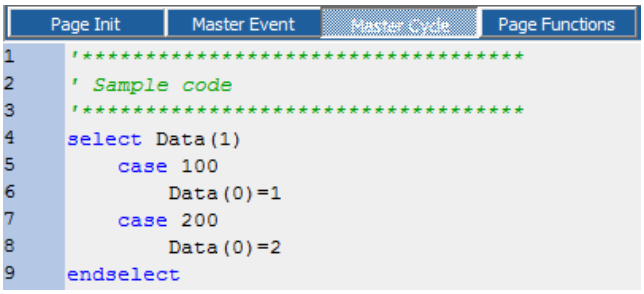
Variables used



DEFINE Used



Code in Master Ciclo Main




```
'*****************************
' Sample code
'*****************************
select Data(1)
      case 100
            Data(0)=1
      case 200
            Data(0)=2
endselect
```

Example Download

5.3 Modbus RTU Master Object

Thsi Object, manage, the RTU MODBUS MASTER protocol.

Property

BaudRate	Comm Baud rate
TimeOut	Time Out for SLAVE response (millisecond). This must be more great by a slaves TimeOut
Parita	0 none - 1 odd - 2 even
N. bit car	Number bit for char
N. bit stop	Number stop bit

Methods

function .write_regn(nodo as char, addr as uint, value as *int) as char

Preset single register func 16 ModBus RTU

Parameters

nodo	Node slave modbus
addr	Start Address register to write (Slave)
Value	Unsigned integer (values to write)

Return

0	Write OK
1	Error respons
2	Time Out
3	Data len > 127

function .read_regn(nodo as char, addr as uint, value as *int) as char

Read single register func 3 ModBus RTU

Parameters

nodo	Node slave modbus
addr	Start Address register to read (Slave)
Value	Pointer to unsigned integer (value to read)

Return

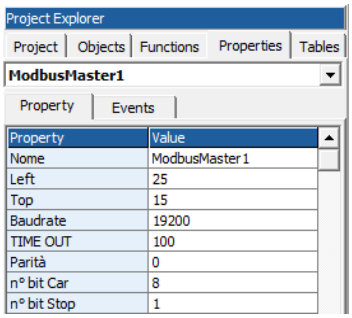
0	Read OK
1	Error respons
2	Time Out
3	lData len > 127
4	Checksum error

5.4 Example ModBus Master

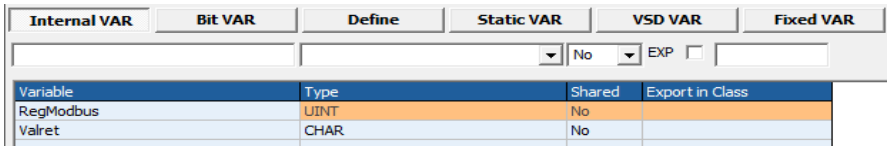
In the next example, are read and written, some registers 16 bit in a slave

Objects used:

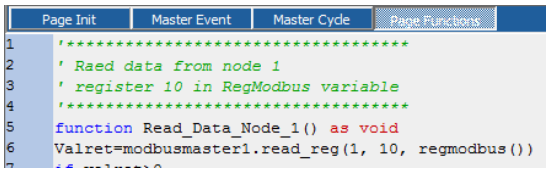
Modbus → CmodbusMaster → ModBus Master Protocol



Variables used



Code in Main Page Functions




```
'**********************************
' Raed data from node 1
' register 10 in RegModbus variable
'**********************************
function Read_Data_Node_1() as void
Valret=modbusmaster1.read_regn(1, 10, regmodbus())
if Valret>0
      ' read error
endif
endfunction
'**********************************
' Write data to node 1
' register 10 RegModbus variable
'**********************************
function Write_Data_Node_1() as void
RegModbus=100
Valret=modbusmaster1.write_regn(1, 10, RegModbus)
if valret>0
      ' write error
endif
endfunction
```

Example Download

6 Analog Inputs Read

The NGMEVO board, has 8 analog inputs managed by VTB functions
In this Board revision, the analog inputs, have a 12 bit resolution (value from 0 to 4095)

6.1 Inputs Read

Syntax

NG_ADC(Channel **as Char**) **as uint**

Parameters

Channel Channel number (from 0 to 7)

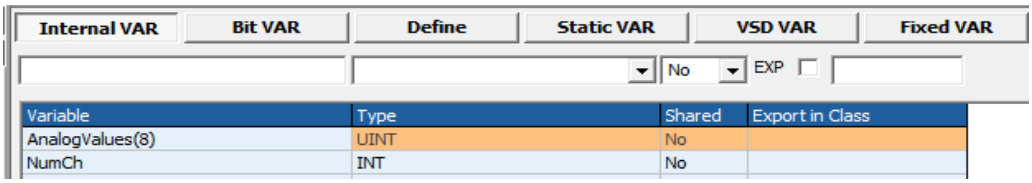
Return Value

Returns the analog value (from 0 to 4095)
Where 0 is the minimum voltage level (0 volt) , 4095 is the maximun voltage level configured in the input (normally 10 Volt)

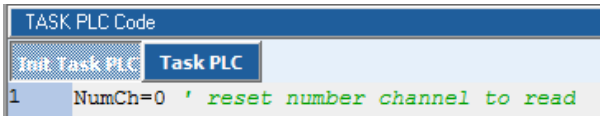
6.2 Example Analog inputs read

In the following example, are read the analog inputs from 0 to 7. the values are written in the array AnalogValues
The channels are read in TaskPlc

Variables used

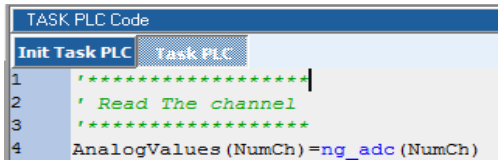


Code in Init TaskPlc




```
NumCh=0      ' reset number channel to read
```

Code in Task Plc




```
'******************
' Read The channel
'******************
AnalogValues(NumCh)=ng_adc(NumCh)
inc NumCh 'increase channel number
if NumCh=8 ' limit
      NumCh=0
endif
```

Example Download

7 Writing Analog Output 0-10V on NGMEVO

The NGMEVO can use an Analog Output 0-10V. The Analog output must be configured in Promax. It excludes the digital Output 1

7.1 Writing Analog 0-10V

The Analog Output is managed by the object:

General → Cpwm.vco → PWM NGM – EVO

It initializes the system for analog output 0-10V.
When the object is inserted, the digital output 1 is inhibited

The max analog output value is 12 Volt and not 10 Volt. For obtain 10 Volt output the max value in Quibdi PWM_VAL() function, must be 212 and not 255

Property

Enable	True (1) for enabling analog output
Polarity	Must be True
Center Align	Must be False
Freq	Must be 50000
Divisioni	Must be 256

Methods

Pwm_val(Channel as char, Value as uint) as void

Parameters

Channel	Output Channel (must be 0 , it is used for future expansions)
Value	Output Divisions 0 = 0 Volt 212 = 10 Volt 255 = 12 Volt

WARNING
A VALUE 255 is 12 Volt
USE MAX VALUE 212 FOR 10 Volt

8 Internal Encoder read on NGMEVO

The NGMEVO can use an internal encoder channel with freq max 35 Khz

8.1 NG_ENC – ENCODER MANAGEMENT

First to use the internal encoder, is necessary insert the following function in the TASK INIT MAIN:

System_Utility(151,0,0,0)

After that the channel can be read:

NG_ENC(Channel **as Char**, Value **as *Long**) **as void**

Parameters

Channel	Number of channel **MUST BE 16**
Value	Pointer to a long variable where will be contained the counter

WARNING

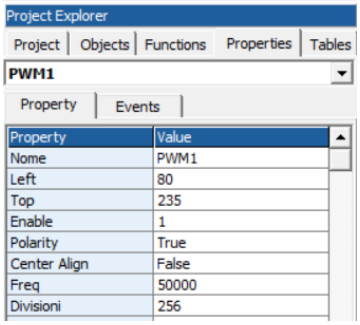
FOR A SINCRONOUS FUNCTION, USE NG_ENC ONLY IN TASK PLC

8.2 Example Analog Output 0-10V on NGM EVO

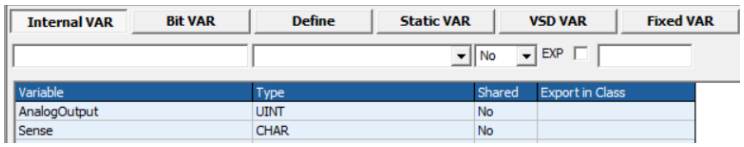
In the following example, the analog oputput, is written in the task plc

Objects used:

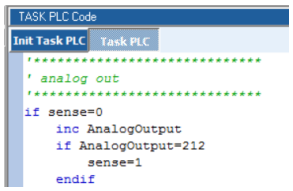
General→ Cpwm.vco→ PWM NGM – EVO



Variables used



Code in Task Plc




```
'*****************************
' analog out
'*****************************
if sense=0
      inc AnalogOutput
      if AnalogOutput=212
            sense=1
      endif
else
      dec AnalogOutput
      if AnalogOutput=0
            sense=0
      endif
endif

pwm_val(0,AnalogOutput) ' output val
endif
endfunction
```

Example Download

9 CanOpen Management

The NGMEVO board, can use, one Canopen line Master/Slave
The MASTER port, can use the PDO programmable by CanOpen configuurator.(see Canopen Configurator - Link Chapr. 14)

9.1 PXCO_SDODL

This function allows to send data to a node of the canopen net using the protocol SDO. It is supported only the SDO EXPEDITED mode allowing to send up to 4byte of data length.

Syntax
PXCO_SDODL(node **as char**, index **as int**,subidx **as uchar**,len **as long**,data **as *char**) **as char**

Parameters

Node	Node ID of the SLAVE to whch send data
Index, subindex	Address in the Object-Dictionary of the data to be written
Len	Number of bytes to send
***data**	Pointer to the data to send

Return value

char	**0**	No error
	<>0	Communication error
	=2	The node responded with a SDO ABORT CODE, calling the function ***read_sdoac*** in the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 will be available the relative error code.

WARNING
DO NOT USE THIS FUNCTION IN TASK PLC

9.2 PXCO_SDOUL

This function allows to read data from a node of the canopen net using the protocol SDO. It is supported only the SDO EXPEDITED mode allowing to read up to 4byte of data length.

Syntax
PXCO_SDOUL(node **as char**, index **as uint**,subidx **as uchar**,dati **as *char**) **as char**

Parameters

Node	Node ID of the SLAVE to whch send data
Index, subindex	Address in the Object-Dictionary of the data to be written
***data**	Pointer to the data to send

Return value

char	**0**	No error
	<>0	Communication error
	=2	The node responded with a SDO ABORT CODE, calling the function ***read_sdoac*** int the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC0 will be available the relativeerror code.

WARNING
DO NOT USE THIS FUNCTION IN TASK PLC

9.3 READ_SDOAC

Reading of the SDO ABORT CODE sended by a node in the canopen net as answer to a request done with the function PXCO_SDODL or PXCO_SDOUL. The read code will be written in the system variables _SYSTEM_SDOAC0 e _SYSTEM_SDOAC1.
Refer to the DS301 specific of the CAN OPEN for the code error values.

Syntax

```
READ_SDOAC() as void
```

9.4 PXCO_SEND

Sending of a CAN frame at low level. This function allows to send in the net a CAN frame with a desired COB-ID and DATS. For example it's possible to send manually PDO frames, HEART-BEAT frames, etc.
Should be specified the manage of PDO is managed AUTOMATICALLY by the CANOPEN CONFIGURATOR.

Syntax

```
PXCO_SEND(id as int, Len as char,Dati as *char) as char
```

Parameters

id	COB-ID value
Len	Number of data to send
***Data**	Pointer to the data buffer

Return value

char	**0**	No error
	<>0	Communication error

9.5 PXCO_NMT

Sending of a NMT frame of the CAN OPEN. NMT protocol allows to set the state of the nodes in the net. Remind that all the nodes correctly configured (canopen configurator) are automatically set in START state.

Syntax

```
PXCO_NMT(state as char, node as char) as char
```

Parameters

state	State to set: 1 = START NODE 2 = STOP NODE 128 = PRE-OPERATIONAL 129 = RESET NODE 130 = RESET COMUNICATION
node	Number of the node

Return value

char	**0**	No error
	<>0	Communication error

WARNING
DO NOT USE THIS FUNCTION IN TASK PLC

9.6 READ_EMCY

Reads the last EMERGENCY OBJECT frame sended by a CAN OPEN node.
The emergency code is written in the system array _SYSTEM_EMCY(8) and it will contain all the 8 bytes of the EMERGENCY OBJECT frame as from the DS301 specific of the CAN OPEN. Usually it is called cyclically. The emergency code depends by type of connected device, therefore refer to its manual.

Syntax

```
READ_EMCY()  as char
```

Return Value

char	**0**	No error
	<>0	Node that generated the emergency object.

_SYSTEM_EMCY							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Emergency Error Code		**Error Register**	**Manufacturer specific Error Code**				

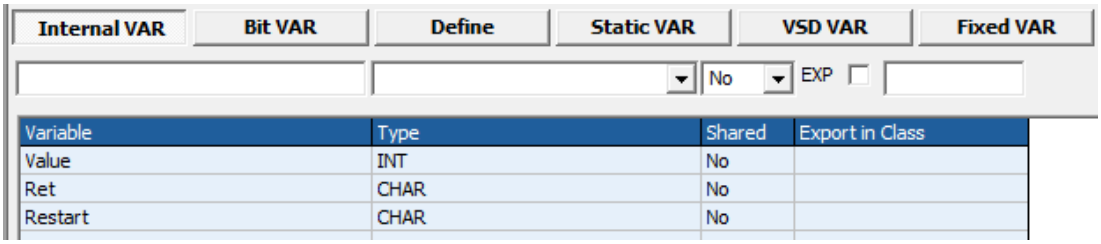
WARNING

THE SYSTEM DOESN'T BUFFER MORE THAN ONE MESSAGE, THEN IF MORE EMERGENCY OBJECT ARE SENDED ALONG A SINGLE TASK PLC, ONLY THE LAST WILL BE READ.
AN EMERGENCY OBJECT DOES NOT MEAN THAT THERE IS ACTUALLY A NODE IN AN EMERGENCY. THE DS301 SPECIFIC PROVIDE THAT AN EMERGENCY OBJECT ARE SEND ALSO ON ALARM RESET. FURTHERMORE SOME DEVICES CAN BE SEND THIS FRAME AT START UP.

9.7 Example CanOpen Functions

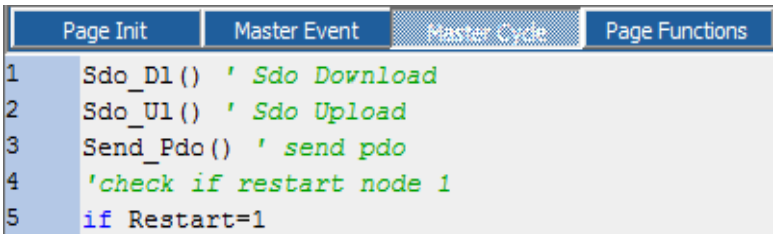
In the following example, are used the Canopen Functions.

Variables used



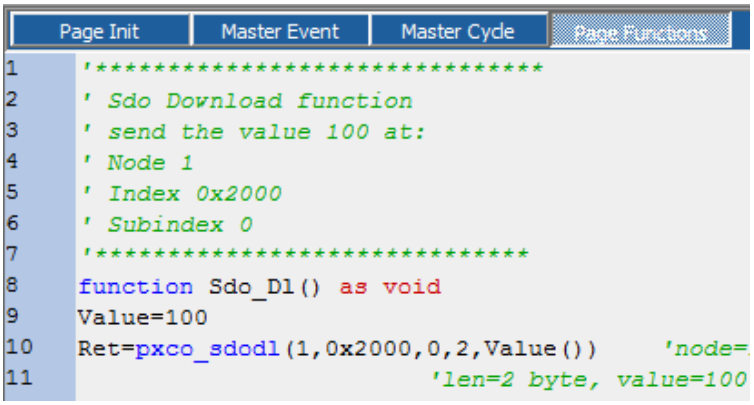
Variable	Type	Shared	Export in Class
Value	INT	No	
Ret	CHAR	No	
Restart	CHAR	No	

Code in Master Ciclo Main




```
Sdo_Dl() ' Sdo Download
Sdo_Ul() ' Sdo Upload
Send_Pdo() ' send  pdo
'check if restart node 1
if Restart=1
      Restart=0 ' reset flag restart
      Ret=pxco_nmt(1,1) ' Start Node
      if Ret<>0   'test error
            '...
      endif
endif

'polling emergency object
Ret=Read_emcy()
if Ret<>0
      ' in Ret node error
      ' in _SYSTEM_EMCY code error
endif
```

Code in Main Page Functions




```
'*******************************
' Sdo Download function
' send the value 100 at:
' Node 1
' Index 0x2000
' Subindex 0
'*******************************
function Sdo_Dl() as void
Value=100
Ret=pxco_sdodl(1,0x2000,0,2,Value())'node=1, index=0x2000, subidx=0,
                                    'len=2 byte, value=100
if Ret<>0   'test error
      if Ret=2
            read_sdoac()'Read SDO ABORT CODE
            'in _SYSTEM_SDOAC0 code error
            'in _SYSTEM_SDOAC1 code error
      endif
      '...
endif
endfunction

'*******************************
' Sdo Upload function
' read the value at:
' Node 1
' Index 0x2000
' Subindex 0
'*******************************
function Sdo_Ul() as void
Ret=pxco_sdoul(1,0x2000,0,Value())  'node=1, index=0x2000, subidx=0,
                                    'read in value
if Ret<>0   'test error
      if Ret=2
            read_sdoac()'Read SDO ABORT CODE
            'in _SYSTEM_SDOAC0 code error
            'in _SYSTEM_SDOAC1 code error
      endif
      '...
endif
endfunction

'*******************************
' Send PDO
' COB - ID = 0x201
```

```
' 2 Bytes
' SVariable in Value
'*****************************
function Send_Pdo() as void
Value=100
Ret=pxco_send(0x201,2,Value())      'cob-id=0x201) 2 bytes
if Ret<>0   'test error
            '...
endif
endfunction
```

Example Download

9.8 Example CanOpen Axes interpolation mode

In the following example, are managed 3 CanOpen Axes in linear interpolation.

ATTENTION:
All speed are managed in mm/min if setted the following parameters
RapX,RapY,RapZ
All axes target positions are managed in micron (0.001 mm) if setted the following parameters
RapX,RapY,RapZ

Objects used

Motor Control → CobjInterpola → Interpolatore

Project Explorer

Project | Objects | Functions | Properties | Tables

Interp

Property | Events

Property	Value
Nome	Interp
Left	15
Top	10
N.assi	3
N.tratti	16
Vper	1024
Div. Vper	1024
Abilita arcto	1

Motor Control → CstdCanOpen → Ds402 x 3

Project Explorer

Project | Objects | Functions

AxisX

Property | Events

Property	Value
Name	AxisX
Left	10
Top	85
Node	1
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

Project Explorer

Project | Objects | Functions

AxisY

Property | Events

Property	Value
Name	AxisY
Left	55
Top	85
Node	2
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

Project Explorer

Project | Objects | Functions

AxisZ

Property | Events

Property	Value
Name	AxisZ
Left	100
Top	85
Node	3
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	1000

Are managed the following functions:

Wait_Move – Axes state movement
Parameters No
Return 1 Axes in movement
0 Axes stop

Move_Axes – Move the Axes in linear interpolation
Parameters Vel → Feed Axes in mm/min
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
(Stop Axes only if edge > SGLP)
Px,Py,Pz → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axes – Set interpolation Acceleration
Parameters Value → Value in count per TAU
Return No

Stop_Axes – Stop Axes
Parameters No
Return No

Enable_Axis_X_Y_Z – Enable the Axes control and preset at value 0
Parameters No
Return No

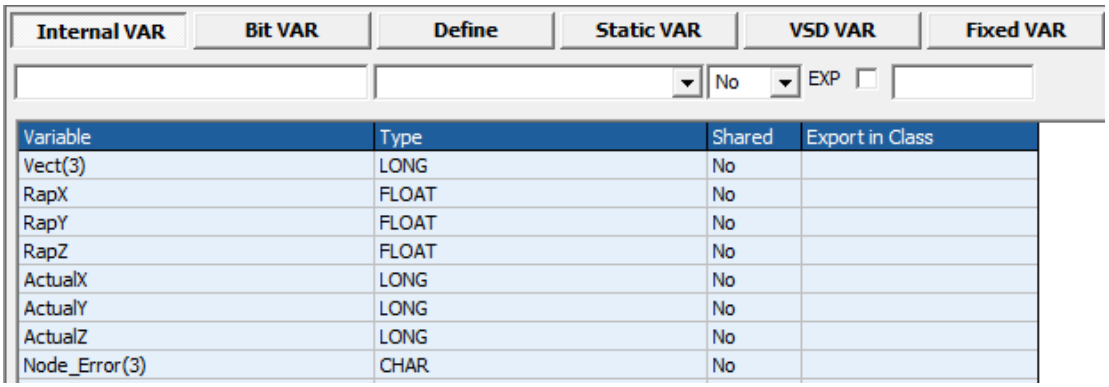
Disable_Axis_X_Y_Z – Disable the Axes control
Parameters No
Return No

cancfgerr – CanOpen Custom Error. This function is called at Canopen Node init (node setted in configuration by Canopen configurator) when the node, reply error
Parameters Node → Node number in error
Err → Error code
Return No

Close_cancfgerr - CanOpen Custom Error. This function is called at end Canopen nodes configuration
Parameters No
Return No

Open_cancfgerr - CanOpen Custom Error. This function is called at start Canopen nodes configuration
Parameters Nodes → Nodes number in configuration
Return No

Variables used

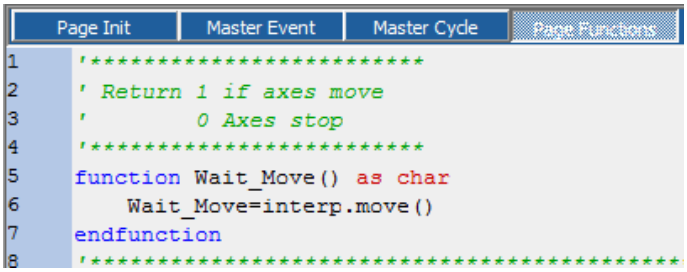


Internal VAR | Bit VAR | Define | Static VAR | VSD VAR | Fixed VAR

No | EXP

Variable	Type	Shared	Export in Class
Vect(3)	LONG	No	
RapX	FLOAT	No	
RapY	FLOAT	No	
RapZ	FLOAT	No	
ActualX	LONG	No	
ActualY	LONG	No	
ActualZ	LONG	No	
Node_Error(3)	CHAR	No	

Code in Main Page Functions



Page Init | Master Event | Master Cycle | Page Functions

```
'*************************
' Return 1 if axes move
'        0 Axes stop
'*************************
function Wait_Move() as char
    Wait_Move=interp.move()
endfunction
'*****************************************
```

```
'*************************
' Return 1 if axes move
'        0 Axes stop
'*************************
function Wait_Move() as char
      Wait_Move=interp.move()
endfunction

'************************************************************
' Move Axes
' Vel= interp vel Axes in mm/min
' Flg if 1 move without buffer
'        0 move in buffer mode
' Px,Py,Pz Axes value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'       0 The movement is not inserted in the buffer
'         in this case, is necessary reload the movement
'************************************************************
function Move_Axes(Vel as long, Flg as char, Px as long, Py as long,Pz as
long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Vect(0)=Px
      Vect(1)=Py
      Vect(2)=Pz
      Move_Axes=interp.moveto(Vel, Flg, Vect())
endfunction

'************************************************************
' Set ACC
' Value Acc value in count
'************************************************************
function Acc_Axes(Value as long) as void
```

```
      interp.acc=Value
endfunction
'*********************************************************
' Stop Axes
'*********************************************************
function Stop_Axes() as void
      interp.stop()
endfunction
'*******************
' Axis X enable
'*******************
function Enable_X() as void
AxisX.modo=0 ' remove interpolation mode
AxisX.start=0 ' stop PDO Qx
'Preset Axis X 0, not change y,z
Vect(0)=0
Vect(1)=interp.pc(1)
Vect(2)=interp.pc(2)
interp.preset(Vect())
AxisX.home=0 'preset driver
'enable axis
AxisX.enable=1
AxisX.start=1 ' start PDO Qx
AxisX.modo=2 ' set interpolation mode
endfunction
'*******************
' Axis X Disable
'*******************
function Disable_X() as void
AxisX.enable=0
endfunction

'*******************
' Axis Y enable
'*******************
function Enable_Y() as void
AxisY.modo=0 ' remove interpolation mode
AxisY.start=0 ' stop PDO Qx
'Preset Axis Y 0, not change x,z
Vect(0)=interp.pc(0)
Vect(1)=0
Vect(2)=interp.pc(2)
interp.preset(Vect())
AxisY.home=0 'preset driver
'enable axis
AxisY.enable=1
AxisY.start=1 ' start PDO Qx
AxisY.modo=2 ' set interpolation mode
endfunction

'*******************
' Axis Y Disable
'*******************
function Disable_Y() as void
AxisY.enable=0
endfunction

'*******************
' Axis Z enable
```

```
'********************
function Enable_Z() as void
AxisZ.modo=0 ' remove interpolation mode
AxisZ.start=0 ' stop PDO Qx
'Preset Axis Z 0, not change x,y
Vect(0)=interp.pc(0)
Vect(1)=interp.pc(1)
Vect(2)=0
interp.preset(Vect())
AxisZ.home=0 'preset driver
'enable axis
AxisZ.enable=1
AxisZ.start=1      ' start PDO Qx
AxisZ.modo=2       ' set interpolation mode
endfunction

'********************
' Axis Z Disable
'********************
function Disable_Z() as void
AxisZ.enable=0
endfunction

'************************
' Error check
' CanOpen node
'************************
function cancfgerr(node as int,err as uchar) as void
Node_Error(node)=err ' copy the code error
endfunction
'************************
' Close init CanOpen
'************************
function close_cancfgerr() as void
endfunction
'************************
' Custom error init
' CanOpen node
'************************
function open_cancfgerr(nodes as int) as void
' Reset nodes  status error
Node_Error(0)=0
Node_Error(1)=0
Node_Error(2)=0
endfunction
```

Code in Init Task PLC

TASK PLC Code

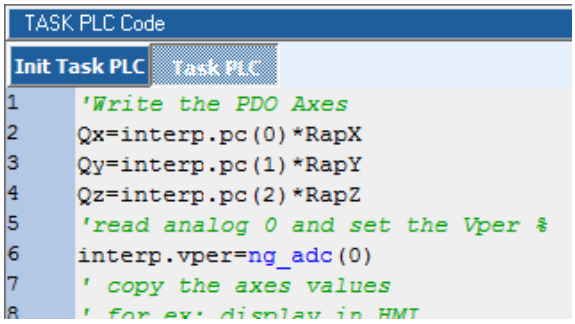
Init Task PLC | Task PLC

```
1  '************************************************
2  'Ex: Motor Encoder Revolution = 10000 i/rev
3  'Motor inserted directly in the Screw 5 mm step
4  'Rap=10000/5000=2
5  '************************************************
6  Rapx=1
7  Rapy=1
8  Rapz=1
```

```
'************************************************
```

```
'Ex: Motor Encoder Revolution = 10000 i/rev
'Motor inserted directly in the Screw 5 mm step
'Rap=10000/5000=2
'************************************************
Rapx=1
Rapy=1
Rapz=1
```

Code in Task PLC




```
'Write the PDO Axes
Qx=interp.pc(0)*RapX
Qy=interp.pc(1)*RapY
Qz=interp.pc(2)*RapZ
'read analog 0 and set the Vper %
interp.vper=ng_adc(0)
' copy the axes values
' for ex: display in HMI
' value in 0.001 mm
ActualX=interp.pc(0)
ActualY=interp.pc(1)
ActualZ=interp.pc(2)
```

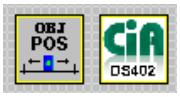
Example Download

9.9 Example CanOpen Axes position mode

In the following example, are management, a CanOpen Axis by VTB OBJECT
See doc Vtb Object Guide for more informations.

WARNING:
All speed are managed in mm/min if setted the following parameters:
MSOF e DSOF
All axes target positions are managed in micron (0.001 mm) if setted the following parameters:
MSOF e DSOF

Objects used:



Motor Control Plus → CobjPos → Posizionatore

Project Explorer

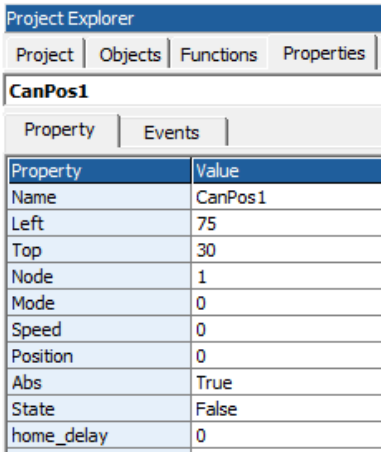
Project | Objects | Functions | Properties

Pos1

Property | Events

Property	Value
Vper	1024
Div. Vper	1024
AccQstop	10
Acc	5
RzeroMode	1
RzeroOffset	0
RzeroPreset	0
RzeroVel	10
RzeroVelf	5
RzeroAcc	10
Msof	10000
Dsof	5000
LimitN	-99999999
LimitP	99999999
Gioco	0
Vgioco	1
MsofV	1
DsofV	1
RZERO ENABLE	True
AXIS TYPE	1
VTB AXIS OBJECT	CanPos1
PDO NAME	qx
STEP CHANNEL	0
STEP NODE	1

Motor Control → CstdCanOpen → Ds402



Property	Value
Name	CanPos1
Left	75
Top	30
Node	1
Mode	0
Speed	0
Position	0
Abs	True
State	False
home_delay	0

Are managed the following functions:

Wait_Move – Axis state movement
Parameters No
Return 1 Axis in movement
0 Axes stop

Move_Axis – Move the Axis
Parameters Vel → Feed Axes in mm/min
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
Px, → Axes target values in 0.001 mm
Return 0 Movement inserted in the buffer – buffer empty
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axis – Set Acceleration
Parameters Value → Value in count per TAU
Return No

Stop_Axis – Stop Axes
Parameters No
Return No

Enable – Enable the Axis control and preset at value 0
Parameters No
Return No

Disable – Disable the Axes control
Parameters No
Return No

StartHome – Start homing - Vel in pos1.rzerovel and pos1.rzerovelf
Parameters No
Return No

CheckHome – Check homing state
Parameters No
Return 1 homing finished

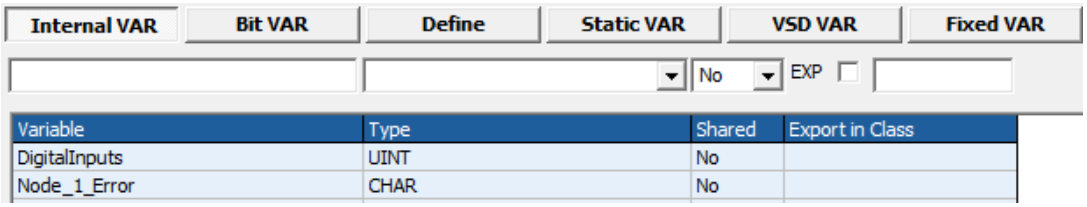
StopHome – Stop homing
Parameters No
Return No

cancfgerr – CanOpen Custom Error. This function is called at Canopen Node init (node setted in configuration by Canopen configurator) when the node, reply error
Parameters Node → Node number in error
Err → Error code
Return No

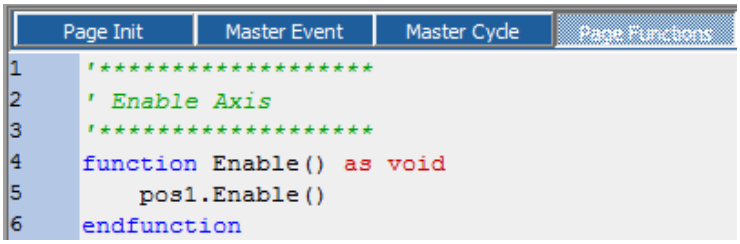
Close_cancfgerr - CanOpen Custom Error. This function is called at end Canopen nodes configuration
Parameters No
Return No

Open_cancfgerr - CanOpen Custom Error. This function is called at start Canopen nodes configuration
Parameters Nodes → Nodes number in configuration
Return No

Variables used



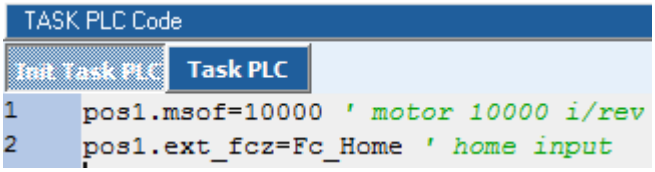
Code in Main Page Functions




```
'********************
' Enable Axis
'********************
function Enable() as void
      pos1.Enable()
endfunction
'********************
' Disable Axis
'********************
function Disable() as void
      pos1.Disable()
endfunction
'********************
' Preset Axis
'********************
function Preset(Val as long) as void
      pos1.Preset(Val)
endfunction
'*************************
' Return 1 if axis move
'        0 Axis stop
'*************************
function Wait_Move() as char
      Wait_Move=pos1.move()
endfunction
'********************
' Axis Stop Move
'********************
function Stop() as void
      pos1.Stop()
endfunction
'*****************************
' Start Homing
' Homing input see in task plc
'*****************************
function StartHome() as void
      pos1.StartHome()
endfunction
'*****************************
' Check if homing finished
```

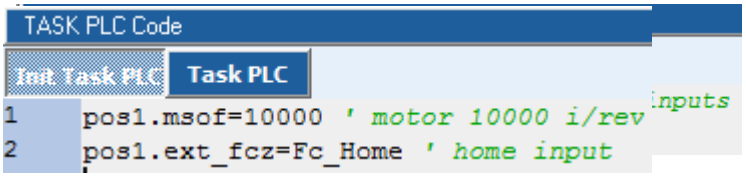
```
' Return 1 if finished
'****************************
function CheckHome() as char
      CheckHome=pos1.status_home
endfunction
'****************************
' Stop home function
'****************************
function StopHome() as void
      pos1.StopHome()
endfunction
'*************************************************************
' Move Axis
' Vel=  vel Axis in mm/min
' Flg if 1 move without buffer
'        0 move in buffer mode
' Px  Axis value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'       0 The movement is not inserted in the buffer
'         in this case, is necessary reload the movement
'*************************************************************
function Move_Axis(Vel as long, Flg as char, Px as long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Move_Axis=pos1.moveto(Vel, Flg, Px)
endfunction
'*************************************************************
' Set ACC
' Value Acc value in count
'*************************************************************
function Acc_Axis(Value as long) as void
      pos1.acc=Value
endfunction
'************************
' Error check
' CanOpen node
'************************
function cancfgerr(node as int,err as uchar) as void
Node_1_Error=err ' copy the code error
endfunction
'************************
' Close init CanOpen
'************************
function close_cancfgerr() as void
endfunction
'************************
' Custom error init
' CanOpen node
'************************
function open_cancfgerr(nodes as int) as void
' Reset node 1 status error
Node_1_Error=0
endfunction
```

Code in Init Task PLC




```
pos1.msof=10000 ' motor 10000 i/rev
pos1.dsof=5000 ' 5 mm per revolution motor
```

Code in Task PLC



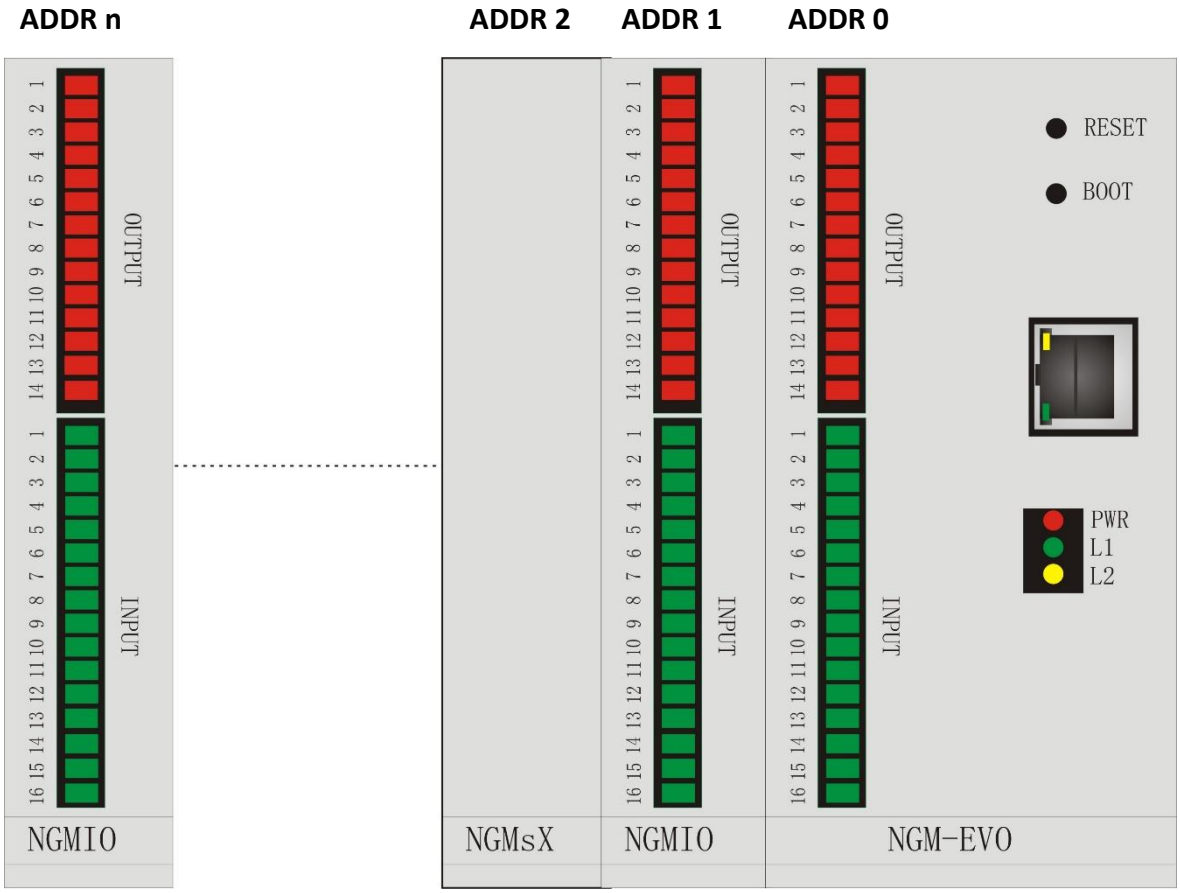

```
DigitalInputs=ng_di(0) ' read digital inputs
pos1.ext_fcz=Fc_Home ' home input
```

Example Download

10NGMIO-NGMsX Addressing

Inside to NGMEVO, can be included, expansions boards NGMIO and NGMsX
These expansions boards, are managed by VTB functions with phisical address from 0 to 7
This address is automatically assigned by local position in the BUS
The expansion board more near to NGMEVO CPU, takes the address 0, the next, address 1 etc.
The expansion at ADDR 0 is always included.

Address	Expansion Nr
0	Board 0 (near to NGMEVO)
1	Board 1
2	Board 2
3	Board 3
4	Board 4
5	Board 5
6	Board 6
7	Board 7



11 Digital I/O on NGMIO

The NGIO expansions boards, allows to use 16 digital inputs and 14 digital outputs, management by VTB functions. About addressing see chapr. 8

11.1 NG_DI – Read Digital Inputs

Read the Digital Inputs state.
The Digital Inputs are bit mapped – from 0 to 15

Input	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Syntax

NG_DI(CardNumber **as Char**) **as uint**

Parameters

CardNumber Expansion number (from 0 to 7 see chapr. 8)

Return Value

Uint Value - 16 inputs bit mapped
bit = 1 → Input ON
bit = 0 → Input OFF

11.2 NG_DO – Writ Digital Outputs

Writes the digital outputs state
The Digital Outputs are bit mapped – from 0 to 14

Output		14	13	12	11	10	9		8	7	6	5	4	3	2	1
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

WARNING
BIT 8 AND 15 ARE NOT USED

Syntax

NG_DO(CardNumber **as Char**, StatoOutputs **as Uint**) **as void**

Parameters

CardNumber Expansion number (from 0 to 7 see chapr. 8)

StatoOutputs Output state
bit = 1 → Output ON
bit = 0 → Output OFF

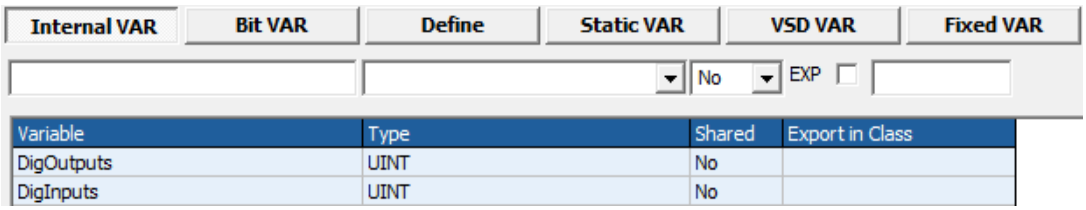
11.3 Example Digital I/O

In the next example, are managed the Digital I/O in the following mode:

UPDATING I/O IN TASK PLC

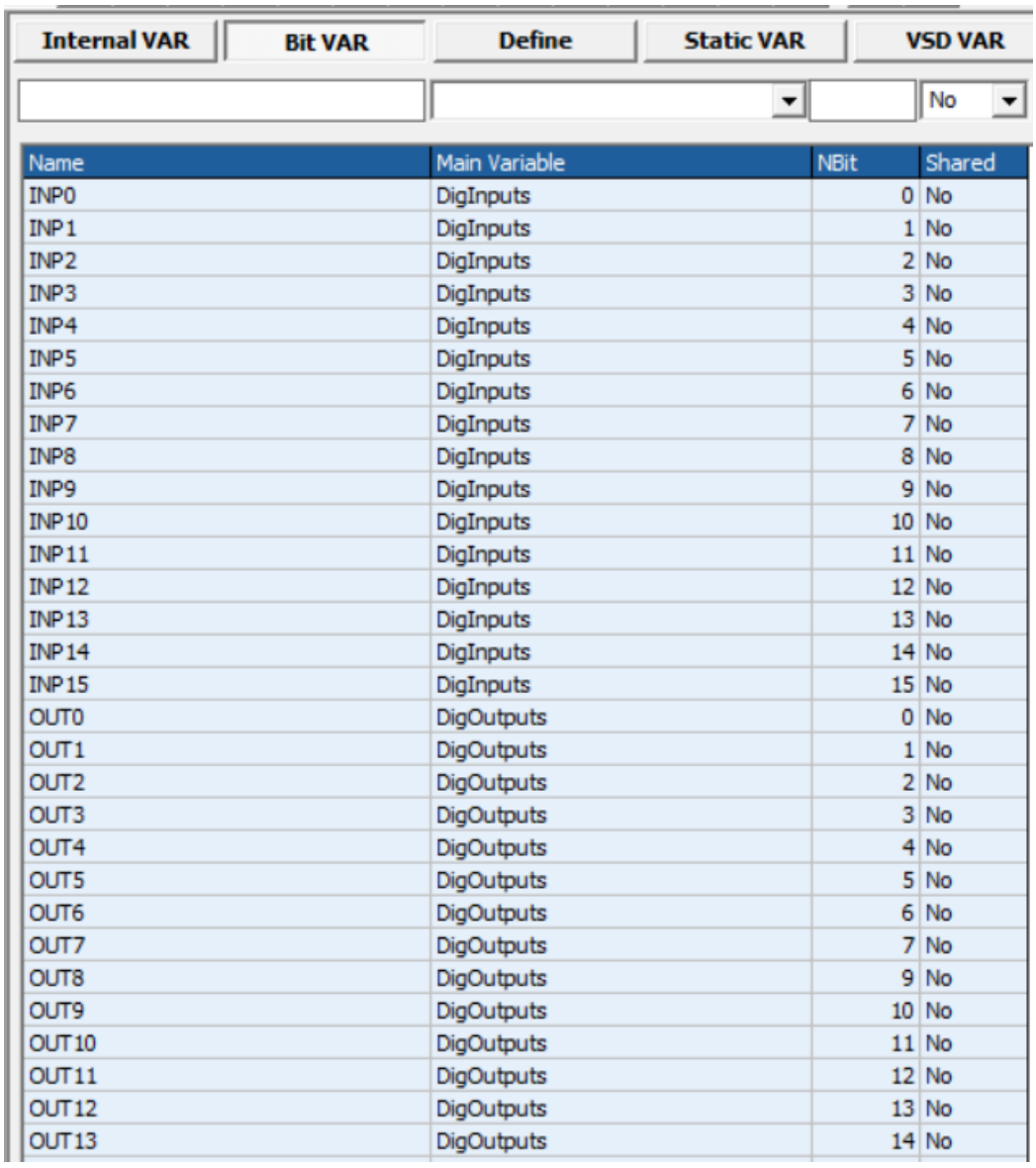
Management I/O in bit mode. The first 3 inputs are copied in the first 3 outpts

Variables used



Variable	Type	Shared	Export in Class
DigOutputs	UINT	No	
DigInputs	UINT	No	

BIT Used



Name	Main Variable	NBit	Shared
INP0	DigInputs	0	No
INP1	DigInputs	1	No
INP2	DigInputs	2	No
INP3	DigInputs	3	No
INP4	DigInputs	4	No
INP5	DigInputs	5	No
INP6	DigInputs	6	No
INP7	DigInputs	7	No
INP8	DigInputs	8	No
INP9	DigInputs	9	No
INP10	DigInputs	10	No
INP11	DigInputs	11	No
INP12	DigInputs	12	No
INP13	DigInputs	13	No
INP14	DigInputs	14	No
INP15	DigInputs	15	No
OUT0	DigOutputs	0	No
OUT1	DigOutputs	1	No
OUT2	DigOutputs	2	No
OUT3	DigOutputs	3	No
OUT4	DigOutputs	4	No
OUT5	DigOutputs	5	No
OUT6	DigOutputs	6	No
OUT7	DigOutputs	7	No
OUT8	DigOutputs	9	No
OUT9	DigOutputs	10	No
OUT10	DigOutputs	11	No
OUT11	DigOutputs	12	No
OUT12	DigOutputs	13	No
OUT13	DigOutputs	14	No

Code in Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
1   OUT0=INP0  ' copy input 0 on outpus 0
2   OUT1=INP1  ' copy input 1 on outpus 1
3   OUT2=INP2  ' copy input 2 on outpus 2
```

```
OUT0=INP0 ' copy input 0 on outpus 0
OUT1=INP1 ' copy input 1 on outpus 1
OUT2=INP2 ' copy input 2 on outpus 2
DigInputs=ng_di(0) ' udpate digital inputs
ng_do(0,DigOutputs) ' update digital outputs
```

Example Download

12 Analog Outputs and relè outputs NGMsX

The expansion boards NGMsX, manage 2 Analog Outputs +/- 10 V 12 bit and 2 relè outputs up to 1 A.

12.1 NG_DAC – Write Analog Outputs NGMsX

This function allows to update the analog outputs of each channel equipped in the NGMEVO expansions **NGMsX**
These expansions have a digital to analog converter at 12 bit, with a range of +/-10V. Therefore a value of +2047 corresponds to 10V in output, a value of -2047 corresponds to -10V.
The selection of the channel is made by an index from 0 to 7, each expansion manages two channels:

Index Channel	Expansion Addr
0	Board 0
1	
2	Board 1
3	
4	Board 2
5	

The maximum number of analog outputs is 6 (max 3 NGMsX boards).

Syntax

NG_DAC(Channel as Char, Val as Long) as void

Parameters

Channel Number of channel (from 0 to 5)
val Analog output value (from -2047 to +2047)

12.2 NG_DAC_CAL - CALIBRATION OF THE ANALOG OUTPUT OFFSET

This function allows to calibrate the OFFSET of the analog outputs. Usually it can be occur that the analog output has a little value of voltage (OFFSET) in the order of mV also if zero has been set. With ***ng_dac_cal*** we can null this voltage setting a value opposite to the offset one. Remind that for each unit the output value will be about 4mV.

Syntax

NG_DAC_CAL(Channel as Char, Offset as Long) as void

Parameters

Channel Numero Canale (from 0 to 5)
Offset OFFSET value (from -2047 to +2047)

WARNING

THE OFFSET VALUE ISN'T SAVED AND IT MUST BE SET AT EACH TURN-ON.
SAVE THIS VALUE IN NGMEVO FRAM MEMORY

12.3 NG_RELE - RELE' on NGMsX

This function allows to update the two RELAIS equipped in each expansion card **NGMsX**.
Usually these RELAIS are connected to the input ENABLE of the SERVO DRIVER but they can be managed for any applications. The channel selection is made as for the reading of encoders.

Index Channel	Expansion Addr
0	Board 0
1	
2	Board 1
3	
4	Board 2
5	

Syntax

NG_RELE(Channel **as Char**, State **as Char**) **as void**

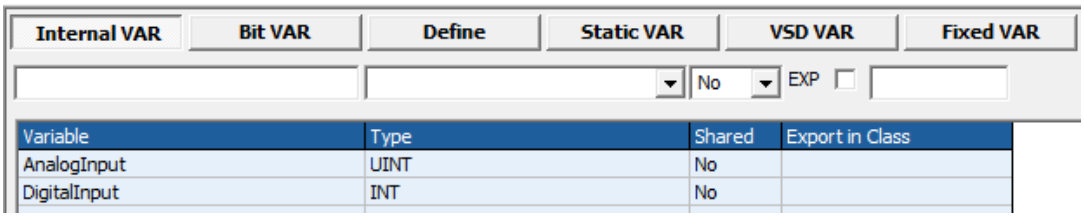
Parameters

Channel Number of channel (from 0 to 5)
State State of the relay:
0 OFF (contact opened)
1 ON (contact closed)

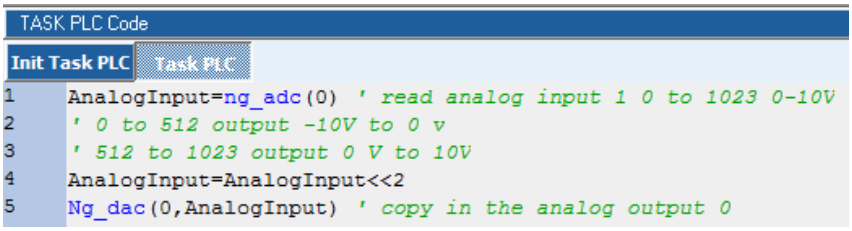
12.4 Example Analog Outputs and relè outputs

In the next examople, are managed the analog outputs and relè
Is read the analog input 0 and copied in the analog output 0.
The digital input 1 is copied in the relè output

Variables used



Code in Task PLC




```
AnalogInput=ng_adc(0) ' read analog input 1 0 to 1023 0-10V
' -2047 0 output -10V to 0 v
' 0 2048 output 0 V to 10V
AnalogInput=AnalogInput<<2
Ng_dac(0,AnalogInput) ' copy in the analog output 0
DigitalInput=ng_di(0) ' read digital input
if DigitalInput & 1 ' test input 1
      ng_rele(0,1) ' set rele'
else
      ng_rele(0,0) ' reset rele'
endif
```

Example Download

13 Encoder and Index Read NGMsX

The expansions boards NGMsX, allows to use 2 incremental channels encoder line and 2 Index zero encoder.

13.1 NG_ENC – READ CHANNEL ENCODER

This function allows to read the quadrature encoder input of each channel equipped on the expansion card **NGMsX.** The resolution is 32 bits. This function read only the increment which will be added to a variable passed by its pointer. Therefore the real encoder counter will be contained in a variable defined in the application and it will can be zeroed in any time. For a correct processing of the encoders we recommend to use this function only in TASK PLC and then use it at the occurrence.
The selection of the channel is made by an index from 0 to 15, each expansion manages two channels:

Index Channel	Expansion Addr
0	Board 0
1	
2	Board 1
3	
4	Board 2
5	

Syntax

NG_ENC(Channel as Char, Value as *Long) as void

Parameters

Channel Number of channel (from 0 to 5) **(max 3 NGMsX boards).**
Value Pointer to a long variable where will be contained the counter

WARNING
FOR A SINCRONOUS FUNCTION, USE NG_ENC ONLY IN TASK PLC

13.2 NG-T0 - ZERO INDEX OF ENCODER

This function allows to read the state of the zero index input of each encoder channel equipped in the expansion card **NG-IO**. The channel selection is made as for the reading of encoders.

Syntax

NG_T0(Channel as Char) as char

Parameters

Channel Number of channel (from 0 to 5)

Return Value

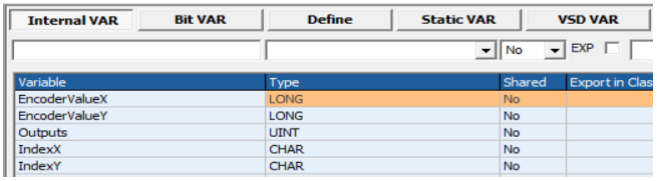
State of the index input:

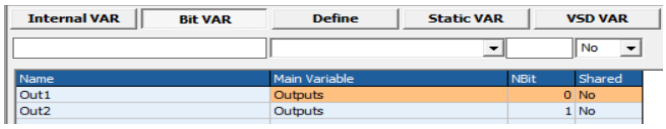
0 OFF
1 ON

13.3 Example Read Encoder NGMsX and Index

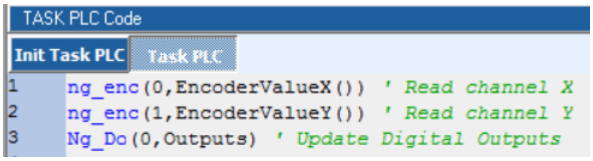
In this example, are read 2 channel encoder and 2 Index to NGIO Addr 0

Variables used



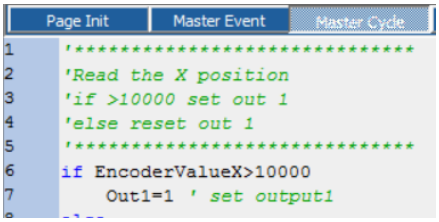


Code in Task PLC




```
ng_enc(0,EncoderValueX()) ' Read channel X
ng_enc(1,EncoderValueY()) ' Read channel Y
Ng_Do(0,Outputs) ' Update Digital Outputs
```

Codice in Task MAIN




```
'*******************************
'Read the X position
'if >10000 set out 1
'*******************************
if EncoderValueX>10000
      Out1=1 ' set output1
else
      Out1=0 ' reset output1
endif
'*******************************
'Read the Y position
'if >5000 set out 2
'*******************************
if EncoderValueY>5000
      Out2=1 ' set output2
else
      Out2=0 ' reset output2
endif
IndexX=ng_t0(0) 'read index X
IndexY=ng_t0(1) 'read index Y
```

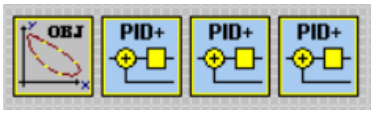
Example Download

13.4 Example Analog Axes in Interpolation Mode

In the following example, are managed, 3 Analog Axes +/- 10V with encoder Loop and PID filter. In linear interpolation.

WARNING:
ATTENTION:
All speed are managed in mm/min if setted the following parameters
RapX,RapY,RapZ
All axes target positions are managed in micron (0.001 mm) if setted the following parameters
RapX,RapY,RapZ

Objects used:



Motor Control → CobjInterpola → Interpolatore

Project Explorer

Project | Objects | Functions | Properties | Tables

Interp

Property | Events

Property	Value
Nome	Interp
Left	15
Top	10
N.assi	3
N.tratti	16
Vper	1024
Div. Vper	1024
Abilita arcto	1

Motor Control Plus→ CPidPlus → Pid NG

Property	Value
Nome	PidX
Left	55
Top	10
EnablePid	False
Kp	10
Ki	0
Kv	0
Kd	0
Err_Sat	10000
NG ENC CHANNEL	0
NG DAC CHANNEL	0
ENABLE KP	True
ENABLE KI	True
ENABLE KV	True
ENABLE KD	False
Divisore	100
Dir	1
ServoErr	10000
TServoErr	1000
EnableDelay	50

Property	Value
Nome	PidY
Left	100
Top	10
EnablePid	False
Kp	10
Ki	0
Kv	0
Kd	0
Err_Sat	10000
NG ENC CHANNEL	1
NG DAC CHANNEL	1
ENABLE KP	True
ENABLE KI	True
ENABLE KV	True
ENABLE KD	False
Divisore	100
Dir	1
ServoErr	10000
TServoErr	1000
EnableDelay	50

Property	Value
Nome	PidZ
Left	145
Top	10
EnablePid	False
Kp	10
Ki	0
Kv	0
Kd	0
Err_Sat	10000
NG ENC CHANNEL	2
NG DAC CHANNEL	2
ENABLE KP	True
ENABLE KI	True
ENABLE KV	True
ENABLE KD	False
Divisore	100
Dir	1
ServoErr	10000
TServoErr	1000
EnableDelay	50

Are managed the following functions:

Wait_Move – Axes state movement
Parameters *No*
Return *1 Axes in movement*
0 Axes stop

Move_Axes – Move the Axes in linear interpolation
Parameters *Vel → Feed Axes in mm/min*
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
(Stop Axes only if edge > SGLP)
Px,Py,Pz → Axes target values in 0.001 mm
Return *0 Movement inserted in the buffer – buffer empty*
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axes – Set interpolation Acceleration
Parameters *Value → Value in count per TAU*
Return *No*

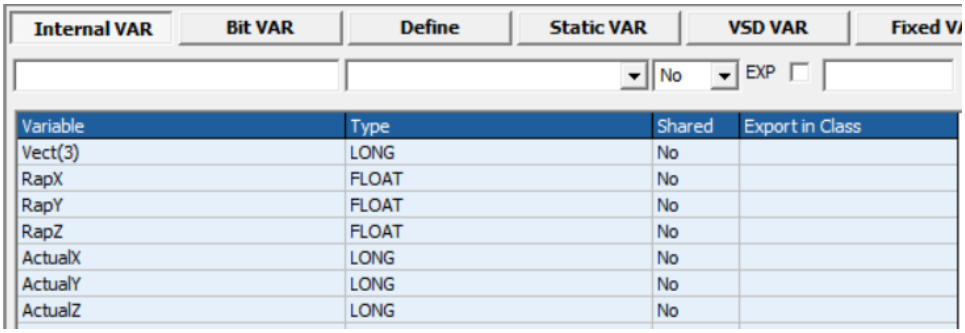
Stop_Axes – Stop Axes
Parameters *No*
Return *No*

Enable_Axis_X_Y_Z – Enable the Axes control and preset at value 0
Parameters *No*
Return *No*

Disable_Axis_X_Y_Z – Disable the Axes control
Parameters *No*
Return *No*

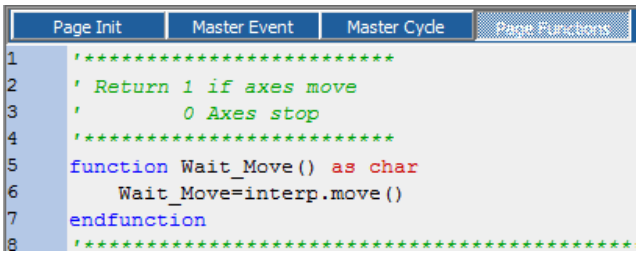
Test_Following_Error – Test axes following error
If errror, disable all axes
Parameters *No*
Return *No*

Variables used



Variable	Type	Shared	Export in Class
Vect(3)	LONG	No	
RapX	FLOAT	No	
RapY	FLOAT	No	
RapZ	FLOAT	No	
ActualX	LONG	No	
ActualY	LONG	No	
ActualZ	LONG	No	

Code in Main Page Functions

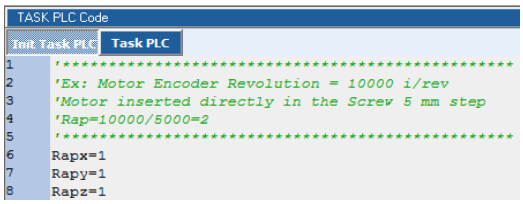



```
'*************************
' Return 1 if axes move
'        0 Axes stop
'*************************
function Wait_Move() as char
      Wait_Move=interp.move()
endfunction

'*********************************************************
' Move Axes
' Vel= interp vel Axes in mm/min
' Flg if 1 move without buffer
'        0 move in buffer mode
' Px,Py,Pz Axes value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'       0 The movement is not inserted in the buffer
'         in this case, is necessary reload the movement
'*********************************************************
function Move_Axes(Vel as long, Flg as char, Px as long, Py as long,Pz as
long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Vect(0)=Px
      Vect(1)=Py
      Vect(2)=Pz
      Move_Axes=interp.moveto(Vel, Flg, Vect())
endfunction

'*********************************************************
' Set ACC
' Value Acc value in count
'*********************************************************
function Acc_Axes(Value as long) as void
      interp.acc=Value
```

```
endfunction



'*************************************************************
' Stop Axes
'*************************************************************
function Stop_Axes() as void
      interp.stop()
endfunction
'********************
' Axis X enable
'********************
function Enable_X() as void
PidX.enablepid=0
'Preset Axis X 0, not change y,z
Vect(0)=0
Vect(1)=interp.pc(1)
Vect(2)=interp.pc(2)
interp.preset(Vect())
PidX.posr=0
'enable axis
PidX.enablepid=1
PidX.enable() ' closes the rele' on NGIO
endfunction
'********************
' Axis X Disable
'********************
function Disable_X() as void
Pidx.disable()
endfunction
'********************
' Axis Y enable
'********************
function Enable_Y() as void
PidY.enablepid=0
'Preset Axis Y 0, not change X,z
Vect(0)=interp.pc(0)
Vect(1)=0
Vect(2)=interp.pc(2)
interp.preset(Vect())
PidY.posr=0
'enable axis
PidY.enablepid=1
PidY.enable() ' closes the rele' on NGIO
endfunction
'********************
' Axis Y Disable
'********************
function Disable_Y() as void
PidY.disable()
endfunction
'********************
' Axis Z enable
'********************
function Enable_Z() as void
PidZ.enablepid=0
```

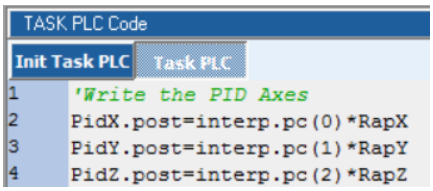
```
'Preset Axis Z 0, not change X,Y
Vect(0)=interp.pc(0)
Vect(1)=interp.pc(1)
Vect(2)=0
interp.preset(Vect())
PidZ.posr=0
'enable axis
PidZ.enablepid=1
PidZ.enable() ' closes the rele' on NGIO
endfunction

'********************
' Axis Z Disable
'********************
function Disable_Z() as void
PidZ.disable()
endfunction
'****************************
' Test following error
' Disable all Axes if error
'****************************
function Test_Following_Error()as void
dim Error as char
error=0
if PidX.err=1 ' test X
      error=1
endif
if PidY.err=1 ' test Y
      error=1
endif
if PidZ.err=1 ' test Z
      error=1
endif
if error=1 'if serror disable all motor
      Disable_X()
      Disable_Y()
      Disable_Z()
endif
endfunction
```

Code in Init Task PLC




```
'************************************************
'Ex: Motor Encoder Revolution = 10000 i/rev
'Motor inserted directly in the Screw 5 mm step
'Rap=10000/5000=2
'************************************************
Rapx=1
Rapy=1
Rapz=1
```

Code in Task PLC




```
'Write the PID Axes
PidX.post=interp.pc(0)*RapX
PidY.post=interp.pc(1)*RapY
PidZ.post=interp.pc(2)*RapZ
'read analog 0 and set the Vper %
interp.vper=ng_adc(0)
' copy the axes values
' for ex: display in HMI
' value in 0.001 mm
ActualX=PidX.posr ' read actual position X
ActualY=PidY.posr ' read actual position Y
ActualZ=PidZ.posr ' read actual position Z
```

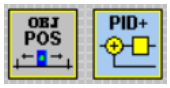
Example Download

13.5 Example Analog Axis in Position Mode

In the following example, are management, a CanOpen Axis by VTB OBJECT
See doc Vtb Object Guide for more informations.

WARNING:
All speed are managed in mm/min if setted the following parameters:
MSOF e DSOF
All axes target positions are managed in micron (0.001 mm) if setted the following parameters:
MSOF e DSOF

Objects used:



Motor Control Plus → CobjPos → Posizionatore

Property	Value
Nome	Pos1
Left	25
Top	30
N.TRATTI	8
Vper	1024
Div. Vper	1024
AccQstop	10
Acc	5
RzeroMode	1
RzeroOffset	0
RzeroPreset	0
RzeroVel	10
RzeroVelf	5
RzeroAcc	10
Msof	10000
Dsof	5000
LimitN	-99999999
LimitP	99999999
Gioco	0
Vgioco	1
MsofV	1
DsofV	1
RZERO ENABLE	True
AXIS TYPE	4
VTB AXIS OBJECT	PidX
PDO NAME	0
STEP CHANNEL	0
STEP NODE	1

Motor Control Plus→ CPidPlus → Pid NG

Property	Value
Nome	PidX
Left	75
Top	30
EnablePid	False
Kp	10
Ki	0
Kv	0
Kd	0
Err_Sat	10000
NG ENC CHANNEL	0
NG DAC CHANNEL	0
ENABLE KP	True
ENABLE KI	True
ENABLE KV	True
ENABLE KD	False
Divisore	100
Dir	1
ServoErr	10000
TServoErr	1000
EnableDelay	50

Are managed the following functions:

Wait_Move – Axis state movement
Parameters *No*
Return *1 Axis in movement*
0 Axes stop

Move_Axis – Move the Axis
Parameters *Vel → Feed Axes in mm/min*
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
Px, → Axes target values in 0.001 mm
Return *0 Movement inserted in the buffer – buffer empty*
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axis – Set Acceleration
Parameters *Value → Value in count per TAU*
Return *No*

Stop_Axis – Stop Axes
Parameters *No*
Return *No*

Enable – Enable the Axis control and preset at value 0
Parameters *No*
Return *No*

Disable – Disable the Axes control
Parameters *No*
Return *No*

StartHome – Start homing - Vel in pos1.rzerovel and pos1.rzerovelf
Parameters *No*
Return *No*

CheckHome – Check homing state
Parameters *No*
Return *1 homing finished*

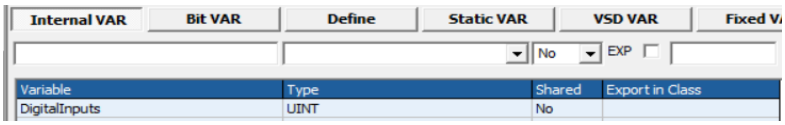
StopHome – Stop homing

Parameters ***No***
Return ***No***

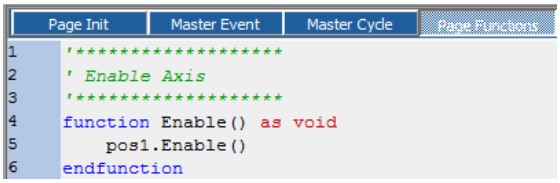
Test_Following_Error – Test axis following error
If errror, disable axis

Parameters ***No***
Return ***No***

Variables used



Code in Main Page Functions




```
'*******************
' Enable Axis
'*******************
function Enable() as void
      pos1.Enable()
endfunction
'*******************
' Disable Axis
'*******************
function Disable() as void
      pos1.Disable()
endfunction
'*******************
' Preset Axis
'*******************
function Preset(Val as long) as void
      pos1.Preset(Val)
endfunction
'*************************
' Return 1 if axis move
'         0 Axis stop
'*************************
function Wait_Move() as char
      Wait_Move=pos1.move()
endfunction
'*******************
' Axis Stop Move
'*******************
function Stop() as void
      pos1.Stop()
endfunction
'*****************************
' Start Homing
' Homing input see in task plc
'*****************************
function StartHome() as void
      pos1.StartHome()
endfunction
'*****************************
' Check if homing finished
' Return 1 if finished
'*****************************
function CheckHome() as char
      CheckHome=pos1.status_home
endfunction
```

```
'*****************************
' Stop home function
'*****************************
function StopHome() as void
      pos1.StopHome()
endfunction
'************************************************************
' Move Axis
' Vel=  vel Axis in mm/min
' Flg if 1 move without buffer
'         0 move in buffer mode
' Px  Axis value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'        0 The movement is not inserted in the buffer
'          in this case, is necessary reload the movement
'************************************************************
function Move_Axis(Vel as long, Flg as char, Px as long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Move_Axis=pos1.moveto(Vel, Flg, Px)
endfunction
'************************************************************
' Set ACC
' Value Acc value in count
'************************************************************
function Acc_Axis(Value as long) as void
      pos1.acc=Value
endfunction

'***************************
' Test following error
' Disable Axis
'***************************
function Test_Following_Error() as void
if PidX.err=1 ' test Axis
      Disable()
endif
endfunction
```

Code in Init Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
1   pos1.msof=10000 ' motor 10000 i/rev
2   pos1.ext_fcz=Fc_Home ' home input
```

```
pos1.msof=10000 ' motor 10000 i/rev
pos1.dsof=5000 ' 5 mm per revolution motor
```

Code in Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
1   DigitalInputs=ng_di(0) ' read digital inputs
2   pos1.ext_fcz=Fc_Home ' home input
```

```
DigitalInputs=ng_di(0) ' read digital inputs
pos1.ext_fcz=Fc_Home ' home input
```

Example Download

14 PULSE/DIR channels on NGMEVO and NGMsX

The NGMEVO can use 4 channels PULSE/DIR on CPU and up to 6 channel PULSE /DIR on NGMsX expansion. The difference between NGMEVO channels and NGMsX channels is the maximum clock used:

NGMEVO	***clock position mode***	***400 Khz total***
NGMEVO	***clock interpolation mode***	***125 Khz total***
NGMsX	***clock position mode***	***500 Khz per channel***
NGMsX	***clock interpolation mode***	***500 Khz per channel***

CHANNELS DISTRIBUTION NGMEVO - NGMsX

Canale 0	**NGMEVO**
Canale 1	**NGMEVO**
Canale 2	**NGMEVO**
Canale 3	**NGMEVO**
Canale 4	**FIRST EXPANSION NGMsX (only interpolation mode)**
Canale 5	**FIRST EXPANSION NGMsX (only interpolation mode)**
Canale 6	**SECOND EXPANSION NGMsX (only interpolation mode)**
Canale 7	**SECOND EXPANSION NGMsX (only interpolation mode)**
Canale 8	**THIRD EXPANSION NGMsX (only interpolation mode)**
Canale 9	**THIRD EXPANSION NGMsX (only interpolation mode)**

.

14.1 PP_STEP – Generating STEP/DIR signals

This function, is the primitive that allows the generation STEP and DIR signal on the specified channel. Generally it is used, by objects that allows to ***"Ramp and Position"*** generator.

Syntax

PP_STEP(Channel **as Char**, Value **as Long**) **as void**

Parameters

Channel Number of the STEP/DIR channel (see channels distribution)

Value Absolute value of the position of the step/dir axis

WARNING

THIS FUNCTION MUST BE INSERTED IN TASK_PLC

14.2 PP_PRESET – PRESET OF STEP/DIR POSITION

This function updates the current position of a step/dir channel.

Syntax

PP_PRESET(Channel **as Char**, Value **as Long**) **as void**

Parameters

Channel Numero del canale STEP/DIR
Value Valore della posizione che assumerà il l'asse step/dir

WARNING
FOR A CORRECT AXES PRESET, SEE THE EXAMPLE USED IN THE INTERPOLATOR OR POSITIONER

14.3 PP_GETPOS – READING OF ACTUAL POSITION

This function reads the actual position of a step/dir channel. **The value will correspond to the DOUBLE of the real position**.

Syntax

PP_GETPOS(Channel **as Char**) **as long**

Parameters

Channel Number of the STEP/DIR channel

Return Value

Long *Actual position x 2*

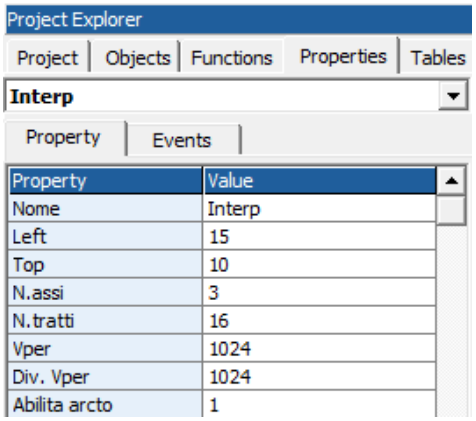
14.4 Example STEP/DIR Axes in Interpolation Mode

In the following example, are managed, 3 STEP/DIR Axes In linear interpolation.

WARNING:
ATTENTION:
All speed are managed in mm/min if setted the following parameters
RapX,RapY,RapZ
All axes target positions are managed in micron (0.001 mm) if setted the following parameters
RapX,RapY,RapZ

Objects used:

Motor Control → CobjInterpola → Interpolatore



Are managed the following functions:

Wait_Move – Axes state movement
Parameters ***No***
Return ***1 Axes in movement***
0 Axes stop

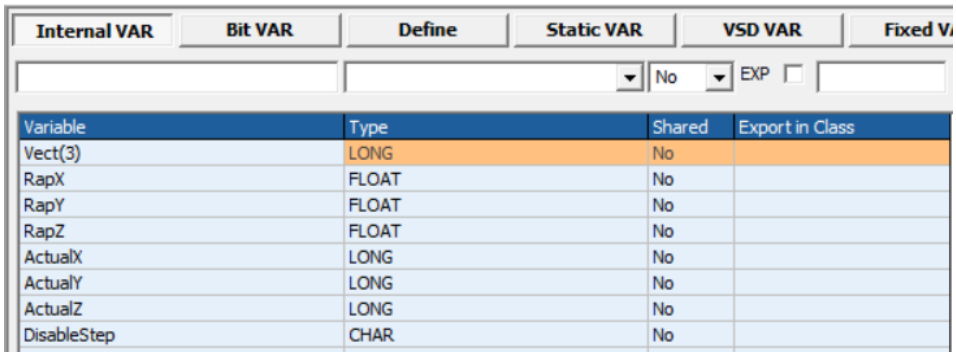
Move_Axes – Move the Axes in linear interpolation
Parameters ***Vel → Feed Axes in mm/min***
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
(Stop Axes only if edge > SGLP)
Px,Py,Pz → Axes target values in 0.001 mm
Return ***0 Movement inserted in the buffer – buffer empty***
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axes – Set interpolation Acceleration
Parameters ***Value → Value in count per TAU***
Return ***No***

Stop_Axes – Stop Axes
Parameters ***No***
Return ***No***

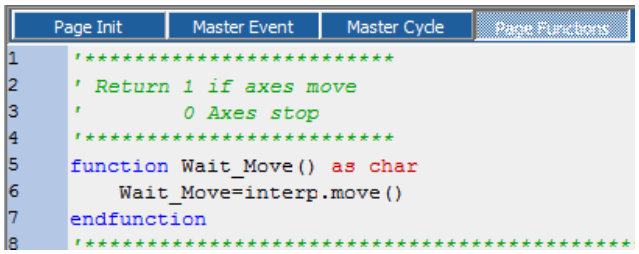
Enable_Axis_X_Y_Z – Enable the Axes control and preset at value 0
Parameters ***No***
Return ***No***

Variables used



Variable	Type	Shared	Export in Class
Vect(3)	LONG	No	
RapX	FLOAT	No	
RapY	FLOAT	No	
RapZ	FLOAT	No	
ActualX	LONG	No	
ActualY	LONG	No	
ActualZ	LONG	No	
DisableStep	CHAR	No	

Code in Main Page Functions




```
'*************************
' Return 1 if axes move
'        0 Axes stop
'*************************
function Wait_Move() as char
      Wait_Move=interp.move()
endfunction

'*********************************************************
' Move Axes
' Vel= interp vel Axes in mm/min
' Flg if 1 move without buffer
'        0 move in buffer mode
' Px,Py,Pz Axes value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'       0 The movement is not inserted in the buffer
'         in this case, is necessary reload the movement
'*********************************************************
function Move_Axes(Vel as long, Flg as char, Px as long, Py as long,Pz as
long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Vect(0)=Px
      Vect(1)=Py
      Vect(2)=Pz
      Move_Axes=interp.moveto(Vel, Flg, Vect())
endfunction

'*********************************************************
' Set ACC
' Value Acc value in count
'*********************************************************
function Acc_Axes(Value as long) as void
      interp.acc=Value
```

```
endfunction



'*************************************************************
' Stop Axes
'*************************************************************
function Stop_Axes() as void
      interp.stop()
endfunction
'********************
' Axis X enable
'********************
function Enable_X() as void
DisableStep=1
'Preset Axis X 0, not change y,z
Vect(0)=0
Vect(1)=interp.pc(1)
Vect(2)=interp.pc(2)
interp.preset(Vect())
'enable axis
DisableStep0=0
endfunction
'********************
' Axis Y enable
'********************
function Enable_Y() as void
DisableStep=1
'Preset Axis Y 0, not change X,z
Vect(0)=interp.pc(0)
Vect(1)=0
Vect(2)=interp.pc(2)
interp.preset(Vect())
'enable axis
DisableStep=0
endfunction
'********************
' Axis Z enable
'********************
function Enable_Z() as void
DisableStep=1
'Preset Axis Z 0, not change X,Y
Vect(0)=interp.pc(0)
Vect(1)=interp.pc(1)
Vect(2)=0
interp.preset(Vect())
PidZ.posr=0
'enable axis
DisableStep=0
endfunction
```

Code in Init Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
'************************************************
'Ex: Motor Encoder Revolution = 10000 i/rev
'Motor inserted directly in the Screw 5 mm step
'Rap=10000/5000=2
'************************************************
Rapx=1
Rapy=1
Rapz=1
```

Code in Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
if DisableStep=0 ' disable output step
    pp_step(0, interp.pc(0)*RapX)   'Update the X Axis
    pp_step(1, interp.pc(1)*RapY)   'Update the Y Axis
    pp_step(2, interp.pc(2)*RapZ)   'Update the Z Axis
endif
```

```
if DisableStep=0 ' disable output step
      pp_step(0, interp.pc(0)*RapX) 'Update the X Axis
      pp_step(1, interp.pc(1)*RapY) 'Update the Y Axis
      pp_step(2, interp.pc(2)*RapZ) 'Update the Z Axis
endif
'read analog 0 and set the Vper %
interp.vper=ng_adc(0)
' copy the axes values
' for ex: display in HMI
' value in 0.001 mm
ActualX=interp.pc(0) ' read actual position X
ActualY=interp.pc(1) ' read actual position Y
ActualZ=interp.pc(2) ' read actual position Z
```

Example Download

14.5 Example STEP/DIR Axis in Position Mode

In the following example, are management, a CanOpen Axis by VTB OBJECT
See doc Vtb Object Guide for more informations.

WARNING:
All speed are managed in mm/min if setted the following parameters:
MSOF e DSOF
All axes target positions are managed in micron (0.001 mm) if setted the following parameters:
MSOF e DSOF

Objects used:

Motor Control Plus → CobjPos → Posizionatore

Property	Value
Nome	Pos1
Left	25
Top	30
N.TRATTI	8
Vper	1024
Div. Vper	1024
AccQstop	10
Acc	5
RzeroMode	1
RzeroOffset	0
RzeroPreset	0
RzeroVel	10
RzeroVelf	5
RzeroAcc	10
Msof	10000
Dsof	5000
LimitN	-99999999
LimitP	99999999
Gioco	0
Vgioco	1
MsofV	1
DsofV	1
RZERO ENABLE	True
AXIS TYPE	2
VTB AXIS OBJECT	0
PDO NAME	0
STEP CHANNEL	0
STEP NODE	0

Are managed the following functions:

Wait_Move – Axis state movement
Parameters *No*
Return *1 Axis in movement*
0 Axes stop

Move_Axis – Move the Axis
Parameters *Vel → Feed Axes in mm/min*
Flg → Set to 1 for disable the movements buffer
(Stop axes at end trajectory)
Set to 0 for enable the movements buffer
Px, → Axes target values in 0.001 mm
Return *0 Movement inserted in the buffer – buffer empty*
1 Buffer full (you must repeat Move_Axes up to when buffer empty)

Acc_Axis – Set Acceleration
Parameters *Value → Value in count per TAU*
Return *No*

Stop_Axis – Stop Axes
Parameters *No*
Return *No*

Enable – Enable the Axis control and preset at value 0
Parameters *No*
Return *No*

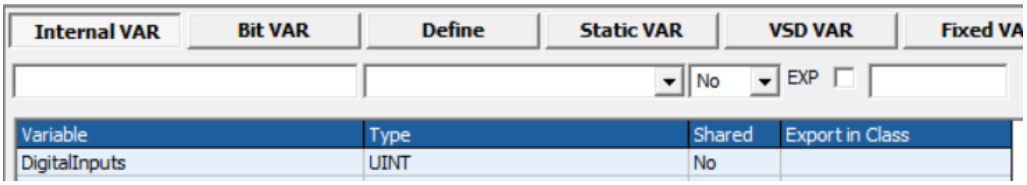
Disable – Disable the Axes control
Parameters *No*
Return *No*

StartHome – Start homing - Vel in pos1.rzerovel and pos1.rzerovelf
Parameters *No*
Return *No*

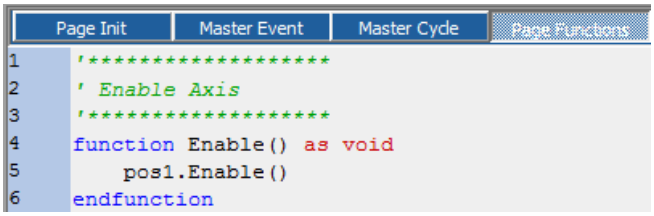
CheckHome – Check homing state
Parameters *No*
Return *1 homing finished*

StopHome – Stop homing
Parameters *No*
Return *No*

Variables used



Code in Main Page Functions




```
'******************
' Enable Axis
'******************
function Enable() as void
      pos1.Enable()
endfunction
'******************
' Disable Axis
'******************
function Disable() as void
      pos1.Disable()
endfunction
'******************
' Preset Axis
'******************
function Preset(Val as long) as void
      pos1.Preset(Val)
endfunction
'************************
' Return 1 if axis move
'         0 Axis stop
'************************
function Wait_Move() as char
      Wait_Move=pos1.move()
endfunction
'******************
' Axis Stop Move
'******************
function Stop() as void
      pos1.Stop()
endfunction
'****************************
' Start Homing
' Homing input see in task plc
'****************************
function StartHome() as void
      pos1.StartHome()
endfunction
'****************************
' Check if homing finished
```

```
' Return 1 if finished
'****************************
function CheckHome() as char
      CheckHome=pos1.status_home
endfunction
'****************************
' Stop home function
'****************************
function StopHome() as void
      pos1.StopHome()
endfunction
'*********************************************************
' Move Axis
' Vel=  vel Axis in mm/min
' Flg if 1 move without buffer
'        0 move in buffer mode
' Px  Axis value in 0.001 mm
'Return 1 if movement is inserted in the buffer
'       0 The movement is not inserted in the buffer
'         in this case, is necessary reload the movement
'*********************************************************
function Move_Axis(Vel as long, Flg as char, Px as long) as char
      Vel=Vel*TAU/60 ' Transform in mm/min
      Move_Axis=pos1.moveto(Vel, Flg, Px)
endfunction
'*********************************************************
' Set ACC
' Value Acc value in count
'*********************************************************
function Acc_Axis(Value as long) as void
      pos1.acc=Value
endfunction
```

Code in Init Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
1   pos1.msof=10000 ' motor 10000 i/rev
2   pos1.ext_fcz=Fc_Home ' home input
```

```
pos1.msof=10000 ' motor 10000 i/rev
pos1.dsof=5000 ' 5 mm per revolution motor
```

Code in Task PLC

TASK PLC Code

Init Task PLC | Task PLC

```
1   DigitalInputs=ng_di(0) ' read digital inputs
2   pos1.ext_fcz=Fc_Home ' home input
```

```
DigitalInputs=ng_di(0) ' read digital inputs
pos1.ext_fcz=Fc_Home ' home input
```

Example Download

15 Permanent Memory

The NGMEVO allows to use, the permanent memory on FRAM

15.1 Internal FRAM Memory

The internal FRAM Memory has a capacity 16 or 32 Kb.

15.1.1 IMS_READ – Read fram memory

Reads from the internal memory at address ADDR a number of byte as in NBYTE and writes them in the array pointed by Punt..

Syntax

IMS_READ(Punt **as *Char**, Addr **as Long**, Nbyte **as Long**) **as Char**

Parameters

Punt	Pointer to data buffer where read data will be saved
Addr	Start address in the reserved area of the device
Nbyte	Number of bytes to be read

Return Value

Char	0	No error
	<>0	Writing error

15.1.2 IMS_WRITE – Write fram memory

Writes in the internal FRAM at the address contained in ADDR, the data pointed by Punt for a total of NBYTE of data.

Syntax

IMS_WRITE(Punt **as *Char**, Addr **as Long**, Nbyte **as Long**) **as Char**

Parameters

Punt	Pointer to data buffer to be written
Addr	Start address in the reserved area of the device
Nbyte	Number of bytes to be written

Return value:

Char	0	No error
	<>0	Writing error

WARNING
THE FRAM CAN BE MANAGED BYTE TO BYTE AND NOT IN BLOCK MODE LIKE FRAM MEMORY

15.2 Example save/load in FRAM

In the following example, are saved and loaded by FRAM the values in a Long Vector. This example can be used for a machines parameters management .
Is used a Checksum (parameters values sum) and saved in the LAST position of array.
The Checksum is used to ensure, the parameters integrity

Are managed the following functions:

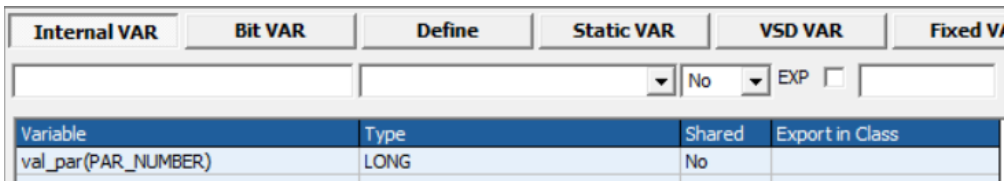
LoadPar – Load from FRAM the values

Parameters ***No***
Return ***0 OK***
1 Error FRAM
2 Error checksum

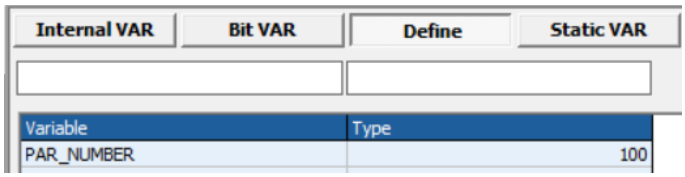
SavePar – Save in FRAM the values

Parameters ***No***
Return ***0 OK***
1 Error FRAM

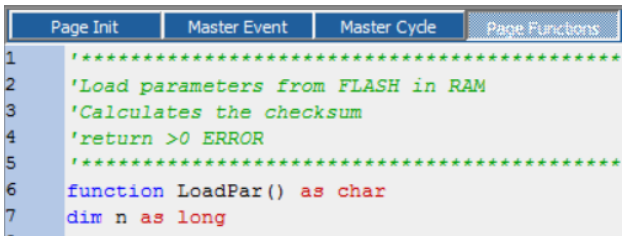
Variables used



DEFINE used



Code in Main Page Functions




```
'*************************************************
'Load parameters from FRAM in RAM
'Calculates the checksum
'return >0 ERROR
'*************************************************
function LoadPar() as char
dim n as long
dim ckl as long
dim ck as long
dim Ret as char
'PAR_NUMBER is number of parameters
```

```
'all parameters are in long
Ret=ims_read(val_par(),0,PAR_NUMBER*4) ' reads parameters from FRAM and
'puts in val_par vector


if Ret<>0
      'LOAD ERROR !!!!
      LoadPar=1 'return ERROR 1
      return
endif
ck=val_par(PAR_NUMBER) 'gets the check sum in last position
ckl=0
for n=0 to n<(PAR_NUMBER-1) 'calculates the checksum
      ckl=ckl+val_par(n)
next n
if ckl=0          'if all parameters are ZERO - chekcsum error
      ckl=ck+1
endif
if ckl<>ck
      'CheckSum ERROR
      LoadPar=2 'return ERROR 2
else
      LoadPar=0 'return OK
endif
endfunction

'*********************************
'Save the parameters in FRAM
'Return >0 ERROR
'*********************************
function SavePar() as char
dim ck as long
dim n as long
dim Ret as char
ck=0
for n=0 to n<(PAR_NUMBER)-1 'calculates the checksum
      ck=ck+val_par(n)
next n
val_par(PAR_NUMBER-1)=ck 'put the checksum
Ret=ims_write(val_par(),0,PAR_NUMBER*4) 'save the parameters
if Ret<>0
      'SAVE ERROR !!!!
      SavePar=1 'return ERROR 1
else
      SavePar=0 'return OK
endif
endfunction
```

Example Download

Index