SoMachine

Modbus RTU Communications - Read/Write Variables

Modbus_RW_Var.project

Example Guide

04/2012
The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2012 Schneider Electric. All rights reserved.
# Table of Contents

SAFETY INFORMATION .......................................................................................................................... 5

ABOUT THE BOOK ............................................................................................................................... 7

1. DESCRIPTION ................................................................................................................................. 13
   1.1. Presentation ............................................................................................................................. 13
   1.2. Main Features .......................................................................................................................... 15
   1.3. Functional Analysis of the Example ....................................................................................... 16
   1.4. Functions Used in this Example ............................................................................................ 18
   1.5. Hardware Installation .............................................................................................................. 20

2. DESCRIPTION OF THE EXAMPLE CONTENT .............................................................................. 22

3. CREATION OF THE PROJECT ....................................................................................................... 23

4. SERIAL LINE MODBUS MANAGER ............................................................................................. 26

5. LIBRARY MANAGER ....................................................................................................................... 30

6. ST, LD, OR CFC PROGRAM ............................................................................................................ 31
   6.1. ST Program ............................................................................................................................ 32
   6.2. LD Program ............................................................................................................................ 41
   6.3. CFC Program ........................................................................................................................ 50

7. CONFIGURATION OF THE ATV12 MODBUS SLAVE ................................................................. 57

8. RUNNING THE EXAMPLE ................................................................................................................. 62
   8.1. MAST Task Configuration ...................................................................................................... 62
   8.2. Downloading the Example to the Controller ......................................................................... 64
   8.3. Running the Example on the Controller ................................................................................. 67
Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠️ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

⚠️ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, can result in death or serious injury.

⚠️ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, can result in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and the installation, and has received safety training to recognize and avoid the hazards involved.
About the Book

At a Glance

Document Scope

This document describes one of the SoMachine examples.

Since the example described in this document is intended for learning purposes only, it must not be run, nor tested, on products that are part of a machine or process.

Validity Note

This document has been updated with the release of SoMachine V3.1.

The technical characteristics of the device(s) described in this manual also appear online. To access this information online:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to <a href="http://www.schneider-electric.com">www.schneider-electric.com</a></td>
</tr>
<tr>
<td>2</td>
<td>In the Search box on the home page, type a model number. Do not type any blank spaces in the model number. To get information on a grouping of similar modules, you can use the characters **; do not use dots or xx's.</td>
</tr>
<tr>
<td>3</td>
<td>Under All, click Products → Product Datasheets and select the model number that interests you.</td>
</tr>
<tr>
<td>4</td>
<td>To save or print a data sheet as a .pdf file, click Export to PDF.</td>
</tr>
</tbody>
</table>

The characteristics presented in this manual should be the same as those that appear online. In line with our policy of constant improvement we may revise content over time to improve clarity and accuracy. In the event that you see a difference between the manual and online information, use the online information as your reference.

Related Documents

<table>
<thead>
<tr>
<th>Title of Documentation</th>
<th>Reference Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modicon M238 Logic Controller Hardware Guide</td>
<td>EIO0000000016 (ENG);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000017 (FRE);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000018 (GER);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000019 (SPA);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000020 (ITA);</td>
</tr>
<tr>
<td></td>
<td>EIO0000000021 (CHS)</td>
</tr>
<tr>
<td>Title of Documentation</td>
<td>Reference Number</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M238 ExecLoader User Guide</td>
<td>EIO00000000374 (ENG); EIO0000000737 (FRE); EIO0000000738 (GER); EIO0000000739 (SPA); EIO0000000740 (ITA); EIO0000000741 (CHS)</td>
</tr>
<tr>
<td>Modicon M258 Logic Controller Hardware Guide</td>
<td>EIO0000000432 (ENG); EIO0000000433 (FRE); EIO0000000434 (GER); EIO0000000435 (SPA); EIO0000000436 (ITA); EIO0000000437 (CHS)</td>
</tr>
<tr>
<td>Modicon LMC058 Motion Controller Hardware Guide</td>
<td>EIO0000000438 (ENG); EIO0000000439 (FRE); EIO0000000440 (GER); EIO0000000441 (SPA); EIO0000000442 (ITA); EIO0000000443 (CHS)</td>
</tr>
<tr>
<td>SoMachine Modbus and ASCII Read/Write Functions PLC Communication Library Guide</td>
<td>EIO00000000361 (ENG); EIO0000000742 (FRE); EIO0000000743 (GER); EIO0000000744 (SPA); EIO0000000745 (ITA); EIO0000000746 (CHS)</td>
</tr>
<tr>
<td>Modicon M238 Logic Controller Programming Guide</td>
<td>EIO0000000384 (ENG); EIO0000000385 (FRE); EIO0000000386 (GER); EIO0000000387 (ITA); EIO0000000388 (SPA); EIO0000000389 (CHS)</td>
</tr>
<tr>
<td>Modicon M258 Logic Controller Programming Guide</td>
<td>EIO0000000402 (ENG); EIO0000000403 (FRE); EIO0000000404 (GER); EIO0000000405 (SPA); EIO0000000406 (ITA); EIO0000000407 (CHS)</td>
</tr>
<tr>
<td>Modicon LMC058 Motion Controller Programming Guide</td>
<td>EIO0000000408 (ENG); EIO0000000409 (FRE); EIO0000000410 (GER); EIO0000000411 (ITA); EIO0000000412 (SPA); EIO0000000413 (CHS)</td>
</tr>
<tr>
<td>Altivar 12 Variable speed drives for asynchronous motors User manual</td>
<td>BBV28581 (ENG); BBV28580 (FRE)</td>
</tr>
<tr>
<td>Altivar 12 Variable speed drives for asynchronous motors Modbus Communication Manual</td>
<td>BBV28590 (ENG)</td>
</tr>
<tr>
<td>ATV12 Communication Parameters</td>
<td>BBV51917 (ENG)</td>
</tr>
</tbody>
</table>
Product Related Information

This document and its related SoMachine project file focus on specific Functions and Function Blocks of the Schneider Electric libraries provided with SoMachine, and on specific features available in SoMachine if these features are related to these libraries. They are intended to help you with developing, testing, commissioning, and integrating applicative software of your own design on control systems.

It is intended for new SoMachine users who already have some degree of expertise in the design and programming of control systems.

⚠️ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠️ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines. ¹
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Before You Begin

The products specified in this document have been tested under actual service conditions. Of course, your specific application requirements may be different from those assumed for this and any related examples described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any examples specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

WARNING

REGULATORY INCOMPATIBILITY

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

The use and application of the information contained herein require expertise in the design and programming of automated control systems. Only the user or integrator can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or process, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, the user or integrator must also consider any applicable local, regional or national standards and/or regulations.

Some of the major software functions and/or hardware components used in the proposed architectures and examples described in this document cannot be substituted without significantly compromising the performance of your application. Further, any such substitutions or alterations may completely invalidate any proposed architectures, descriptions, examples, instructions, wiring diagrams and/or compatibilities between the various hardware components and software functions specified herein and in related documentation. You must be aware of the consequences of any modifications, additions or substitutions. A residual risk, as defined by EN/ISO 12100-1, Article 5, will remain if:

- it is necessary to modify the recommended logic and if the added or modified components are not properly integrated in the control circuit.
- you do not follow the required standards applicable to the operation of the machine, or if the adjustments to and the maintenance of the machine are not properly made (it is essential to strictly follow the prescribed machine maintenance schedule).
- the devices connected to any safety outputs do not have mechanically-linked contacts.
CAUTION

EQUIPMENT INCOMPATIBILITY

Read and thoroughly understand all device and software documentation before attempting any component substitutions or other changes related to the application examples provided in this document.

Failure to follow these instructions can result in injury or equipment damage.

Start-up and Test

Before using electrical control and automation equipment after design and installation, the application and associated functional safety system must be subjected to a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such testing be made and that enough time is allowed to perform complete and satisfactory testing.

<table>
<thead>
<tr>
<th>CAUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUIPMENT OPERATION HAZARD</td>
</tr>
<tr>
<td>• Verify that all installation and set up procedures have been completed.</td>
</tr>
<tr>
<td>• Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.</td>
</tr>
<tr>
<td>• Remove tools, meters and debris from equipment.</td>
</tr>
<tr>
<td>Failure to follow these instructions can result in injury or equipment damage.</td>
</tr>
</tbody>
</table>

Verify that the completed system, including the functional safety system, is free from all short-circuits and grounds, except those grounds installed according to local regulations. If high-potential voltage testing is necessary, follow the recommendations in equipment documentation to help prevent injury or equipment damage.
Operation and Adjustments

Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly installed and operated.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the hands and other parts of the body are free to enter the pinch points or other hazardous areas where serious injury can occur. Software products alone cannot protect an operator from injury. For this reason, the software cannot be substituted for or take the place of point-of-operation protection.

<table>
<thead>
<tr>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY</strong></td>
</tr>
<tr>
<td>• Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.</td>
</tr>
<tr>
<td>• Do not reach into machinery during operation.</td>
</tr>
<tr>
<td><strong>Failure to follow these instructions can result in death, serious injury, or equipment damage.</strong></td>
</tr>
</tbody>
</table>

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

**NOTE:** Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the examples and implementations suggested herein.

It is sometimes possible to adjust the equipment incorrectly and this may produce unsatisfactory or unsafe operation. Always use the manufacturer instructions as a guide to functional adjustments. Personnel who have access to these adjustments must be familiar with the equipment manufacturer instructions and the machinery used with the electrical equipment.

Only those operational adjustments actually required by the machine operator should be accessible to the operator. Access to other controls should be restricted to help prevent unauthorized changes in operating characteristics.

User Comments

We welcome your comments about this document. You can reach us by e-mail at techcomm@schneider-electric.com.
1. Description

1.1. Presentation

This example presents communications between a Modbus master and a Modbus slave using the Modbus RTU (Remote Terminal Unit) protocol. In this example, these exchanges are performed between an LMC058 motion controller (the Modbus master), and an ATV12 (Altivar 12) variable speed drive (the Modbus slave).

The controller's program is created using SoMachine software and the ATV12 variable speed drive is configured using the HMI of its front panel.

In this example, the purpose of these communications consists in performing a periodic read request of the ETA parameter (status word) of the ATV12, and three aperiodic read & write requests of several other parameters of the ATV12.

Related SoMachine project: **Modbus_RW_Var.project**

Supported SoMachine Languages:

- ✔ CFC
- ✔ ST
- ✔ LD
- □ IL
- □ FBD
- □ SFC

**Key features:** Modbus RTU communications for reading and writing data on a Modbus slave

**Requirements:** To use this example, the user must have:

- installed SoMachine V3.1 on a PC;
- run at least one Basic SoMachine example.

**NOTE:** Because it is required to run the communication functions used in this example's programs on the controller, do not run this example in SIMULATION mode.
NOTE: This example guide is also applicable to any M238 Logic Controller or M258 Logic Controller. This guide assumes that you are using a LMC038 Motion Controller, but it also describes the modifications required for using a M238 Logic Controller or M258 Logic Controller instead of this LMC038 Motion Controller.
1.2. Main Features

The main features of this example include:

- Configuration of a Modbus RTU serial connection using SoMachine.
- Addressing a Modbus slave using the **PLCCommunication** library.
- Periodic read request of one parameter of the ATV12 Modbus slave (ETA: status word) using the Modbus function #3 (Read Holding Registers).
- Aperiodic write request of one parameter of the ATV12 Modbus slave (LFR: frequency setpoint) using the Modbus function #6 (Write Single Register), followed by a read request of the very same parameter using the Modbus function #3 (Read Holding Registers).
- Aperiodic write request of two parameters of the ATV12 Modbus slave (ACC & DEC: acceleration & deceleration) using the Modbus function #16 (Write Multiple Registers), followed by a read request of the very same parameters using the Modbus function #3 (Read Holding Registers).
- Aperiodic read/write request of seven parameters of the ATV12 Modbus slave (SP2 to SP8: preset speeds 2 to 8) using the Modbus function #23 (Read/Write Multiple Registers).
- Use example of the data read and written by the controller on the ATV12 variable speed drive.
1. Description

1.3. Functional Analysis of the Example

The program described in this example, whatever its programming language, performs the following treatments:

Start

1-second delay

Result = OK?

Yes

1-second periodic Read of the ETA parameter (@3201) on the ATV12 drive

Result = OK?

No

ATV12 present: Yes AND Comm. error: No

ATV12 present: No OR Comm. error: Yes

End

Result = OK?

No

Yes

Converts LFR parameters (INT to WORD)

Converts ACC & DEC parameters (INT to WORD)

Converts SP2 to SP8 parameters (INT to WORD)

Converts LFR (WORD to INT)

Converts ACC & DEC (WORD to INT)

Converts SP2 to SP8 (WORD to UINT)

2. Start

1-second delay

Result = OK?

Yes

1-second periodic Read of the ETA parameter (@3201) on the ATV12 drive

Result = OK?

No

ATV12 present: Yes AND Comm. error: No

ATV12 present: No OR Comm. error: Yes

End

Result = OK?

No

Yes

Converts LFR parameters (INT to WORD)

Converts ACC & DEC parameters (INT to WORD)

Converts SP2 to SP8 parameters (INT to WORD)

Converts LFR (WORD to INT)

Converts ACC & DEC (WORD to INT)

Converts SP2 to SP8 (WORD to UINT)
1. Description

(1) In this example, the Modbus function #23 (Read/Write Multiple Registers) is used both to write SP2 to SP8 parameters and to read back the values of the very same parameters. Here, this is intended to check that these parameters have been updated on the ATV12 variable speed drive.

Depending on the Modbus slaves and/or the registers, it is possible that reading/writing the same registers using Modbus function #23 in this manner does not result in identical values. It depends on how the Modbus slave handles this command.

In this diagram, the [green boxes] indicate where the functions of the PLCCommunication library are used.
1. Description

1.4. Functions Used in this Example

The Functions (and Function Blocks) used in this example are listed below, grouped by library:

- **PLCCommunication** library (Schneider Electric)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDM</td>
<td>Convert a string into an address</td>
<td></td>
</tr>
<tr>
<td>READ_VAR</td>
<td>Reads data from a Modbus device</td>
<td></td>
</tr>
<tr>
<td>SINGLE_WRITE</td>
<td>Write a single internal register to a Modbus device</td>
<td></td>
</tr>
<tr>
<td>WRITE_READ_VAR</td>
<td>Read and write internal registers on a Modbus device</td>
<td></td>
</tr>
<tr>
<td>WRITE_VAR</td>
<td>Write data to a Modbus device</td>
<td></td>
</tr>
</tbody>
</table>

- **Standard** library (System)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TON</td>
<td>Timer function block: implement a turn-on delay</td>
<td>Function Blocks ➔ { } Standard ➔ Timer</td>
</tr>
</tbody>
</table>

- **Util** library (System)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Location in the Input Assistant</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLINK</td>
<td>Generates a pulsating signal</td>
<td>Function Blocks ➔ { } Util ➔ Signals</td>
</tr>
</tbody>
</table>
Please refer to the SoMachine online help.

NOTE: In the remaining sections of this document, the former sentence instructs you to refer to the online help of SoMachine which is accessible through the upper-right help button.

Please refer to the SoMachine online help for detailed information on these Functions and Function Blocks: Function description, Graphical representation, I/O Variables description, and more.

To install these libraries in your own project, please refer to Library Manager (see page 30).
1.5. Hardware Installation

Required Devices

<table>
<thead>
<tr>
<th>N°</th>
<th>Designation</th>
<th>Reference</th>
<th>Use or Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SoMachine Software</td>
<td>MSD CHNLMUA</td>
<td>SoMachine Software, 1-station license, installed on a PC</td>
</tr>
<tr>
<td></td>
<td>SoMachine Solution Extension Software (1)</td>
<td>MSD CHLKMUV30S0</td>
<td>SoMachine Solution Extension Software, 1-station license, installed on the same PC than SoMachine Software</td>
</tr>
<tr>
<td>2a</td>
<td>Terminal port/USB port cordset</td>
<td>TCS XCN AM UM3P</td>
<td>From the mini B USB port on the LMC058, M238, or M258 controller base to the type A USB port on the PC terminal for programming and updating firmware; length: 3 m (10 ft)</td>
</tr>
<tr>
<td>2b</td>
<td>Programming cable</td>
<td>BMX XCA USB H018</td>
<td>Same as TCS XCN AM UM3P, but with two ground connections along the cable; length: 1.8 m (6 ft)</td>
</tr>
<tr>
<td>3</td>
<td>Modicon LMC058 Motion controller (2)</td>
<td>LMC058 •••••••</td>
<td>Compact base motion controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programming port name: Pgr Port</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modbus port name: MBS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or Modicon M238 Logic controller (2)</td>
<td>TM238 ••••••••••</td>
<td>Compact base logic controller with 24 I/O (removable battery to be ordered separately: TSX PLP 01)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programming port name: Prg. Port</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modbus port name: SL1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or Modicon M258 Logic controller (2)</td>
<td>TM258 ••••••••••</td>
<td>Compact base logic controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programming port name: Pgr Port</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modbus port name: MBS</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Modbus RS485 cordset with 2 × RJ45 connectors</td>
<td>VW3 A8 306 R03, or VW3 A8 306 R10, or VW3 A8 306 R30</td>
<td>From the Serial port marked MBS on the LMC058 or M258 controller (or the SL1 port on the M238 controller) to the RJ45 connector of the ATV12 drive Length 0.3 m (1 ft), 1.0 m (3.3 ft), or 3.0 m (10 ft), depending on the reference of the cable</td>
</tr>
<tr>
<td>5</td>
<td>Altivar 12 drive</td>
<td>ATV12•••••</td>
<td>Altivar 12 variable speed drive for asynchronous motor, used here as a Modbus slave</td>
</tr>
</tbody>
</table>

(1) This software is only required if you are using a Motion Controller for Solutions (ref. LMC058•••••S0) or a Logic Controller for Solutions (ref. TM238•••••••••S0 or TM258•••••••••S0).

(2) This example guide describes how to create a SoMachine project for a Modicon LMC058 Motion controller, but it also describes how to adapt this project for a Modicon M238 Logic controller or a Modicon M258 Logic controller.
NOTE: No line terminator (ref. VW3 A8 306 RC) is required because of the short length of the Modbus RS485 cordset.

For the hardware setup of your controller, please refer to its Hardware Guide:

- Modicon LMC058 Motion Controller Hardware Guide
- Modicon M238 Logic Controller Hardware Guide
- Modicon M258 Logic Controller Hardware Guide

Please refer to the Altivar 12 Variable speed drives for asynchronous motors User manual for the hardware setup of the Altivar 12 drive.
2. Description of the Example Content

In SoMachine, the configuration of the example is made with the following devices:

- **1 Motion Controller**: LMC058LF42S0
  
  **NOTE**: This controller can be replaced with any LMC038 Motion Controller, M238 Logic Controller, or M258 Logic Controller.

- **1 Modbus Manager** on the controller to configure it as the Master of the Modbus RTU network

The program of the LMC058, M238, or M258 controller is made of the following items:

- **Library Manager**: List of the libraries linked to the programs of this example.
- **ST program**: Contains the source code for implementing what is described in *Functional Analysis of the Example* (see page 16). This is the default program since it is called by the MAST task of the controller.
- **LD program**: Translation of the ST program into LD language. To run this program on the controller, instead of the ST program, change the POU called by the MAST task of the controller from PLC_PRG_ST to PLC_PRG_LD.
- **CFC program**: Translation of the ST program into CFC language. To run this program on the controller, instead of the ST program, change the POU called by the MAST task of the controller from PLC_PRG_ST to PLC_PRG_CFC.
- **Task Configuration**: The standard MAST task, cyclically called every 20ms.
- **Serial Line**: Configured as a Modbus_Manager.

Aside from configuring the ATV12 drive, as described in *Configuration of the ATV12 Modbus Slave* (see page 57), there is no other software operation to perform on this device.

Example of the content visible in the Devices panel of the Program tab:
3. Creation of the Project

The steps listed in the following table describe how to create the SoMachine project, including the device(s) used in this example. No details are given here since it is assumed that you already know the Basic commands of SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the <strong>Create new machine</strong> part of the <strong>Home</strong> tab, select <strong>Start with empty project</strong> to create a new SoMachine project. Give this new project the following name: <strong>Modbus_RW_Var</strong>.</td>
</tr>
</tbody>
</table>
| 2    | In the **Configuration** tab:  
  ➢ Add an LMC058 Motion Controller.  
  In the following screen capture, a LMC058LF42S0 Motion Controller is selected and added to the SoMachine project: |
3. Creation of the Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Or, add an M238 or M258 Logic Controller if you plan to use this type of controller instead of an LMC058 Motion Controller.</td>
</tr>
</tbody>
</table>

In the following screen capture, a TM258LD42DT4L Logic Controller is selected and added to the SoMachine project:

![Screen Capture of SoMachine](image)

**NOTE:** Details on the selected controller are displayed in the **Information** section of SoMachine.

- **SoMachine controller version:** Defines the version of the selected controller; it is displayed in the **Information** section of SoMachine.

- **Target controller firmware version:** Defines the firmware version of your controller. This version is shown when you select your controller’s node, as shown as in *Downloading the Example to the Controller* (see page 65).

For compatibility purposes between a SoMachine controller version and a target controller firmware version, only the first three numbers of a version must be identical. In the preceding dialog displays, the **2.0.2.30** SoMachine controller version is compatible with any **2.0.2.*** target controller firmware version.

For each controller model, SoMachine only presents the latest available version. If you check the **Display all versions (for expert only)** option, SoMachine will list the supported controller firmware versions. However, a good practice consists in using the latest available version and updating the firmware of your controller, if required. Please refer to the document that corresponds to your controller:

- *Modicon LMC058 Motion Controller Programming Guide*
- *M238 ExecLoader User Guide*
- *Modicon M258 Controller Programming Guide*
### 3. Creation of the Project

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 3    | Rename this controller.  
In the following screen capture, the LMC058 Motion Controller is renamed to **LMC058_Controller**: |

<table>
<thead>
<tr>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: LMC058_Controller(LMC058LF4250)</td>
</tr>
<tr>
<td>Vendor: Schneider Electric</td>
</tr>
<tr>
<td>Version: 2.0.2.30</td>
</tr>
<tr>
<td>Order-#: LMC058LF4250</td>
</tr>
</tbody>
</table>

**Description**
LMC058 Performance Motion controller for Solutions - 2 x 5 fast sink input (200kHz, 24Vdc), 2 x 2 fast push-pull outputs (100kHz, 24Vdc, 0.2 A), 2 x 2 sink inputs (24Vdc), 12 sink inputs (24Vdc) and 12 source outputs (24Vdc, 0.5A), 1 Ethernet port, 1 serial line port, 1 CANopen master, 1 CANmotion master and 1 Encoder connector. Timer and calendar. Removable terminal blocks. |
| 4 | Save your new project. |
# 4. Serial Line Modbus Manager

The steps listed in the following table describe how to add and configure the Modbus Manager of the LMC058 controller.

If you use a M238 or a M258 controller instead of a LMC058 controller, these steps also tell you how to adapt them to your controller.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | **LMC058 or M258 controller:** Click on the SoMachine-Network_Manager Serial Line port:  
[Diagram of LMC058 or M258 controller with Serial Line: SoMachine-Network_Manager highlighted]  
This will remove this Manager and suggest replacing it by another Serial Line Manager.  
**M238 controller:** This step is not necessary since a default Modbus_Manager device is configured for its **Serial Line 1**. In the case of this controller, you must go to *Hardware Configuration (see page 27)*. |
| 2    | **LMC058 or M258 controller:** Confirm the removal of the SoMachine-Network_Manager by clicking the OK button:  
[Diagram of M238 or M258 controller with Serial Line 1: Modbus_Manager highlighted] |
### 4. Serial Line Modbus Manager

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 3    | **LMC058 or M258 controller:** The *Add device* window is automatically displayed to allow you to replace the Manager just removed:  
  - Select the **Schneider Electric** vendor.  
  - Select the **Modbus_Manager** device.  
  - **Click on the Add and close button.**  
  This will configure the Serial Line port of the LMC058 controller as a **Modbus_Manager**. |

| 4    | **Hardware Configuration**  
  To open the hardware configuration of your controller, double-click on its image: |

| 5    | **LMC058 or M258 controller:** In the left-hand panel, select:  
  - **Communication** menu  
  - **Serial Line**  
  - **Physical Settings**  
  **M238 controller:** In the left-hand panel, select:  
  - **Communication** menu  
  - **Serial Line 1**  
  - **Physical Settings** |
### 4. Serial Line Modbus Manager

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 6    | **In the **Configuration** tab of the central panel, set the configuration shown below:**  
  - Baud rate: 19200  
  - Parity: None  
  - Data bits: 8  
  - Stop bits: 1  
  - Physical Medium: RS 485  
  - Polarisation Resistor: No |

**LC058 or M258 controller:** In the left-hand panel, select:  
- **Communication** menu  
- **Serial Line**  
- **Protocol Settings**

**M238 controller:** In the left-hand panel, select:  
- **Communication** menu  
- **Serial Line 1**  
- **Protocol Settings**
In the **Configuration** tab of the central panel, change the **Addressing** from **Slave** to **Master**:

Click on the **Back** button to revert to the main **Configuration** tab.

A network object has automatically been connected to the Serial Line port of your controller to depict the Modbus network:

**Information**

- **Name**: Modbus_Manager(Modbus_Manager)
- **Vendor**: Schneider Electric
- **Order-#**: 
- **Description**: Modbus manager

**Parameters**

- **Address**: 0

Modify address configuration
### 5. Library Manager

The steps listed in the following table describe how to add and/or check the list of the libraries linked to this example.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Select the **Program** tab.  
In the **Devices** tree view, double-click on the **Library Manager** to open the list of the libraries linked to the **Application** software of this example. |
| 2    | Check that the **PLCCommunication**, **Standard**, and **Util** libraries are already linked, as shown below: |

**NOTE:** These libraries are grayed to inform that they have been automatically linked to the program upon addition of the controller to the project and that they cannot be removed.
6. ST, LD, or CFC Program

Each of the following three chapters describes how to create the program used in the example. Choose the language of your program (ST, LD, or CFC) and refer to the corresponding chapter:

- ST Program ........................................................................................... 32
- LD Program ........................................................................................... 41
- CFC Program ........................................................................................ 50

You only need to write your SoMachine program in one of these three languages.

In addition, each of these three chapters begins with detailed explanations of any optional steps.
## 6. ST, LD, or CFC Program

### 6.1. ST Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Creation of the POU:</strong> Create a new POU in ST language, called <strong>PLC_PRG_ST</strong>.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

| 2    | **ST variables:** In the upper part of the ST editor, declare the following variables: |

```plaintext
PROGRAM PLC_PRG_ST
VAR

(*********************)
(*** TON Variables ***)
(*********************)

// TON Function Block for delaying the start of this program
TON_START                   : TON;
// Delayed Rising Edge signal for starting the program
v_xStartProgram             : BOOL := FALSE;

(**********************)
(*** ADDM Variables ***)
(**********************)

// ADDM Function Block for formatting the address of the ATV12 Modbus Slave
ADDM_MODBUS_ATV12           : ADDM;
// ADDRESS structure for the address of the ATV12 Modbus Slave
v_addressModbusAtv12        : ADDRESS;
// "Done" result of the Address conversion
v_xAddressDone              : BOOL := FALSE;
// "Error" result of the Address conversion
v_xAddressError             : BOOL := FALSE;
// Result of the Address conversion: OK if "Done" without any "Error"
v_xAddressIsOK              : BOOL := FALSE;

(******************************)
(*** ETA Register Variables ***)
(******************************)
```

---

**EIO0000000913.01 04/2012**
### 6. ST, LD, or CFC Program

#### Step 1: Reading ETA Register

- **BLINK Function Block** for periodic reading of the ETA register
  - `BLINK ETA : BLINK;`

- **Command to read the ETA register**
  - `v_xReadETARregister : BOOL := FALSE;`

#### Step 2: Reading ETA Value

- **READ_VAR Function Block** for reading the ETA register of the ATV12 Modbus device
  - `READ_VAR_ETA : READ_VAR;`

- **Buffer** for the value of the ETA register
  - `v_wRegisterETA := 0;`

- **Done** result of the ETA register read operation
  - `v_xReadETADone : BOOL := FALSE;`

- **Busy** output of the ETA register read operation
  - `v_xReadETABusy : BOOL := FALSE;`

- **Error** result of the ETA register read operation
  - `v_xReadETAError : BOOL := FALSE;`

- **Communication error** with the ATV12 Modbus device (Timeout excluded)
  - `v_xCommErrorAtv12 : BOOL := FALSE;`

#### Step 3: Writing LFR Register

- **MANUAL command for starting one LFR register write operation**
  - `v_xCmdManualWriteLFR : BOOL := FALSE;`

- **Automatic command for reading the new value of the LFR register**
  - `v_xCmdAutoReadLFR : BOOL := FALSE;`

- **Command to write the LFR register**
  - `v_xWriteLFRRegister : BOOL := FALSE;`

- **Value** of the LFR register to write on the device (-400.0 Hz to +400.0 Hz; unit: 0.1 Hz)
  - `v_iWriteLFRValue := 0;`

- **SINGLE_WRITE Function Block** for writing the LFR register of the ATV12 Modbus device
  - `SINGLE_WRITE_LFR : SINGLE_WRITE;`

- **Value** of the LFR register to write on the device (after conversion to WORD)
  - `v_wWriteLFRValue := 0;`

- **Busy** output of the LFR register write operation
  - `v_xWriteLFRBusy := FALSE;`

- **Command to read the LFR register**
  - `v_xReadLFRRegister : BOOL := FALSE;`

- **READ_VAR Function Block** for reading the new value of the LFR register of the ATV12 Modbus device
  - `READ_VAR_LFR : READ_VAR;`

- **Buffer** for the value of the LFR register
  - `v_wRegisterLFR := 0;`

- **Busy** output of the LFR register read operation
  - `v_xReadLFRBusy := FALSE;`

- **Value** of the LFR register read on the device (-400.0 Hz to +400.0 Hz; unit: 0.1 Hz)
  - `v_iReadLFRValue := 0;`

#### Step 4: Writing ACC & DEC Registers

- **MANUAL command for starting one ACC & DEC registers write operation**
  - `v_xCmdManualWriteACC_DEC : BOOL := FALSE;`

- **Automatic command for reading the new values of the ACC & DEC registers**
  - `v_xCmdAutoReadACC_DEC : BOOL := FALSE;`
## 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Command to write the ACC &amp; DEC registers</td>
<td>v_xWriteACC_DECRegisters : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Value of the ACC &amp; DEC registers to write on the device (0.0 s to 999.9 s; // unit: 0.1 s)</td>
<td>v_iWriteACCValue : INT := 0; v_iWriteDECValue : INT := 0;</td>
</tr>
<tr>
<td>// WRITE_VAR Function Block for writing the ACC &amp; DEC registers of the ATV12 // Modbus device</td>
<td>WRITE_VAR_ACC_DEC : WRITE_VAR;</td>
</tr>
<tr>
<td>// Buffer for the values of the ACC &amp; DEC registers to write on the device (after // conversion to WORD)</td>
<td>v_wWriteACC_DECValues : ARRAY [0..1] OF WORD := [0,0];</td>
</tr>
<tr>
<td>// &quot;Busy&quot; output of the ACC &amp; DEC registers write operation</td>
<td>v_xWriteACC_DECBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Command to read the ACC &amp; DEC registers</td>
<td>v_xReadACC_DECRegisters : BOOL := FALSE;</td>
</tr>
<tr>
<td>// READ_VAR Function Block for reading the new value of the ACC &amp; DEC registers // of the ATV12 Modbus device</td>
<td>READ_VAR_ACC_DEC : READ_VAR;</td>
</tr>
<tr>
<td>// Buffer for the values of the ACC &amp; DEC registers</td>
<td>v_wRegisterACC_DEC : ARRAY[0..1] OF WORD := [0,0];</td>
</tr>
<tr>
<td>// &quot;Busy&quot; output of the ACC &amp; DEC registers read operation</td>
<td>v_xReadACC_DECBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Value of the ACC &amp; DEC registers read from the device (0.0 s to 999.9 s; unit: // 0.1 s)</td>
<td>v_iReadACCValue : INT := 0; v_iReadDECValue : INT := 0;</td>
</tr>
<tr>
<td>(**************************************)</td>
<td>(** SP2 to SP8 Registers Variables ***)</td>
</tr>
<tr>
<td>(**************************************)</td>
<td>// MANUAL command for starting one SP2 to SP8 registers write &amp; read operation</td>
</tr>
<tr>
<td>// Command to write &amp; read the SP2 to SP8 registers</td>
<td>v_xWrRdSP2_SP8Registers : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Values of the SP2 to SP8 registers to write on the device (0.0 Hz to 400.0 Hz; // unit: 0.1 Hz)</td>
<td>v_uiWriteSP2Value : UINT := 0; v_uiWriteSP3Value : UINT := 0; v_uiWriteSP4Value : UINT := 0; v_uiWriteSP5Value : UINT := 0; v_uiWriteSP6Value : UINT := 0; v_uiWriteSP7Value : UINT := 0; v_uiWriteSP8Value : UINT := 0;</td>
</tr>
<tr>
<td>// WRITE_READ_VAR Function Block for writing &amp; reading the SP2 to SP8 registers // of the ATV12 Modbus device</td>
<td>WRITE_READ_VAR_SP2_SP8 : WRITE_READ_VAR;</td>
</tr>
<tr>
<td>// Buffer for the values of the SP2 to SP8 registers to write on the device // (after conversion to WORD)</td>
<td>v_wWriteSP2_SP8Values : ARRAY [0..6] OF WORD := [7(0)];</td>
</tr>
<tr>
<td>// Buffer for the values of the SP2 to SP8 registers read from the device</td>
<td>v_wReadSP2_SP8Values : ARRAY [0..6] OF WORD := [7(0)];</td>
</tr>
<tr>
<td>// &quot;Busy&quot; output of the SP2 to SP8 registers write &amp; read operation</td>
<td>v_xWriteSP2_SP8Busy : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Values of the SP2 to SP8 registers read from the device (0.0 Hz to 400.0 Hz; // unit: 0.1 Hz)</td>
<td>v_uiReadSP2Value : UINT := 0; v_uiReadSP3Value : UINT := 0; v_uiReadSP4Value : UINT := 0; v_uiReadSP5Value : UINT := 0; v_uiReadSP6Value : UINT := 0; v_uiReadSP7Value : UINT := 0; v_uiReadSP8Value : UINT := 0;</td>
</tr>
</tbody>
</table>
### 6. ST, LD, or CFC Program

#### Step 3

**ST program:** In the lower part of the ST editor, implement the following program:

```plaintext
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><strong>ST program:</strong> In the lower part of the ST editor, implement the following program:</td>
</tr>
</tbody>
</table>

```

```
(* TON Function Block for delaying the start of this program *)
(TON_START.IN := TRUE; // IN - Enabled (TRUE)
TON_START.PT := T#1S; // IN - Duration of the TON timer (1 second)
TON_START(); // Function Block call
v_xStartProgram := TON_START.Q); // OUT - Resulting delay before running the rest of this program
(*TON_START.ET*) // OUT - This output is not used in this program

(** Formatting the Address of the ATV12 Modbus Slave **)
(* ADDM Function Block(s): ADDM ***)
(* Note: In ST language, this call syntax is required for Function Blocks that include *)
(* at least one "VAR_IN_OUT" variable ("AddrTable" in the case of ADDM) *)

ADDMODBUS_ATV12(
AddrTable := v_addressModbusAtv12, // IN/OUT - Resulting ADDRESS structure
Execute := v_xStartProgram , // IN - Rising Edge signal that triggers this Function Block
Addr := ‘1.2’ , // IN - Modbus Serial Address Format = \<communication port number>.<slave address>
Done => v_xAddressDone , // OUT - Resulting address is OK
Error => v_xAddressError , // OUT - Resulting address is not OK
(*CommError*)
); // OUT - This error code is not used in this program

(* Result of the Address conversion: OK if 'Done' without any 'Error' *)
v_xAddressIsOK := v_xAddressDone AND NOT v_xAddressError;

(** Periodic Communications: ETA Register read once per second **)
(* READ_VAR Function Block(s): READ_VAR ***)

READ_VAR_ETA(
Execute := v_xReadETARegister, // IN - Rising Edge signal that triggers this Function Block
Abort := FALSE, // IN - Function Block not aborted (FALSE)
Addr := v_addressModbusAtv12, // IN - Formatted address of the ATV12 Modbus device
Timeout := 5, // IN - Timeout of 500 ms
ObjType := ObjectType.MW, // IN - Type of object to be read: MW
FirstObj := 3201, // IN - First object to be read: ETA register (address = 3201)
Quantity := 1, // IN - Number of objects to read: 1 register
Buffer := ADR(v_wRegisterETA), // IN - Address of the variable for RECEIVING the value of the ETA register
Done => v_xReadETADone, // OUT - "Done" result of the ETA register

(* Command to read the ETA register: IF Address is OK AND 1-second clock *)
v_xReadETADone := v_xAddressIsOK AND v_xClockReadETARegister;

(* Note: In ST language, this call syntax is required for Function Blocks of the "PLCCommunication" library that use an 'Addr' INPUT variable (data type = ADDRESS) *)
```

EIO000000913.01 04/2012
### 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy =&gt; v_xReadETABusy,</td>
<td>// read operation</td>
</tr>
<tr>
<td>(<em>Aborted</em>)</td>
<td>// OUT - &quot;Busy&quot; output of the ETA register read operation</td>
</tr>
<tr>
<td>Error =&gt; v_xReadETAError,</td>
<td>// read operation</td>
</tr>
<tr>
<td>(<em>OperError</em>)</td>
<td>// OUT - &quot;Error&quot; result of the ETA register read operation</td>
</tr>
<tr>
<td>CommError =&gt; v_bReadETACommError);</td>
<td>// register read operation</td>
</tr>
<tr>
<td>(<em>OperError</em>)</td>
<td>// OUT - This error code is not used in this program</td>
</tr>
</tbody>
</table>

/* The ATV12 Modbus device is present (TRUE) if it has correctly answered (once the BUSY is FALSE) */

IF NOT v_xReadETABusy AND NOT v_xReadETAError AND v_xReadETADone THEN

v_xPresenceAtv12 := TRUE; // The ATV12 is present
v_xCommErrorAtv12 := FALSE; // No communication error

ELSEIF NOT v_xReadETABusy AND \READ_VAR_ETA.Error THEN

IF (v_bReadETACommError = CommunicationErrorCodes.TimedOut) THEN

v_xPresenceAtv12 := FALSE;

ELSE

v_xCommErrorAtv12 := TRUE;

END_IF

END_IF

/******************************************************************
 *** On-demand command : LFR Register Write + LFR Register Read ***
 *** Function Block(s): SINGLE_WRITE + READ_VAR                 ***
******************************************************************

(* Conversion from INT to WORD of the LFR register value to write on the device *)
(* NOTE: The conversion must preserve the +/- sign in the case of the LFR register. *)

v_wWriteLFRValue := INT_TO_WORD(v_iWriteLFRValue);

(* Command to write the LFR register: IF Address is OK AND Manual Command to Write LFR *)

v_xWriteLFRRegister := v_xAddressIsOK AND v_xCmdManualWriteLFR;

(* SINGLE_WRITE Function Block for writing the LFR register of the ATV12 Modbus device *)

(* Note: In ST language, this call syntax is required for Function Blocks of the *)

(* 'PLCCommunication' library that use an 'Addr' INPUT variable (data type = ADDRESS) *)

SINGLE_WRITE_LFR{

Execute := v_xWriteLFRRegister, // IN - Rising Edge signal that triggers this Function Block
Abort := FALSE, // IN - Function Block not aborted (FALSE)
Addr := v_addressModbusAtv12, // IN - Formatted address of the ATV12 Modbus device
Timeout := 5, // IN - Timeout of 500 ms
ObjType := ObjectType.MW, // IN - Type of object to be written: MW --> The Modbus function #6 (write single register) is used
FirstObj := 8502, // IN - Object to be written: LFR register (address = 8502)
theWord := v_wWriteLFRValue, // IN - Value to write in the LFR register of the ATV12 Modbus device
(*Done*) // OUT - This output is not used in this program
Busy => v_xWriteLFRBusy); // OUT - "Busy" output of the ETA register read operation

(*Upon completion of the LFR WRITE operation, an automatic LFR READ operation *)
(* is performed to check if the value of the LFR register has been updated *)
IF v_xWriteLFRRegister AND NOT v_xWriteLFRBusy THEN  
  v_xCmdAutoReadLFR := TRUE; 
END_IF  

(* Command to read the LFR register: IF Address is OK AND Automatic Command to Read LFR *)  
v_xReadLFRRegister := v_xAddressIsOK AND v_xCmdAutoReadLFR;  

(* READ_VAR Function Block for reading the new value of the LFR register of the ATv12 Modbus device *)  
(* Note: In ST language, this call syntax is required for Function Blocks of the ‘PLCCommunication’ library that use an ‘Addr’ INPUT variable (data type = ADDRESS) *)  
READ_VAR_LFR(  
  Execute := v_xReadLFRRegister, // IN - Rising Edge signal that triggers this Function Block  
  Abort := FALSE, // IN - Function Block not aborted (FALSE)  
  Addr := v_addressModbusAtv12, // IN - Formatted address of the ATv12 Modbus device  
  Timeout := 5, // IN - Timeout of 500 ms  
  ObjType := ObjectType.MW, // IN - Type of object to be read: MW --> The Modbus function #3 (read holding registers) is used  
  FirstObj := 8502, // IN - First object to be read: LFR register (address = 8502)  
  Quantity := 1, // IN - Number of objects to read: 1 register  
  Buffer := ADR(v_wRegisterLFR), // IN - Address of the variable for RECEIVING the value of the LFR register  
  (*Done*) // OUT - This output is not used in this program  
  Busy => v_xReadLFRBusy); // OUT - "Busy" output of the LFR register read operation  

(* Upon completion of the LFR READ operation, both LFR WRITE and LFR READ operations are ended AND the read value is converted from WORD to INT. *)  
(* are ended AND the read value is converted from WORD to INT. *)  
IF v_xReadLFRRegister AND NOT v_xReadLFRBusy THEN  
  v_xCmdManualWriteLFR := FALSE;  
  v_xCmdAutoReadLFR := FALSE;  
(* Conversion from WORD to INT of the LFR register value read on the device *)  
(* NOTE: The conversion must preserve the +/- sign in the case of the LFR register. *)  
  v_iReadLFRValue := WORD_TO_INT(v_wRegisterLFR);  
END_IF  

/* On-demand command : ACC & DEC Registers Write + ACC & DEC Registers Read */  
/* Function Block(s): WRITE_VAR + READ_VAR */  
*/

(* Conversion from INT to WORD of the ACC & DEC registers values to write on the device *)  
v_wWriteACC_DECValues[0] := INT_TO_WORD(v_iWriteACCValue); // 1st WORD = ACC Register  
v_wWriteACC_DECValues[1] := INT_TO_WORD(v_iWriteDECValue); // 2nd WORD = DEC Register  

(* Command to write the ACC & DEC registers: IF Address is OK AND Manual Command to Write ACC & DEC *)  
v_xWriteACC_DECRegisters := v_xAddressIsOK AND v_xCmdManualWriteACC_DEC;  

(* WRITE_VAR Function Block for writing the ACC & DEC registers of the ATv12 Modbus device *)  
(* Note: In ST language, this call syntax is required for Function Blocks of the ‘PLCCommunication’ library that use an ‘Addr’ INPUT variable (data type = ADDRESS) *)
ADDRESS *)
WRITE_VAR_ACC_DEC(
  Execute := v_xWriteACC_DECRegisters, // IN - Rising Edge signal that
          // triggers this Function Block
  Abort := FALSE, // IN - Function Block not aborted
          // (FALSE)
  Addr := v_addressModbusAtv12, // IN - Formatted address of the ATV12
         // Modbus device
  Timeout := 5, // IN - Timeout of 500 ms
  ObjType := ObjectType.MW, // IN - Type of object to be written:
          // MW --> The Modbus function #16
          // (write multiple
          // registers) is used
  FirstObj := 9001, // IN - Object to be written: ACC &
               // DEC registers (addresses = 9001 & 9002)
  Quantity := 2, // IN - Number of objects to write:
                 // 2 registers
  Buffer := ADR(v_wWriteACC_DECValues), // IN - Address of the variables which
          // values will be SENT to the ACC &
          // DEC Registers
          // (*Done*)                                  // OUT - This output is not used in
          // this program
          // Busy => v_xWriteACC DECBusy);              // OUT - "Busy" output of the ACC & DEC
          // registers read operation
  (*Aborted*)                            // OUT - This output is not used in this program
  (*Error*)                              // OUT - This output is not used in this program
  (*CommError*)                           // OUT - This output is not used in this program
  (*OperError*)                           // OUT - This output is not used in this program
)

(* Upon completion of the ACC & DEC WRITE operation, an automatic ACC & DEC READ
operation *)
(* is performed to check if the values of the ACC & DEC registers have been updated *)
IF v_xWriteACC_DECRegisters AND NOT v_xWriteACC_DECBusy THEN
  v_xCmdAutoReadACC_DEC := TRUE;
END_IF

(* Command to read the ACC & DEC registers: IF Address is OK AND Automatic Command to
Read ACC & DEC *)
v_xReadACC_DECRegisters := v_xAddressIsOK AND v_xCmdAutoReadACC_DEC;

(* READ_VAR Function Block for reading the new values of the ACC & DEC registers of
the ATV12 Modbus device *)
(* Note: In ST language, this call syntax is required for Function Blocks of the
*)
(* 'PLCCommunication' library that use an 'Addr' INPUT variable (data type =
ADDRESS) *)
READ_VAR_ACC_DEC(
  Execute := v_xReadACC_DECRegisters, // IN - Rising Edge signal that
          // triggers this Function Block
  Abort := FALSE, // IN - Function Block not aborted
          // (FALSE)
  Addr := v_addressModbusAtv12, // IN - Formatted address of the ATV12
         // Modbus device
  Timeout := 5, // IN - Timeout of 500 ms
  ObjType := ObjectType.MW, // IN - Type of object to be read: MW
          // --> The Modbus function #3 (read
          // holding registers) is used
  FirstObj := 9001, // IN - Objects to be read: ACC & DEC
                  // registers (addresses = 9001 & 9002)
  Quantity := 2, // IN - Number of objects to read:
                 // 2 registers
  Buffer := ADR(v_wRegisterACC_DEC), // IN - Address of the variables for
          // RECEIVING the values of the ACC &
          // DEC registers
          // (*Done*)                                  // OUT - This output is not used in this
          // program
          // Busy => v_xReadACC DECBusy);              // OUT - "Busy" output of the ACC & DEC
          // registers read operation
  (*Aborted*)                            // OUT - This output is not used in this program
  (*Error*)                              // OUT - This output is not used in this program
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>// program (<em>CommError</em>) // OUT - This output is not used in this program (<em>OperError</em>) // OUT - This output is not used in this program</td>
</tr>
</tbody>
</table>
|      | /* Upon completion of the ACC & DEC READ operations, both ACC & DEC WRITE and ACC & DEC READ *)
|      | /* operations are ended AND the read values are converted from WORD to INT. */
|      | IF v_xReadACC_DECRegisters AND NOT v_xReadACC_DECBusy THEN
|      | v_xCmdManualWriteACC_DEC := FALSE;
|      | v_xCmdAutoReadACC_DEC := FALSE;
|      | /* Conversion from WORD to INT of the ACC & DEC registers values read on the device */
|      | v_iReaACCValue := WORD_TO_INT(v_wRegisterACC_DEC[0]);
|      | v_iReaDECValue := WORD_TO_INT(v_wRegisterACC_DEC[1]);
|      | END_IF

(*****************************************************************************)
| *** On-demand command : SP2 to SP8 Registers Write & Read ***
| *** Function Block(s): WRITE_READ_VAR ***
|*******************************************************************************)

// Conversion from UINT to WORD of the SP2 to SP8 registers values to write on the device
v_wWriteSP2_SP8Values[0] := UINT_TO_WORD(v_uiWriteSP2Value); // 1st WORD=SP2 Register
v_wWriteSP2_SP8Values[1] := UINT_TO_WORD(v_uiWriteSP3Value); // 2nd WORD=SP3 Register
v_wWriteSP2_SP8Values[2] := UINT_TO_WORD(v_uiWriteSP4Value); // 3rd WORD=SP4 Register
v_wWriteSP2_SP8Values[3] := UINT_TO_WORD(v_uiWriteSP5Value); // 4th WORD=SP5 Register
v_wWriteSP2_SP8Values[4] := UINT_TO_WORD(v_uiWriteSP6Value); // 5th WORD=SP6 Register
v_wWriteSP2_SP8Values[5] := UINT_TO_WORD(v_uiWriteSP7Value); // 6th WORD=SP7 Register
v_wWriteSP2_SP8Values[6] := UINT_TO_WORD(v_uiWriteSP8Value); // 7th WORD=SP8 Register

(* Command to write the SP2 to SP8 registers: IF Address is OK AND Manual Command to Write & Read SP2 to SP8 *)
IF v_xAddressIsOK AND v_xCmdManualWrRdSP2_SP8 THEN
  v_xWrRdSP2_SP8Registers := v_xAddressIsOK AND v_xCmdManualWrRdSP2_SP8;
END_IF

(* WRITE_READ_VAR Function Block for writing & reading the SP2 to SP8 registers of the ATV12 Modbus device *)
(* Note: In ST language, this call syntax is required for Function Blocks of the *)
(* 'PLCCommunication' library that use an 'Addr' INPUT variable (data type = ADDRESS) *)
WRITE_READ_VAR_SP2_SP8(
  Execute := v_xWrRdSP2_SP8Registers, // IN - Rising Edge signal that triggers this Function Block
  Abort := FALSE, // IN - Function Block not aborted (FALSE)
  Addr := v_addressModbusAtv12, // IN - Formatted address of the ATV12 Modbus device
  Timeout := 5, // IN - Timeout of 500 ms
  ObjType := ObjectType.MW, // IN - Type of object to be written: MW --> The Modbus function #23 (read/write multiple registers) is used
  FirstWriteObj := 11410, // IN - Object to be written: SP2 to SP8 registers (addresses = 11410 to 11416)
  WriteQuantity := 7, // IN - Number of objects to write: 7 registers
  WriteBuffer := ADR(v_wWriteSP2_SP8Values), // IN - Address of the variables which values will be SENT to the SP2 to SP8 Registers
  FirstReadObj := 11410, // IN - Object to be read: SP2 to SP8 registers (addresses = 11410 to 11416)
  ReadQuantity := 7, // IN - Number of objects to read: 7 registers
  ReadBuffer := ADR(v_wReadSP2_SP8Values), // IN - Address of the variables for RECEIVING the values of the SP2 to SP8 registers
  (*Done*) // OUT - This output is not used in this program
  Busy := v_xWriteSP2_SP8Busy); // OUT - "Busy" output of the ACC & DEC registers read
### Step 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(<em>Aborted</em>) // OUT - This output is not used in this program</td>
</tr>
<tr>
<td></td>
<td>(<em>Error</em>) // OUT - This output is not used in this program</td>
</tr>
<tr>
<td></td>
<td>(<em>CommError</em>) // OUT - This output is not used in this program</td>
</tr>
<tr>
<td></td>
<td>(<em>OperError</em>) // OUT - This output is not used in this program</td>
</tr>
</tbody>
</table>

(* Upon completion of the SP2 to SP8 WRITE & READ operation, it is ended *)

IF v_xWrRdSP2_SP8Registers AND NOT v_xWriteSP2_SP8Busy THEN
  v_xCmdManualWrRdSP2_SP8 := FALSE;
 (* Conversion from WORD to UINT of the SP2 to SP8 registers values read on the device *)
  v_uiReadSP2Value := WORD_TO_UINT(v_wReadSP2_SP8Values[0]);
  v_uiReadSP3Value := WORD_TO_UINT(v_wReadSP2_SP8Values[1]);
  v_uiReadSP4Value := WORD_TO_UINT(v_wReadSP2_SP8Values[2]);
  v_uiReadSP5Value := WORD_TO_UINT(v_wReadSP2_SP8Values[3]);
  v_uiReadSP6Value := WORD_TO_UINT(v_wReadSP2_SP8Values[4]);
  v_uiReadSP7Value := WORD_TO_UINT(v_wReadSP2_SP8Values[5]);
  v_uiReadSP8Value := WORD_TO_UINT(v_wReadSP2_SP8Values[6]);
END_IF
6.2. LD Program

The following table presents one optional step that gives you information on how to program in LD language.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optional step</td>
</tr>
</tbody>
</table>

**How to display titles and comments in LD language**

- Select the **Options…** command of the **Tools** menu.
- Select, in the **Options** window, the **FBD, LD and IL editor** section.
- If you wish to add a title and/or a comment for each LD network, check the **Show network title** and/or the **Show network comment** options, as shown below:

![Options window](image)
6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Creation of the POU:</strong> Create a new POU in LD language, called <strong>PLC_PRG_LD</strong>.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><strong>LD variables:</strong> In the upper part of the LD editor, declare the following variables:</td>
</tr>
</tbody>
</table>

```plaintext
PROGRAM PLC_PRG_LD
VAR

(******************************)
(* *** TON Variables *** *)
(******************************)

// TON Function Block for delaying the start of this program
TON START                   : TON;
// Delayed Rising Edge signal for starting the program
v_xStartProgram             : BOOL := FALSE;

(******************************)
(* *** ADDM Variables *** *)
(******************************)

// ADDM Function Block for formatting the address of the ATV12 Modbus Slave
ADDM MODBUS ATV12           : ADDM;
// ADDRESS structure for the address of the ATV12 Modbus Slave
v_addressModbusAtv12        : ADDRESS;
// "Done" result of the Address conversion
v_xAddressDone              : BOOL := FALSE;
// "Error" result of the Address conversion
v_xAddressError             : BOOL := FALSE;
// Result of the Address conversion: OK if "Done" without any "Error"
v_xAddressIsOK              : BOOL := FALSE;

(******************************)
(* *** ETA Register Variables *** *)
(******************************)

// BLINK Function Block for periodic reading of the ETA register
BLINK ETA                   : BLINK;
// "OUT" output of the BLINK Function Block: Clock
v_xClockReadETARegister     : BOOL := FALSE;
// Command to read the ETA register
v_xReadETARegister          : BOOL := FALSE;
```
### 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>// READ VAR Function Block for reading the ETA register of the ATV12 Modbus device</td>
</tr>
<tr>
<td></td>
<td>READ_VAR_ETA : READ_VAR;</td>
</tr>
<tr>
<td></td>
<td>// Buffer for the value of the ETA register</td>
</tr>
<tr>
<td></td>
<td>v_wRegisterETA : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Done&quot; result of the ETA register read operation</td>
</tr>
<tr>
<td></td>
<td>v_xReadETADone : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the ETA register read operation</td>
</tr>
<tr>
<td></td>
<td>v_xReadETABusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Error&quot; result of the ETA register read operation</td>
</tr>
<tr>
<td></td>
<td>v_xReadETAErro : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// &quot;CommError&quot; result of the ETA register read operation</td>
</tr>
<tr>
<td></td>
<td>v_bReadETCommError : BYTE := CommunicationErrorCodes.CommunicationOK;</td>
</tr>
<tr>
<td></td>
<td>// Result of the ETA register read operation: SUCCESS</td>
</tr>
<tr>
<td></td>
<td>v_xReadETASuccess : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Result of the ETA register read operation: FAILURE</td>
</tr>
<tr>
<td></td>
<td>v_xReadETAFailure : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Presence (TRUE) or absence (FALSE) of the ATV12 Modbus device</td>
</tr>
<tr>
<td></td>
<td>v_xPresenceAtv12 : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Communication error with the ATV12 Modbus device (Timeout excluded)</td>
</tr>
<tr>
<td></td>
<td>v_xCommErrorAtv12 : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>(******************************)</td>
</tr>
<tr>
<td></td>
<td>(** LFR Register Variables ***)</td>
</tr>
<tr>
<td></td>
<td>(******************************)</td>
</tr>
<tr>
<td></td>
<td>// MANUAL command for starting one LFR register write operation</td>
</tr>
<tr>
<td></td>
<td>v_xCmdManualWriteLFR : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Automatic command for reading the new value of the LFR register</td>
</tr>
<tr>
<td></td>
<td>v_xCmdAutoReadLFR : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Command to write the LFR register</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLFRRegister : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Value of the LFR register to write on the device (-400.0 Hz to +400.0 Hz; unit: 0.1 Hz)</td>
</tr>
<tr>
<td></td>
<td>v_iWriteLFRValue : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>// SINGLE_WRITE Function Block for writing the LFR register of the ATV12 Modbus device</td>
</tr>
<tr>
<td></td>
<td>// ** Value of the LFR register to write on the device (after conversion to WORD)</td>
</tr>
<tr>
<td></td>
<td>v_wWriteLFRValue : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the LFR register write operation</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLFRBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Command to read the LFR register</td>
</tr>
<tr>
<td></td>
<td>v_xReadLFRRegister : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// READ_VAR Function Block for reading the new value of the LFR register of the ATV12 Modbus device</td>
</tr>
<tr>
<td></td>
<td>READ_VAR_LFR : READ_VAR;</td>
</tr>
<tr>
<td></td>
<td>// Buffer for the value of the LFR register</td>
</tr>
<tr>
<td></td>
<td>v_wRegisterLFR : WORD := 0;</td>
</tr>
<tr>
<td></td>
<td>// &quot;Busy&quot; output of the LFR register read operation</td>
</tr>
<tr>
<td></td>
<td>v_xReadLFRBusy : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Value of the LFR register read on the device (-400.0 Hz to +400.0 Hz; unit: 0.1 Hz)</td>
</tr>
<tr>
<td></td>
<td>v_iReadLFRValue : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>(******************************)</td>
</tr>
<tr>
<td></td>
<td>(** ACC &amp; DEC Registers Variables ***)</td>
</tr>
<tr>
<td></td>
<td>(******************************)</td>
</tr>
<tr>
<td></td>
<td>// MANUAL command for starting one ACC &amp; DEC registers write operation</td>
</tr>
<tr>
<td></td>
<td>v_xCmdManualWriteACC_DEC : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Automatic command for reading the new values of the ACC &amp; DEC registers</td>
</tr>
<tr>
<td></td>
<td>v_xCmdAutoReadACC_DEC : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Command to write the ACC &amp; DEC registers</td>
</tr>
<tr>
<td></td>
<td>v_xWriteACC_DECRegisters : BOOL := FALSE;</td>
</tr>
<tr>
<td></td>
<td>// Value of the ACC &amp; DEC registers to write on the device (0.0 s to 999.9 s; unit: 0.1 s)</td>
</tr>
<tr>
<td></td>
<td>v_iWriteACCValue : INT := 0;</td>
</tr>
<tr>
<td></td>
<td>v_iWriteDECValue : INT := 0;</td>
</tr>
</tbody>
</table>
### Step 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>EIO0000000913.01 04/2012</td>
</tr>
</tbody>
</table>

```plaintext
// WRITE_VAR Function Block for writing the ACC & DEC registers of the ATV12 Modbus device
WRITE_VAR_ACC_DEC : WRITE_VAR;
// Buffer for the values of the ACC & DEC registers to write on the device (after conversion to WORD)
v_wWriteACC_DECValues : ARRAY [0..1] OF WORD := [0,0];
// "Busy" output of the ACC & DEC registers write operation
v_xWriteACC_DECBusy : BOOL := FALSE;

// Command to read the ACC & DEC registers
v_xReadACC_DECRegisters : BOOL := FALSE;

// READ_VAR Function Block for reading the new value of the ACC & DEC registers of the ATV12 Modbus device
READ_VAR_ACC_DEC : READ_VAR;
// Buffer for the values of the ACC & DEC registers
v_uReadRegisterACC_DEC : ARRAY[0..1] OF WORD := [0,0];
// "Busy" output of the ACC & DEC registers read operation
v_xReadACC_DECBusy : BOOL := FALSE;

// Value of the ACC & DEC registers read from the device (0.0 s to 999.9 s; unit: 0.1 s)
v_iReadACCValue : INT := 0;
v_iReadDECValue : INT := 0;

(**************************************)
*** SP2 to SP8 Registers Variables ***
(**************************************)

// MANUAL command for starting one SP2 to SP8 registers write & read operation
v_xCmdManualWrRdSP2_SP8 : BOOL := FALSE;

// Command to write & read the SP2 to SP8 registers
v_xWrRdSP2_SP8Registers : BOOL := FALSE;

// Values of the SP2 to SP8 registers to write on the device (0.0 Hz to 400.0 Hz; unit: 0.1 Hz)
v_uiWriteSP2Value : UINT := 0;
v_uiWriteSP3Value : UINT := 0;
v_uiWriteSP4Value : UINT := 0;
v_uiWriteSP5Value : UINT := 0;
v_uiWriteSP6Value : UINT := 0;
v_uiWriteSP7Value : UINT := 0;
v_uiWriteSP8Value : UINT := 0;

// WRITE_READ_VAR Function Block for writing & reading the SP2 to SP8 registers of the ATV12 Modbus device
WRITE_READ_VAR_SP2_SP8 : WRITE_READ_VAR;
// Buffer for the values of the SP2 to SP8 registers to write on the device (after conversion to WORD)
v_wWriteSP2_SP8Values : ARRAY [0..6] OF WORD := [7(0)];
// Buffer for the values of the SP2 to SP8 registers read from the device
v_wReadSP2_SP8Values : ARRAY [0..6] OF WORD := [7(0)];
// "Busy" output of the SP2 to SP8 registers write & read operation
v_xWriteSP2_SP8Busy : BOOL := FALSE;

// Values of the SP2 to SP8 registers read from the device (0.0 Hz to 400.0 Hz; unit: 0.1 Hz)
v_uiReadSP2Value : UINT := 0;
v_uiReadSP3Value : UINT := 0;
v_uiReadSP4Value : UINT := 0;
v_uiReadSP5Value : UINT := 0;
v_uiReadSP6Value : UINT := 0;
v_uiReadSP7Value : UINT := 0;
v_uiReadSP8Value : UINT := 0;
```

END_VAR
**Step 4**  
**Action:** In the lower part of the LD editor, implement the following program:

```plaintext
*** TDM Function Block for delaying the start of this program ***
JT = Duration of the TDM timer (1 second)
0 = Resulting delay before running the rest of this program
TDM START

*** Formating the address of the AXUV6 Modbus Slave ***
--- Function Block: ADDMODS
ADDMODS Function block for formating the address of the AXUV6 Modbus Slave
AddTable = Resulting ADDRESS structure
Add = Address of the external device (Modbus Serial Address Format = "<communication port number>-<slave address>")

ADD MODBUS ATU6

ADDMODS ATU6

Result of the Address conversion: OK if 'done' without any 'Error'

*** Periodic Communications: E1A Register read once per second ***
--- Function Block: READ_VAR
READ_VAR Function block for periodic reading of the E1A register: 1-second clock
TIMELOW = Duration of the LOW state (500 ms)
TIMESHIGH = Duration of the HIGH state (500 ms)

READ_VAR

Command to read the E1A register: IF Address is OK AND 1-second clock

READ_VAR

--- Function Block: READ VAR  
READ VAR Function block for reading the E1A register of the AXUV6 Modbus device
Add = Formatted address of the AXUV6 Modbus device
TIME = Timeout of 500 ms
ObjectType = Type of object to be read: MW --> The Modbus function #4 (read holding registers) is used
FirstObj + Quantity = Object to be read: E1A register (address = 3001)
Buffer = Address of the variable for RECEIVING the value of the E1A register

READ VAR

--- Error Handling for TDM Delay functionality
--- Error Handling for ADDMODS function block
--- Error Handling for READ_VAR function block
--- Error Handling for READ VAR function block
--- Error Handling for READ VAR function block
--- Error Handling for READ VAR function block
```
### 6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The ATV2 Modbus device is present (BUSY) if it has correctly answered (once the BUSY is FALSE) -- The ATV2 is present -- No communication error</td>
</tr>
<tr>
<td></td>
<td>v_xReadSTSuccess v_xPresenceAvl2</td>
</tr>
<tr>
<td></td>
<td>v_xCommandErrorAvl2</td>
</tr>
<tr>
<td>10</td>
<td>Otherwise, a communication error is reported: Timeout or other error (once the BUSY is FALSE) -- The ATV2 is absent of Communication error (Timeout excluded)</td>
</tr>
<tr>
<td></td>
<td>v_xReadSTSuccess v_xReadSTFailure</td>
</tr>
<tr>
<td></td>
<td>v_xPresenceAvl2</td>
</tr>
<tr>
<td></td>
<td>v_xCommandErrorAvl2</td>
</tr>
<tr>
<td>11</td>
<td>*** On-sensor command - LPF Register Write &amp; LPF Register Read ***</td>
</tr>
<tr>
<td></td>
<td>FUNCTION block STATUS - 1 MSB 00</td>
</tr>
<tr>
<td></td>
<td>Conversion from INT to WORD of the LPF register value to write on the device MODBUS: the conversion must preserve the +/- sign in the case of the LPF register.</td>
</tr>
<tr>
<td></td>
<td>v_iWriteLPValue v_iWriteLError</td>
</tr>
<tr>
<td></td>
<td>v_iWriteLPValue</td>
</tr>
<tr>
<td>12</td>
<td>Command to write the LPF register: IF Address is OK AND Manual Command to Write LPF</td>
</tr>
<tr>
<td></td>
<td>v_vAddressToXx v_iWriteManualWriteLPF v_iWriteLPRegister</td>
</tr>
<tr>
<td>13</td>
<td>SINGLE_WRITE Function Block for writing the LPF register of the ATV2 Modbus device</td>
</tr>
<tr>
<td></td>
<td>Execute = Rising Edge signal that triggers this Function Block</td>
</tr>
<tr>
<td></td>
<td>Addr = Numerized address of the ATV2 Modbus device</td>
</tr>
<tr>
<td></td>
<td>Timeout = Timeout of 140 ms</td>
</tr>
<tr>
<td></td>
<td>ObjectType = Type of object to be read: MW -- The Modbus function 3 (write simple register) is used</td>
</tr>
<tr>
<td></td>
<td>theWord = Value to write in the LPF register of the ATV2 Modbus device</td>
</tr>
<tr>
<td></td>
<td>v_sWriteLPRegister SINGLE_WRITE</td>
</tr>
<tr>
<td></td>
<td>Execute = 1</td>
</tr>
<tr>
<td></td>
<td>Addr = v_AddressToXx</td>
</tr>
<tr>
<td></td>
<td>ObjectType = MW</td>
</tr>
<tr>
<td></td>
<td>theWord = v_iWriteLPValue</td>
</tr>
<tr>
<td></td>
<td>v_iWriteLError</td>
</tr>
<tr>
<td></td>
<td>v_sWriteLPBusy</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPBusy</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPAbort</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPError</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPTimeout</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPOpenError</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPSuccess</td>
</tr>
<tr>
<td>14</td>
<td>Upon completion of the LPF WRITE operation, an automatic LPF READ operation is performed to check if the value of the LPF register has been updated</td>
</tr>
<tr>
<td></td>
<td>v_sWriteLPReaders v_xWriteLPBusy v_xWriteLPRegister</td>
</tr>
<tr>
<td></td>
<td>v_xWriteLPReaders</td>
</tr>
<tr>
<td>15</td>
<td>Command to read the LPF register: IF Address is OK AND Automatic Command to Read LPF</td>
</tr>
<tr>
<td></td>
<td>v_vAddressToXx v_iCheckAutoReadLPF v_iWriteLPRegister v_iWriteLPValue</td>
</tr>
<tr>
<td></td>
<td>v_iWriteLPValue</td>
</tr>
</tbody>
</table>

---

**EIO0000000913.01 04/2012**
6. ST, LD, or CFC Program

Upon completion of the LSR READ operation, both LSR WRITE and LSR READ operations are ended and the read value of the LSR register is converted from WORD to INT.

Upon completion of the ACC & DEC WRITE operation, an automatic ACC & DEC READ operation is performed to check if the values of the ACC & DEC registers have been updated.

Command to read the ACC & DEC registers: IF Address is OK AND Automatic Command to Read ACC & DEC
6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>READ_VAR Function Block for reading the new values of the ACC &amp; DEC registers of the RTY12 Modbus device&lt;br&gt;ExecTime = Rising Edge signal that triggers this Function Block&lt;br&gt;Add = Formatted address of the RTY12 Modbus device&lt;br&gt;Timeout = Timeout of 5 50 ms&lt;br&gt;ObjType = Type of object to be read; M1 -&gt; The Modbus function # 3 (read holding registers) is used&lt;br&gt;FileSize = Quantity of objects to be read ACC &amp; DEC registers (addresses = 9601 &amp; 9902)&lt;br&gt;Buffer = Address of the variables for RECEIVING the values of the ACC &amp; DEC registers</td>
</tr>
<tr>
<td>24</td>
<td>Upon completion of the ACC &amp; DEC READ operations, both ACC &amp; DEC WRITE and ACC &amp; DEC READ operations are ended AND the read values of the ACC &amp; DEC registers are converted from WORD to INT</td>
</tr>
<tr>
<td>25</td>
<td>*** On-depend command - SP3 to SP6 Registers Write &amp; Read ***</td>
</tr>
<tr>
<td></td>
<td>Conversion from WORD to WORD of the SP2 to SP6 registers values to write on the device 1st WORD to 7th WORD = SP6 registers to SP6 Register</td>
</tr>
<tr>
<td></td>
<td>v_uWriteSPValue[0] —— v_uWriteSPValue[0] —— v_uWriteSPValue[0] —— v_uWriteSPValue[0] —— v_uWriteSPValue[0] —— v_uWriteSPValue[0] —— v_uWriteSPValue[0]</td>
</tr>
<tr>
<td></td>
<td>v_uWriteSPValue —— v_uWriteSPValue —— v_uWriteSPValue —— v_uWriteSPValue —— v_uWriteSPValue —— v_uWriteSPValue —— v_uWriteSPValue</td>
</tr>
<tr>
<td></td>
<td>Command to write the SP2 to SP6 registers: IF Address is OK AND Manual Command to Write &amp; Read SP2 to SP6</td>
</tr>
<tr>
<td></td>
<td>v_uAddress10X  v_uAddress12X  v_uAddress14X  v_uAddress16X</td>
</tr>
<tr>
<td>48</td>
<td>EIO0000000913.01 04/2012</td>
</tr>
</tbody>
</table>
### 6. ST, LD, or CFC Program

#### Step | Action
---|---
29 | **WRITE_READ_VAR SPF 380**

**Function Block for writing a reading the SPF to SPW registers of the ATV21 Modbus Device**

- **Address** = 11410
- **Object** = Variable SPF
- **Quantity** = 200 SPF registers
- **StartPosition** = 0
- **Quantity** = 200 SPF registers

**Variables**:
- **v_WriteSPF_SPF0**
- **v_ReadSPF_SPF0**

Upon completion of the SPF to SPW WRITE & READ operation, the read values of the SPF to SPW registers are converted to WORD TO UINT for variable SPF.

### Variable SPF Conversion

<table>
<thead>
<tr>
<th>WORD TO UINT</th>
<th>SPF Converted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_WriteSPF_SPF0Values[0]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[1]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[2]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[3]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[4]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[5]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
<tr>
<td>v_WriteSPF_SPF0Values[6]</td>
<td>v_WriteSPF_SPF0Value</td>
</tr>
</tbody>
</table>

#### Variables

- **v_WriteSPF_SPF0**
- **v_ReadSPF_SPF0**
6.3. CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Creation of the POU</strong>: Create a new POU in CFC language, called PLC_PRG_CFC.</td>
</tr>
</tbody>
</table>

Upon creation of this POU, it is automatically opened by SoMachine.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>CFC variables</strong>: In the upper part of the CFC editor, declare the following variables:</td>
</tr>
</tbody>
</table>

```plaintext
PROGRAM PLC_PRG_CFC
VAR
    (************************)
    (*** TON Variables ***)
    (************************)
    // TON Function Block for delaying the start of this program
    TON_START                   : TON;
    // Delayed Rising Edge signal for starting the program
    v_xStartProgram            : BOOL := FALSE;

    (************************)
    (*** ADDM Variables ***)
    (************************)
    // ADDM Function Block for formatting the address of the ATV12 Modbus Slave
    ADDM_MODBUS_ATV12           : ADDM;
    // ADDRESS Structure for the address of the ATV12 Modbus Slave
    v_addressModbusAtv12        : ADDRESS;
    // "Done" result of the Address conversion
    v_xAddressDone              : BOOL := FALSE;
    // "Error" result of the Address conversion
    v_xAddressError             : BOOL := FALSE;
    // Result of the Address conversion: OK if "Done" without any "Error"
    v_xAddressIsOK              : BOOL := FALSE;

    (************************)
    (*** ETA Register Variables ***)
    (************************)
    // BLINK Register Variables
    BLINK_ETA                   : BLINK;
```
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| // "OUT" output of the BLINK Function Block: Clock  
  v_wClockReadETARegister : BOOL := FALSE; |
| // Command to read the ETA register  
  v_xReadETARegister : BOOL := FALSE; |
| // READ_VAR Function Block for reading the ETA register of the ATV12 Modbus device  
  READ_VAR ETA : READ_VAR;  
  // Buffer for the value of the ETA register  
  v_wRegisterETA : WORD := 0;  
  // "Done" result of the ETA register read operation  
  v_xReadETADone : BOOL := FALSE;  
  // "Busy" output of the ETA register read operation  
  v_xReadETABusy : BOOL := FALSE;  
  // "Error" result of the ETA register read operation  
  v_xReadETAError : BOOL := FALSE;  
  // "CommError" result of the ETA register read operation  
  v_bReadETACommError : BYTE := CommunicationErrorCodes.CommunicationOK;  
  // Result of the ETA register read operation: SUCCESS  
  v_xReadETASuccess : BOOL := FALSE;  
  // Result of the ETA register read operation: FAILURE  
  v_xReadETAFailure : BOOL := FALSE; |
| // Presence (TRUE) or absence (FALSE) of the ATV12 Modbus device  
  v_xPresenceAtv12 : BOOL := FALSE;  
  // Communication error with the ATV12 Modbus device (Timeout excluded)  
  v_xCommErrorAtv12 : BOOL := FALSE; |
| (***) LFR Register Variables (***)  
  (********************************************************************)  
  // MANUAL command for starting one LFR register write operation  
  v_xCmdManualWriteLFR : BOOL := FALSE;  
  // Automatic command for reading the new value of the LFR register  
  v_xCmdAutoReadLFR : BOOL := FALSE;  
  // Command to write the LFR register  
  v_xWriteLFRRegister : BOOL := FALSE;  
  // Value of the LFR register to write on the device (-400.0 Hz to +400.0 Hz; unit: 0.1 Hz)  
  v_iWriteLFRValue : INT := 0;  
  // SINGLE_WRITE Function Block for writing the LFR register of the ATV12 Modbus device  
  // Value of the LFR register to write on the device (after conversion to WORD)  
  v_wWriteLFRValue : WORD := 0;  
  // "Busy" output of the LFR register write operation  
  v_xWriteLFRBusy : BOOL := FALSE;  
  // Command to read the LFR register  
  v_xReadLFRRegister : BOOL := FALSE; |
| // ACC & DEC Registers Variables  
  (********************************************************************)  
  // MANUAL command for starting one ACC & DEC registers write operation  
  v_xCmdManualWriteACC_DEC : BOOL := FALSE;  
  // Automatic command for reading the new values of the ACC & DEC registers  
  v_xCmdAutoReadACC_DEC : BOOL := FALSE; |
6. ST, LD, or CFC Program

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Command to write the ACC &amp; DEC registers</td>
<td>v_xWriteACC_DECRegisters : BOOL := FALSE;</td>
</tr>
<tr>
<td>// Value of the ACC &amp; DEC registers to write on the device (0.0 s to 999.9 s; unit: 0.1 s)</td>
<td>v_iWriteACCValue : INT := 0;</td>
</tr>
<tr>
<td>v_iWriteDECV Value : INT := 0;</td>
<td></td>
</tr>
<tr>
<td>// WRITE_VAR Function Block for writing the ACC &amp; DEC registers of the ATV12 Modbus device</td>
<td>WRITE_VAR_ACC_DEC : WRITE_VAR;</td>
</tr>
<tr>
<td>// Buffer for the values of the ACC &amp; DEC registers to write on the device (after conversion to WORD)</td>
<td>v_wWriteACC_DECValues : ARRAY [0..1] OF WORD := [0,0];</td>
</tr>
<tr>
<td>// &quot;Busy&quot; output of the ACC &amp; DEC registers write operation</td>
<td>v_xWriteACC_DECBusy := BOOL := FALSE;</td>
</tr>
<tr>
<td>// Command to read the ACC &amp; DEC registers</td>
<td>v_xReadACC_DECRegisters := BOOL := FALSE;</td>
</tr>
<tr>
<td>// READ_VAR Function Block for reading the new value of the ACC &amp; DEC registers of the ATV12 Modbus device</td>
<td>READ_VAR_ACC_DEC := READ_VAR;</td>
</tr>
<tr>
<td>// Buffer for the values of the ACC &amp; DEC registers</td>
<td>v_wRegisterACC_DEC := ARRAY [0..1] OF WORD := [0,0];</td>
</tr>
<tr>
<td>// &quot;Busy&quot; output of the ACC &amp; DEC registers read operation</td>
<td>v_xReadACC_DECBusy := BOOL := FALSE;</td>
</tr>
<tr>
<td>// Value of the ACC &amp; DEC registers read from the device (0.0 s to 999.9 s; unit: 0.1 s)</td>
<td>v_iReadACCValue := INT := 0;</td>
</tr>
<tr>
<td>v_iReadDECVValue := INT := 0;</td>
<td></td>
</tr>
</tbody>
</table>
| **************************
| *** SP2 to SP8 Registers Variables ***
| **************************
| // MANUAL command for starting one SP2 to SP8 registers write & read operation | v_xCmdManualWrRdSP2_SP8 := BOOL := FALSE; |
| // Command to write & read the SP2 to SP8 registers | v_xWrRdSP2_SP8Registers := BOOL := FALSE; |
| // Values of the SP2 to SP8 registers to write on the device (0.0 Hz to 400.0 Hz; unit: 0.1 Hz) | v_uiWriteSP2Value := UINT := 0; |
| v_uiWriteSP3Value := UINT := 0; |
| v_uiWriteSP4Value := UINT := 0; |
| v_uiWriteSP5Value := UINT := 0; |
| v_uiWriteSP6Value := UINT := 0; |
| v_uiWriteSP7Value := UINT := 0; |
| v_uiWriteSP8Value := UINT := 0; |
| // WRITE_READ_VAR Function Block for writing & reading the SP2 to SP8 registers of the ATV12 Modbus device | WRITE_READ_VAR_SP2_SP8 := WRITE_READ_VAR; |
| // Buffer for the values of the SP2 to SP8 registers to write on the device (after conversion to WORD) | v_wWriteSP2_SP8Values := ARRAY [0..6] OF WORD := [7(0)]; |
| // Buffer for the values of the SP2 to SP8 registers read from the device | v_wReadSP2_SP8Values := ARRAY [0..6] OF WORD := [7(0)]; |
| // "Busy" output of the SP2 to SP8 registers write & read operation | v_xWriteSP2_SP8Busy := BOOL := FALSE; |
| // Values of the SP2 to SP8 registers read from the device (0.0 Hz to 400.0 Hz; unit: 0.1 Hz) | v_uiReadSP2Value := UINT := 0; |
| v_uiReadSP3Value := UINT := 0; |
| v_uiReadSP4Value := UINT := 0; |
| v_uiReadSP5Value := UINT := 0; |
| v_uiReadSP6Value := UINT := 0; |
| v_uiReadSP7Value := UINT := 0; |
| v_uiReadSP8Value := UINT := 0; |

END_VAR

---

52  EIO0000000913.01 04/2012
### CFC program

In the lower part of the CFC editor, implement the following program:

```plaintext
TOK Function Block for delaying the start of the program

- **Step:** 3
- **Action:** CFC program
  - In the lower part of the CFC editor, implement the following program:
    - `TOK`: Duration of the TOK time (1 second).
    - `TOK = Recording delay before running the rest of the program`.
    - `ADDW Function Block for accessing the address of the ATV72 Modbus Slave Function Block(s): ADW`
      - **Task:** ADDW
        - **Address:** Modbus Slave Address Format (communication port number + slave address)
      - **Result:** Address conversion OK if Address without any Error
    - **Periodic Communication ETA Register read once per second Function Block(s): READ_VAR`
      - **Task:** READ_VAR
        - **Address:** ETA register (TOK module)
        - **Description:** Command to read the ETA register
          - **Variable:** ETA
          - **Function:** Modbus Slave Address Format (communication port number + slave address)
        - **Result:** ETA register conversion OK
    - **READ_VAR Function Block for reading the ETA register of the ATV72 Modbus device**
      - **Description:** Delta or Delta_H + Delta_H value
        - **Variable:** ETA
        - **Function:** Modbus Slave Address Format (communication port number + slave address)
      - **Result:** ETA register conversion OK
    - **On-demand command:** LTR Register write + LTR Register Read Function Blocks: SINGLE_WRITE + READ_VAR
      - **Description:** Conversion from INT to WORD of the LTR register value to write on the device
        - **Variable:** INT_TO_WORD
      - **Function:** Modbus Slave Address Format (communication port number + slave address)
6. ST, LD, or CFC Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command to write the LTR register. S'Address is OK AND Handl Command to Write LTR</td>
<td>v_addrWriteLFRRegister AND v_4WriteLFRRegister</td>
</tr>
<tr>
<td>SINGLE_WRITE Function block for setting the LTR register of the AYV2 Modbus device</td>
<td>v_4WriteLFRRegister Execute Dump Addr Actual Object = New Value Object = Old Value Mode = OpenSeq V</td>
</tr>
<tr>
<td>Upon completion of the LWR WRITE operation, an automatic LWR READ operation is performed to check if the value of the LTR register has been updated</td>
<td>AND v_4WriteLFRRegister v_4WriteLFRRegister Int o = 0 Command to read the LTR register. S'Address is OK AND Automatic Command to Read LTR</td>
</tr>
<tr>
<td>READ_VAR Function Block to read the new value of the LTR register of the AYV2 Modbus device</td>
<td>EXECUTE = Rising Edge signal that triggers the Function Block v ServiceProvider = 4 READ_VAR v_addrReadLFRRegister v_4ReadLFRRegister int o = 0 Command to read the LTR register. S'Address is OK AND Automatic Command to Read LTR</td>
</tr>
<tr>
<td>Upon completion of the LTR READ operation, both LWR WRITE and LTR READ operations are ended AND the read value is converted from WORD to INT</td>
<td>AND v_4ReadLFRRegister v_4ReadLFRRegister v_wordToInt16 v_wordToInt16 value v_wordToInt16 INT_TO_WORD v_4ReadLFRRegister INT_TO_WORD</td>
</tr>
<tr>
<td>Conversion from WORD to INT of the LTR register value read on the device. NOTE: The conversion must preserve the w sign in the case of the LTR register</td>
<td>v_wordToINT16 v_wordToINT16 INT v_wordToINT16 INT v_wordToINT16 INT</td>
</tr>
</tbody>
</table>
6. ST, LD, or CFC Program

Command to write the ACC & DEC registers
- Address is OK AND Keypad Command to write ACC & DEC

WRITE_VAR Function Block for writing the ACC & DEC registers of the ATV7A Modbus device

WRITE_VAR

- Execute = Rising Edge signal that triggers the Function Block
- Addr = Completed address of the ATV7A Modbus device
- CNT=0 = Type of object to be read/WR
- The Modbus function 16 (write multiple registers) is used
- Profile = Quantity = (Object to be written) ACC & DEC registers addresses = [0x0000, 0x0001, 0x0002, 0x0003]
- Buffer = Address of the registers which values will be sent to the ACC & DEC registers

READ_VAR Function Block for reading the new values of the ACC & DEC registers of the ATV7A Modbus device

READ_VAR

- Execute = Rising Edge signal that triggers the Function Block
- Addr = Completed address of the ATV7A Modbus device
- CNT=1 = Type of object to be read/WR
- The Modbus function 04 (read holding registers) is used
- Profile = Quantity = (Object to be read) ACC & DEC registers addresses = [0x0000, 0x0001, 0x0002, 0x0003]
- Buffer = Address of the registers to store the values of the ACC & DEC registers

Conversion from WORD to INT of the ACC & DEC registers values read on the device

WORD_TO_INT

- IN = WORD to INT
- END = END
- OUT = INT to WORD

Conversion from INT to WORD of the SPS to SPS registers values to write on the device

INT_TO_WORD

- IN = INT to WORD
- OUT = WORD to INT
### 6. ST, LD, or CFC Program

**Step** | **Action**
--- | ---

When you have finished implementing this CFC program, perform the following steps to correctly set the execution order of its blocks:

- Right-click on an empty area of the central worksheet.
- In the **Execution Order** part of the contextual menu, execute the **Order By Data Flow** command.
7. Configuration of the ATV12 Modbus Slave

The steps listed in the following table describe how to configure the parameters and the Modbus communications of the Altivar 12 (ATV12) variable speed drive used in this example. Please refer to the Altivar 12 Variable speed drives for asynchronous motors User manual and to the Altivar 12 Variable speed drives for asynchronous motors Modbus Communication Manual for further information on how to configure an ATV12 variable speed drive using the HMI of its front panel.

The instructions given herein include the steps for resetting your Altivar variable speed drive to its factory settings.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Switch on the Altivar 12 variable speed drive.</td>
</tr>
<tr>
<td>2</td>
<td>Reset the Altivar 12 to its factory setting:</td>
</tr>
<tr>
<td></td>
<td>➢ Press the ENT key.</td>
</tr>
<tr>
<td></td>
<td>➢ Enter the Configuration Mode ConF menu (HMI display: COIF)</td>
</tr>
<tr>
<td></td>
<td>➢ Edit the Factory/ recall customer parameter set (FCS) parameter (HMI display: FCS)</td>
</tr>
<tr>
<td></td>
<td>➢ Select the InI value for resetting the Altivar 12 drive to its factory setting (HMI display: InI)</td>
</tr>
<tr>
<td></td>
<td>➢ Press the ENT key during 2 seconds to activate the factory setting.</td>
</tr>
<tr>
<td></td>
<td>➢ Press the ESC key to exit the Configuration Mode ConF menu.</td>
</tr>
<tr>
<td></td>
<td>➢ Press the ESC key.</td>
</tr>
</tbody>
</table>

DANGER

UNINTENDED EQUIPMENT OPERATION

Before resetting your Altivar variable speed drive to its factory settings, check that the modification of its current configuration is compatible with the wiring diagram used.

Failure to follow these instructions will result in death or serious injury.

NOTE: Since this example is for learning purposes only, it is strongly recommended to install and operate the Altivar drive without wiring it to any motor. This is particularly true if you inhibit its motor output phase loss detection via the Output Phase Loss (OPL) parameter.

DANGER

UNINTENDED EQUIPMENT OPERATION

Before installing or operating an Altivar 12 variable speed drive and/or a motor, read and understand the Altivar 12 Variable speed drives for asynchronous motors User manual.

Failure to follow these instructions will result in death or serious injury.
## 7. Configuration of the ATV12 Modbus Slave

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 3 | **Optional step**  
This step shows you how to configure the control mode of the Altivar 12 so the variable speed drive can be controlled by a Modbus Master. This step is not necessary in the case of this example but could prove useful if you want your controller to control this drive:  
- Press the **ENT** key.  
- Enter the **Configuration Mode ConF** menu (HMI display: **COnF**)  
- Enter the **Complete menu** (FULL) (HMI display: **FULL**)  
- Enter the **Control menu** (Ctl-) (HMI display: **CtL-**)  
- Edit the **Reference channel 1** (Fr1) parameter (HMI display: **Fr1**)  
- Select the **Modbus** (Mdb) value to configure the drive to connect its **Reference channel 1** to the integrated Modbus port (HMI display: **Adb**)  
- Press the **ENT** key.  
- Press the **ESC** key to exit the **Control menu** (Ctl-).  
- Press the **ESC** key to exit the **Complete menu** (FULL).  
- Press the **ESC** key to exit the **Configuration Mode ConF** menu.  
- Press the **ESC** key. |
| 4 | Set the Modbus configuration and address of the Altivar 12:  
- Press the **ENT** key.  
- Enter the **Configuration Mode ConF** menu (HMI display: **COnF**)  
- Enter the **Complete menu** (FULL) (HMI display: **FULL**)  
- Enter the **Communication menu** (COM-) (HMI display: **COA-**)  
- Edit the **Modbus address** (Add) parameter (HMI display: **Add**)  
- Select the **2** value to set the Altivar drive to the Modbus address 2 (HMI display: **2**)  

**NOTE:** This value must be unique for the devices of the same Modbus network. It corresponds to the value used by the **ADDM_MODBUS_ATV12** Function Block instance of this program (see page 67) for addressing the Altivar 12 drive.  
- Press the **ENT** key.  
- Edit the **Modbus baud rate** (tbr) parameter (HMI display: **tbr**)  
- If the HMI display shows **19 2**:  
  - Press the **ESC** key.  
- If the HMI display does not show **19 2**:  
  - Select the **19.2 kbps** (19.2) value to configure the Altivar drive to communicate on the Modbus network at 19200 Bauds (HMI display: **19 2**)  

**NOTE:** This value corresponds to the **Baud rate** parameter of the controller’s **Modbus_Manager** device in the SoMachine project (see page 28).  
- Press the **ENT** key. |
### 7. Configuration of the ATV12 Modbus Slave

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢</td>
<td>Edit the <strong>Modbus format</strong> (tFO) parameter (HMI display: <em>tFO</em>)</td>
</tr>
<tr>
<td>➢</td>
<td>Select the <strong>8N1</strong> value to configure the Altivar drive to communicate on the Modbus network with the following settings: 8 data bits, no parity, and 1 stop bit (HMI display: <em>8n1</em>)</td>
</tr>
<tr>
<td>➢</td>
<td><strong>NOTE:</strong> This value corresponds to the <strong>Parity</strong>, <strong>Data bits</strong>, and <strong>Stop bits</strong> parameters of the controller’s <strong>Modbus_Manager</strong> device in the SoMachine project (see page 28).</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ENT</strong> key.</td>
</tr>
<tr>
<td>➢</td>
<td>Edit the <strong>Modbus time out</strong> (ttO) parameter (HMI display: <em>ttO</em>)</td>
</tr>
<tr>
<td>➢</td>
<td>Select the <strong>30.0</strong> value to set a Modbus timeout of 30 seconds on the Altivar drive (HMI display: <em>30.0</em>)</td>
</tr>
<tr>
<td>➢</td>
<td><strong>NOTE:</strong> With this configuration, the Altivar 12 drive detects a Modbus communication fault (SLF1) if it does not receive any Modbus request at its address within a 30 second limit. This Modbus timeout is only started when the Altivar 12 drive receives a Modbus <strong>write</strong> request (e.g. Write Single Register). Modbus <strong>read</strong> requests (e.g. Read Holding Registers) do not start this timeout, but once it has been started, the configured 30 second timeout is restarted each time the Altivar 12 drive receives a read and/or write Modbus request at its address.</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ENT</strong> key.</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ESC</strong> key to exit the <strong>Communication menu</strong> (COM-).</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ESC</strong> key to exit the <strong>Complete menu</strong> (FULL).</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ESC</strong> key to exit the <strong>Configuration Mode ConF</strong> menu.</td>
</tr>
<tr>
<td>➢</td>
<td>Press the <strong>ESC</strong> key.</td>
</tr>
</tbody>
</table>
### 7. Configuration of the ATV12 Modbus Slave

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 5    | Check the settings of the Modbus fault management on the Altivar 12:  
|      | ➢ Press the **ENT** key.  
|      | ➢ Enter the **Configuration Mode ConF** menu (HMI display: **CONF**)  
|      | ➢ Enter the **Complete menu** (FULL) (HMI display: **FULL**)  
|      | ➢ Enter the **Fault detection management menu** (FLt-) (HMI display: **FLt-**)  
|      | ➢ Edit the **Modbus fault management** (SLL) parameter (HMI display: **SLL**)  
|      | ➢ If the HMI display shows **YES**:  
|      |       • Press the **ESC** key.  
|      | ➢ If the HMI display does not show **YES**:  
|      |       • Select the **Freewheel stop** (YES) value to configure the Altivar 12 drive to report a fault with the freewheel stop if it detects a Modbus communication fault (SLF1) (HMI display: **YES**)  
|      |       • Press the **ENT** key.  
|      | ➢ Press the **ESC** key to exit the **Fault detection management menu** (FLt-).  
|      | ➢ Press the **ESC** key to exit the **Complete menu** (FULL).  
|      | ➢ Press the **ESC** key to exit the **Configuration Mode ConF** menu.  
|      | ➢ Press the **ESC** key. |
| 6    | Inhibit the motor output phase loss detection of the Altivar 12:  
|      | ➢ Press the **ENT** key.  
|      | ➢ Enter the **Configuration Mode ConF** menu (HMI display: **CONF**)  
|      | ➢ Enter the **Complete menu** (FULL) (HMI display: **FULL**)  
|      | ➢ Enter the **Fault detection management menu** (FLt-) (HMI display: **FLt-**)  
|      | ➢ Edit the **Output Phase loss** (OPL) parameter (HMI display: **OPL**)  
|      | ➢ Select the **NO** value to deactivate the motor output phase loss fault detection (HMI display: **NO**)  
|      | ➢ Press the **ENT** key.  
|      | ➢ Press the **ESC** key to exit the **Fault detection management menu** (FLt-).  
|      | ➢ Press the **ESC** key to exit the **Complete menu** (FULL).  
|      | ➢ Press the **ESC** key to exit the **Configuration Mode ConF** menu.  
|      | ➢ Press the **ESC** key. |
### 7. Configuration of the ATV12 Modbus Slave

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Optional step</td>
</tr>
</tbody>
</table>

If you intend to use a motor with your Altivar 12 variable speed drive, you should update the values of the various parameters of its **Motor control menu** (drC-) with the values given on the motor rating plate.

The path of this menu, on the HMI display, is: **Configuration Mode (ConF) ➤ Complete menu (FULL) ➤ Motor control menu (drC-)**

Please refer to the *Altivar 12 Variable speed drives for asynchronous motors User manual* for a description of these parameters and their possible values.
## 8. Running the Example

### 8.1. MAST Task Configuration

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | In the **Devices** tree view:  
- Expand the contents of your controller.  
- Double-click on the **MAST** task of the **Task Configuration** item.  
- Click on the **Add POU** command. |

![Image of Devices tree view with MAST task highlighted]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 2      | In the **Input Assistant** window that is displayed:  
- Select your program.  
  In the following dialog, the **ST Program** (see page 32), **PLC_PRG_ST**, is selected: |

![Image of Input Assistant window with PLC_PRG_ST selected]
### 8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Click <strong>OK</strong>. This adds the selected POU to the list of programs run by the MAST task of the controller.</td>
</tr>
</tbody>
</table>
8. Running the Example

8.2. Downloading the Example to the Controller

The steps listed in the following table describe how to download the example to the controller. If needed, please refer to the SoMachine online help for further information on these steps: search for Communication Settings.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1 | **Connect the USB programming cable between your PC and your controller.**
   | Please refer to Hardware Installation (see page 20), for the reference and usage of this cable.
   | **NOTE:** This USB programming cable is compatible with any LMC058, M238, or M258 controller. Each of these controllers has a mini B USB port designed to program it using SoMachine. |
| 2 | In the Devices tree view, double-click on your controller's device to open its configuration panel.
   | Example: Click on the LMC058_Controller (LMC058LF42S0) item in the case of a LMC058 controller. |
| 3 | In the Communication Settings tab of this panel:
   |   - Click on the Gateway-1 node.
   |   - Click on the Scan network button. |
   |   | If the controller is switched on and connected to your PC with the USB programming cable, it will be detected by SoMachine as shown below: |

In this example, the **LMC058LF42S0 @0080F4400CBC** controller has been detected.
8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Select this controller’s node and click on the <strong>Set active path</strong> button. This will show SoMachine the way to reach your controller. <strong>Option:</strong> Uncheck the <strong>Secure online mode</strong> box to avoid validation messages during future online modifications. <strong>NOTE:</strong> The firmware version of your controller is displayed as the <strong>Target Version</strong>.</td>
</tr>
</tbody>
</table>
| 5    | The following window will display and prompts you to confirm your choice:  
- Read the hazard message;  
- Simultaneously press the `<Alt>` and `<F>` keys to validate your choice and close this window;  
- Or, click on the **Cancel** button if you can not comply with the statements in the hazard message. |

**WARNING**

**UNINTENDED EQUIPMENT OPERATION**

Ensure that the software application being downloaded is installed on the intended device. Confirm you have entered the correct device designation or device address.

Ensure guards are in place so that unintended equipment operation will not cause injury to personnel or damage to equipment.

Read and understand the software User Manual, and know how to operate the equipment.

Failure to follow these instructions can result in death, serious injury or equipment damage.

If you agree to follow these instructions, press ‘Alt+F’.
### 8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Click on the <img src="Image" alt="Login" /> button of the toolbar to establish a connection from SoMachine to the controller.</td>
</tr>
</tbody>
</table>
| 7a   | If you download the current project to the controller for the first time and if there is no project on the controller, the following window is displayed:  

```
No program on the targets. Do you want to perform a download?
```

Click on **Yes** to download the **Application** software to the controller. |
| 7b   | If you download the current project to the controller for the first time and if there is another project on the controller, the following window is displayed:  

```
Unknown version of Application 'Application' on targets. Do you want to perform a download and replace the application?
```

Click on **Yes** to download the **Application** software to the controller. |
| 7c   | If you already have downloaded the current project to the controller, the following window displays if new modifications were made to this project:  

```
The code has been changed since the last download. What do you want to do?
```

- Select the ![Login with download](Image) choice.
- Click on **OK** to download the **Application** software to the controller. |
| 7d   | If you already have downloaded the current project to the controller, the connection is immediate if no modifications were made to this project. |
| 8    | Wait for the completion of the download operation. Once it is finished, the status bar of SoMachine displays the state of the controller: ![STOP](Image) |
| 9    | Click on the ![Start](Image) button of the toolbar to run the **Application** software on the controller.  

The state of the controller switches from ![STOP](Image) to ![RUN](Image) |
8.3. Running the Example on the Controller

The steps listed in the following table describe how to use the example, once it has been downloaded to the controller.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Devices panel, double-click on your PLC_PRG program to open it (e.g. PLC_PRG_ST if you previously selected this program as the program run by the MAST task of your controller). This command opens your program in the central panel of SoMachine; the online values of the variables used by this program are also displayed, as shown below:</td>
</tr>
</tbody>
</table>
| 2    | Addressing of the Modbus slave using the **ADDM** function block: Name of the ADDM Function Block instance: **ADDM_MODBUS_ATV12**  

The **ADDM_MODBUS_ATV12** instance is used to format the address of the Altivar 12 drive (**<slave address>= address = 2**) on the Modbus serial line (**<communication port number>= serial line #1 = 1**). To do so, this address is set into the Addr input of this instance, with the following format (in the case of Modbus communications): 

'**<communication port number>.<slave address>'**

Here, the Addr input is set to '1.2'.

**NOTE:** Do not forget to configure this address on the Altivar 12 variable speed drive, as described in **Configuration of the ATV12 Modbus Slave (see page 58)**.

To test this Function Block instance, proceed as follows:

- Check that this **ADDM_MODBUS_ATV12** instance works correctly:
  - Its Done output (or the **v_xAddressDone** variable) must be equal to TRUE.
  - Its Error output (or the **v_xAddressError** variable) must be equal to FALSE.
  - Its AddrTable structured output (or the **v_addressModbusAtv12** variable) contains the formatted address that will be used in the rest of this program to address the Altivar 12 drive on the Modbus network.
### 8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 3    | **Periodic read request of one parameter (ETA) of the ATV12 drive using the READ_VAR function block:**  
|      | Name of the READ_VAR Function Block instance: READ_VAR_ETA  
|      | A BLINK_ETA instance of the BLINK Function Block is used to periodically trigger this read request (once per second).  
|      | This BLINK_ETA instance triggers the execution of the READ_VAR_ETA instance. The outputs of this READ_VAR_ETA instance are then used to report the Modbus connection state between the controller and the Altivar 12 drive:  
|      | - The v_xPresenceAtv12 variable is set to TRUE and the v_xCommErrorAtv12 variable is reset to FALSE if the Modbus cable is connected between the controller and the Altivar 12 drive.  
|      | - The v_xPresenceAtv12 variable is reset to FALSE if the Modbus cable is disconnected.  
|      | - The v_xCommErrorAtv12 variable is set to TRUE if a communication error occurs (timeout excluded).  
|      | To test this Function Block instance, proceed as follows:  
|      | - Check that the v_xPresenceAtv12 variable is equal to TRUE and that the v_xCommErrorAtv12 variable is equal to FALSE.  
|      | - Disconnect the Altivar 12 drive from the Modbus network by unplugging its Modbus RS485 cordset (ref. VW3 A8 306 R++).  
|      | - Check that the v_xPresenceAtv12 variable is equal to FALSE and that the v_xCommErrorAtv12 variable is equal to FALSE.  
|      | - Plug back the Modbus RS485 cordset to the Altivar 12 drive.  
|      | - Check that the v_xPresenceAtv12 variable is equal to TRUE and that the v_xCommErrorAtv12 variable is equal to FALSE.  
|      | **NOTE:** For now, the Altivar 12 drive does not report any SLF1 fault, even if the duration of the disconnection is greater than the duration configured for its Modbus timeout parameter (see page 59). |
| 4    | **Aperiodic write request of one parameter (LFR) of the ATV12 drive using the SINGLE_WRITE function block:**  
|      | Name of the SINGLE_WRITE Function Block instance: SINGLE_WRITE_LFR  
|      | To test this SINGLE_WRITE_LFR instance, proceed as follows:  
|      | - Change the value of the v_iWriteLFRValue variable: its unit is 0.1 Hz and it ranges from −3,276.7 Hz to +3,276.7 Hz (value = −32,767 to +32,767).  
|      | - Set the v_xCmdManualWriteLFR variable to TRUE.  
|      | This triggers the execution of the SINGLE_WRITE_LFR instance, followed by the execution of a READ_VAR_LFR instance of the READ_VAR Function Block. These Function Block instances update the value of the LFR parameter on the Altivar 12 drive and read its updated value.  
|      | The v_xCmdManualWriteLFR variable is automatically reset to FALSE by the program after being used by the SINGLE_WRITE_LFR and READ_VAR_LFR Function Block instances. |
8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td> Check the value of the <code>v_iReadLFRValue</code> variable: if this value is equal to the <code>v_iWriteLFRValue</code> variable (positive or negative value), then the write operation is successful.</td>
</tr>
</tbody>
</table>

To test the Modbus timeout of the Altivar 12 drive, proceed as follows:

- Disconnect the Altivar 12 drive from the Modbus network by unplugging its Modbus RS485 cordset (ref. VW3 A8 306 R++).
- Wait at least 30 seconds.
- Check that the Altivar 12 drive reports a SLF1 fault (HMI display: **SLF1**) because the duration of the disconnection is greater than the duration configured for its **Modbus time out** (tT0) parameter (see page 59) and because a Modbus write request has been received by the Altivar 12 drive.
- Plug back the Modbus RS485 cordset to the Altivar 12 drive.
- Optional step: switch off, then switch on, the Altivar 12 drive to reset this fault.

<table>
<thead>
<tr>
<th>5</th>
<th>Aperiodic write request of two parameters (ACC and DEC) of the ATV12 drive using the <code>[WRITE_VAR]</code> function block:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name of the WRITE_VAR Function Block instance: <strong>WRITE_VAR_ACC_DEC</strong></td>
</tr>
<tr>
<td></td>
<td>To test this <strong>WRITE_VAR_ACC_DEC</strong> instance, proceed as follows:</td>
</tr>
<tr>
<td></td>
<td> Change the value of the <code>v_iWriteACCValue</code> and <code>v_iWriteDECValue</code> variables: their unit is 0.1 s and they range from 0.1 s to 999.9 s (value = 1 to 9,999).</td>
</tr>
<tr>
<td></td>
<td> Set the <code>v_xCmdManualWriteACC_DEC</code> variable to TRUE.</td>
</tr>
<tr>
<td></td>
<td>This triggers the execution of the <strong>WRITE_VAR_ACC_DEC</strong> instance, followed by the execution of a <strong>READ_VAR_ACC_DEC</strong> instance of the <strong>READ_VAR</strong> Function Block. These Function Block instances update the values of the ACC and DEC parameters on the Altivar 12 drive and read their updated values.</td>
</tr>
<tr>
<td></td>
<td>The <code>v_xCmdManualWriteACC_DEC</code> variable is automatically reset to FALSE by the program after being used by the <strong>WRITE_VAR_ACC_DEC</strong> and <strong>READ_VAR_ACC_DEC</strong> Function Block instances.</td>
</tr>
<tr>
<td></td>
<td> Check the values of the <code>v_iReadACCValue</code> and <code>v_iReadDECCValue</code> variables: if their values are equal to the <code>v_iWriteACCValue</code> and <code>v_iWriteDECValue</code> variables, then the write operation is successful.</td>
</tr>
</tbody>
</table>
### 8. Running the Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 6    | **Aperiodic read/write request of seven parameters (SP2 to SP8) of the ATV12 drive using the `WRITE_READ_VAR` function block:**  
Name of the WRITE_READ_VAR Function Block: `WRITE_READ_VAR_SP2_SP8`  
To test this `WRITE_READ_VAR_SP2_SP8` instance, proceed as follows:  
- Change the values of the seven `v_uiWriteSP2Value` to `v_uiWriteSP8Value` variables: their unit is 0.1 Hz and they range from 0.0 Hz to 400.0 Hz (value = 0 to 4,000).  
- Set the `v_xCmdManualWrRdSP2_SP8` variable to TRUE.  
  This triggers the execution of the `WRITE_READ_VAR_SP2_SP8` instance. This Function Block instance updates the values of the SP2 to SP8 parameters on the Altivar 12 drive and reads their updated values.  
  **NOTE:** Both write and read operations are performed in a single Modbus exchange.  
  The `v_xCmdManualWrRdSP2_SP8` variable is automatically reset to FALSE by the program after being used by the `WRITE_READ_VAR_SP2_SP8` Function Block instance.  
- Check the values of the `v_uiReadSP2Value` to `v_uiWriteSP8Value` variables: if their values are equal to the `v_uiWriteSP2Value` to `v_uiWriteSP8Value` variables, then the read/write operation is successful. |