RIOM

Remote Input Output Module
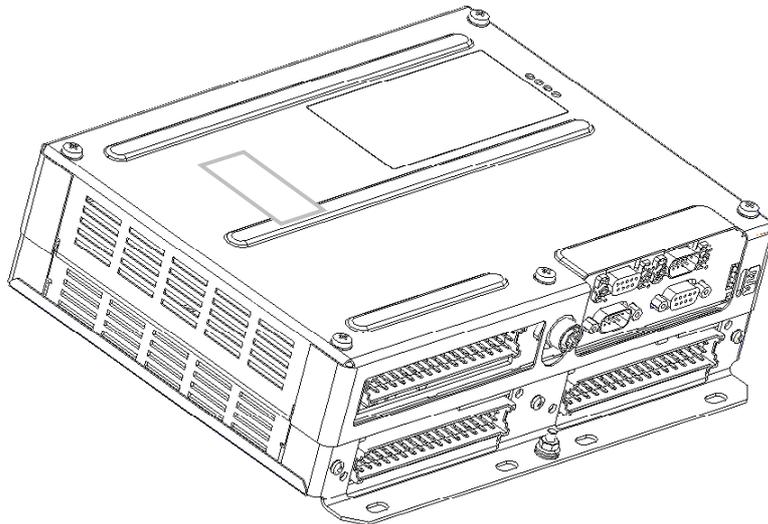
Overview

RIOM is a programmable PLC available in language IEC 61131-3 ISaGRAF V5 workbench from ICS Triplex.

RIOM hardware implementation is explained in the installation guide.

Prerequisite

The developpement of applications on RIOM ISaGRAF requires knowledge of programming in IEC61131-3 languages.

The actual implementation of the RIOM requires skills in electricity and industrial automation.

Required equipment is:

- A development PC running Windows XP, with an RS232 port and an Ethernet network card.
- A RS232 cross cable (2-3, 3-2, 5-5 only) female and male SubD9 connectors.
- An Ethernet cable : RJ45 and M12 connectors.

Version

This documentation describes features included in ISaGRAF RIOM.

Property

RIOM is a registered trademark of Leroy Automatique Industrielle.
ISaGRAF is a registered trademark of ICS Triplex.
Windows XP is a trademark of Microsoft Corporation.
LEROY Automatique Industrielle is constantly developing and improving its products. The information contained herein is Action to change without notice and is in no way legally binding upon the company. This manual may not be duplicated in any form without the prior consent of LEROY Automatique Industrielle.

Contact

Contents

General Overview

Introduction

This chapter describes the basics of RIOM. We detail in this chapter :

- RIOM hardware,
- RIOM software

RIOM hardware

Application on RIOM uses all of its components hardware base:

Product nameplate containing product
reference, its serial number, its index, its
manufacture data code and its Ethernet
MAC address

Product software
downloaded label

CPU board

Peripheral
board

I/O board 1
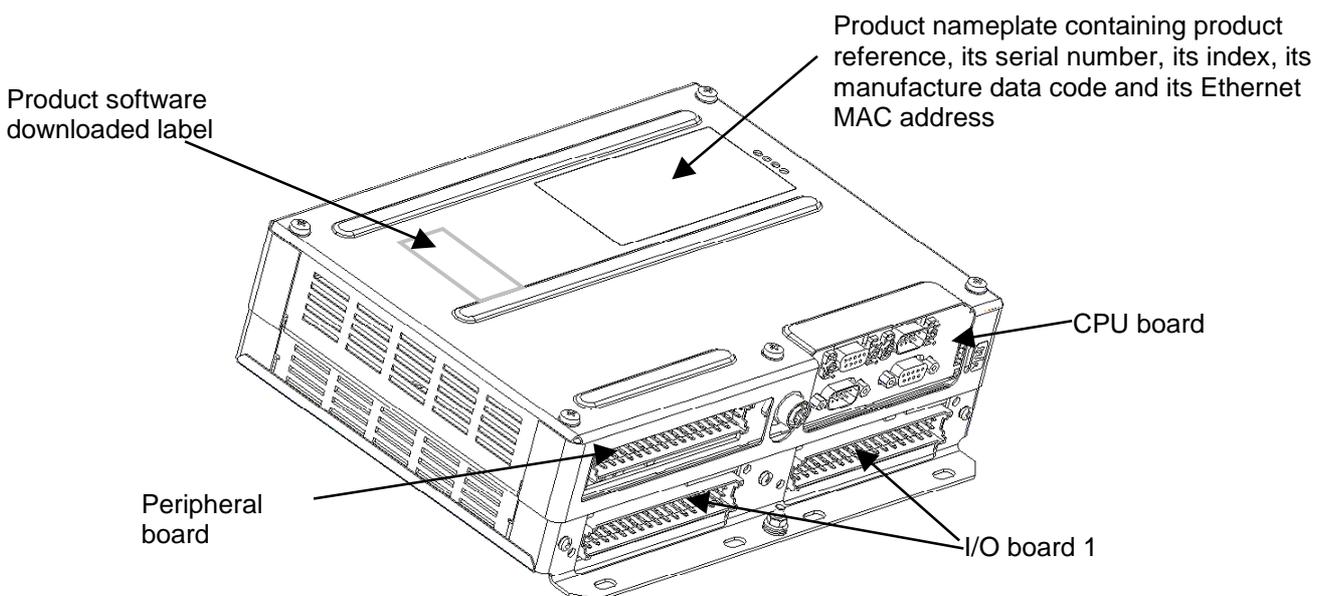
RIOM embedded software

RIOM is based on Linux 2.6.12 Operating System. The ISaGRAF V5 run time is embedded on this core. The BSP ISaGRAF RIOM, (Board Support Package), is a specific Linux distribution.

Linux kernel and file system is the main element: it is the only interface between the system and hardware: its essential functions are the task manager, memory management, and devices monitoring.

The libraries are the interface of applications launched automatically at startup.

After powering up the system, the first software running is U-boot : it performs the initialization of components on the CPU board (micro processor, clock, RAM and Flash component Ethernet ...), then performs the launch of Linux in RAM memory, and at the startup end of ISaGRAF virtual machine .

RIOM embedded software is naturally programmable with the workbench ISaGRAF V5.13 or higher.

Quick start

Overview

This chapter describes all the required operations to implement and test a basic program for the RIOM in less than 15 minutes.
We detail in this chapter the following steps:

- Installing ISaGRAF Workbench
- Creating a new project
- Target settings modification of an existing project
- I/O wiring
- Build, download and debug

Installing ISaGRAF V5

Installating of ISaGRAF V5 workbench:

Insert the CD Rom ISaGRAF V5 ICS Triplex in your Windows XP PC, then run the installation of the workbench. The recognition of your USB donggle can be checked via the tool "Licensing ISaGRAF 5" in the menu ICS Triplex.

Integration Leroy Automation RIOM ISaGRAF files to the workbench

Copy from the CD-ROM directory Leroy "RIOM_ISAGRAF_LNX_xx" in the directory "Template" of the CD in the directory ISaGRAF following : « C:\Documents and Settings\All Users\Documents\ICS Triplex ISaGRAF\Projects\ISaGRAF 5.2\Tpl »

Start ISaGRAF V5 workbench with the new link from start menu.

Creating a new project

Creating a new project :

Open the menu « File / New Project », then select PLC model « RIOM_ISaGRAF_LNX_xx », fill in the name of your new project, and submit.

Import the definition of the PLC :

This operation will update definitions of configuration RIOM.

Open the menu "File / Import / PLC Definition" and select the TDB file from directory /TDB on CD Rom

Existing project updating

Open your project :

Open the menu « File/Open Project » : select the file « PrjLibrary.mdb » contained in the directory of your project and submit.

Cancelling the current build of the project :

This will allow for the deletion of files compiled on the old target.
Click the menu « Project / Cancellation build project ».

Import definition of the PLC :

This will allow recovery of the definitions of configuration RIOM:
The file to retrieve is "RIOM_LNX-Jx.y.txt" or "RIOM_LNX-Jx.y.tdb" previously copied.
Open the menu "File / Import / PLC Definition" and select the file above.

Select Target RIOM in the properties of your ressource

Select your ressource, then click on the menu « Edit/Properties ». In tab « Target/Code », select in the dropdown « Target », « PNG_LNX ».

I/O wiring

In the button bar click the icon for wiring I/O: the result is the opening of the editor of wiring I/O, then click the button to add I/O boards:

Select in dropdown the UCRRthCanLs board:

You must select an CPU in slot 0.
The first IO Board must be in slot 1.

Close the I/O wiring editor.

Console configuration link : PC <-> RIOM

Double click the Setup icon in the project tree: it becomes the physical architecture of your project:
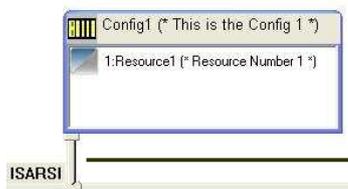
Change the « ISARSI » link with a double click, the window « Network – Properties » appears : modifiy then the com port number, and validate.
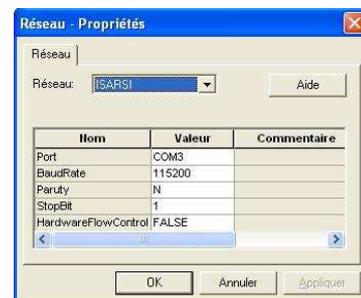
If you want to use Ethernet connection as a console, you have
to double-click on « ISARSI », and select the link "ETCP" in the dropdown ;
then set up communication with the network by double clicking on this link and set the RIOM IP address:

Build

In the menu bar, select the menu "Project / Build Project" or the button bar, click the icon to compile the project: the result is the compilation of your project, with the following message appears in the window messages:
« Name of your project: 0 error (s), 0 warning (s) ».

Download

Click the download button in the main toolbar: it appears the window "Download", select the config1, and click the "Download" button.
After downloading, the following message should appear in the message window: "The download is completed successfully"

Debug

Click on the "Debug" in the main toolbar: your project is in debug mode: all parts of the project can be viewed.
To return to edit mode for your project, click "Stop debug mode":

Fail safe mode

In case of trouble while trying to communicate with target, you can use fail safe mode.

- Switch power off,
- connect together pins 2 and 3 of maintenance serial link,
- Switch power on: led *mode* lites up,
- Isagraf starts but application is stopped; target is wating for worbench communication on serial link.

I/O wiring

Overview

This chapter describes the RIOM board configuration.
We detail in this chapter the settings of CPU and IO boards:

- I/O wiring
- CPU board
- Ethernet settings
- Digital I/O boards : DIOFER
- I/O Board Status

I/O wiring

Click on the menu « **Project** » / « **I/O wiring** » or on the corresponding button: the wiring editor appears.

3 boards maximum can be added.

Each board will be identified with an « Device Index».

Device Index n° 0 : reserved to the UCR named **"UCREthCanLs"**

Note : CPU communications ports are managed in ISaGRAF project with C functions.

Device Index n° 1 to 2: they are reserved to I/O boards named **"DIOFER3216"**. The device index corresponds to the physical position of these.
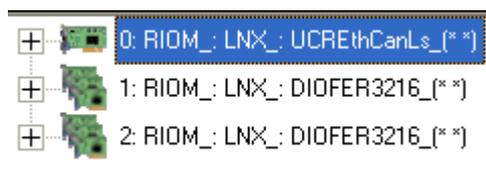
"UCREthCanLs" board

Board parameters are:

- **CardID** (Word, read only) : Internal identification code of the board. Read-only. Value = 610

- **SecuredStart** (BOOL) : Configuration checking before starting IO.
    - ➢ FALSE (default) : no checking.
    - ➢ TRUE : checking before startup : if the physical software configuration are not identical, IO default is raised.

NB : Checking is performed only on the boards reported in the workbench. If Supernumeraries boards are present, they will not be managed by ISaGRAF and the watchdog will not be triggered.

- **WDGTimems**   (WORD)   : not used.
- **FailOnWdg**   (BOOL)   : not used.
- **BusWdgOnWdg** (BOOL)   : not used.
- **StopOnWdg**   (BOOL)   : not used.
- **RebootOnWdg**   (BOOL)   : not used.

This board has a boolean output : rising this output will put all outputs in fallback mode.

Ethernet settings

The changes in the Ethernet network settings is done via the window properties of your resource, you can access via the menu "Edit / Properties" ; in the window, change the value in the "Extended" tab :
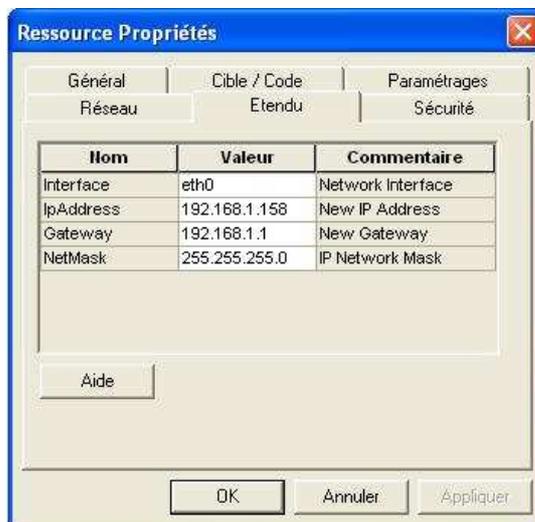
Ethernet settings are :

- Interface : name of Ethernet interface : « eth0 » by default.
- IpAddress : IP address of RIOM on an TCP/IP network. By default IP address is « NULL ». In this case, the LT ignores the other parameters and uses a BOOTP address server, which will send a free IP address to the LT. Format : xxx.xxx.xxx.xxx where xxx [0..255]
- Gateway : IP address of the gateway on the network. If the LT wishes to communicate outside the network to which it belongs, it must address this gateway. By default, this address is « NULL ». Format : xxx.xxx.xxx.xxx where xxx [0..255]
- Netmask : address mask used to show the breakdown of the IP address into sub-network address and device address on the sub-network. This 32-bit mask is composed entirely of 1's for all the sub-network address parts and entirely of 0's for the device address parts. Using the sub-network mask, the LT determines if it must contact the gateway to reach a recipient according to the IP address of the recipient and the sub-network. Format : xxx.xxx.xxx.xxx where xxx [0..255]

"DIOFER3216" board : 32 digital inputs/16 digital outputs

It is made up of 5 sub boards :
- VStatus_ : board status
- VdInput32_ : 32 digital inputs
- VdInputState32_ : 32 digital input states
- VdOutput16_ : 16 digital outputs
- VdOutputState16_: 16 digital output states

Inputs/ Outputs Board Status

Each I/O board declared in the I/O wiring has a status word : it consists of :

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X X X X				Adress (1 or 2)				Communication Pattern: 0xD0: DIOFER3216 on slot 1 0xB0: DIOFER3216 on slot 2							

I/O wiring

CPU Specific Functions

Time management

Writing the date

Function	**Date_Write()**
Action	Write the date in RIOM clock
Parameters	(String[10])DateToWrite : date to write format : "jj/mm/aaaa"
Returned Value	(DINT) Operation status : • 1 : operation successful, • other : operation failed, check the error code at the end of the document.

Reading the date

Function	**Date_Read()**
Action	Read the date in RIOM clock
Parameters	(String[10])DateToRead : date read format : "jj/mm/aaaa"
Returned Value	(DINT) Operation status : • 1 : operation successful, • other : operation failed, check the error code at the end of the document.

Writing the time

Function	**Time_Write()**
Action	Write the time in RIOM clock
Parameters	(String[12])TimeToWrite : time to write in the clock format : "hh:mm:ss:xxx" avec xxx in milliseconds
Returned Value	(DINT) Operation status : • 1 : operation successful, • other : operation failed, check the error code at the end of the document.

Reading the time

Function	**Time_Read()**
Action	Read the time
Parameters	(String[12])DateToRead : time read in the clock format : "hh:mm:ss:xxx" avec xxx in milliseconds

Returned Value	(DINT) Operation status :
	• 1 : operation successful,
	• other : operation failed, check the error code at the end of the document.

Data Storage

RIOM has flash memory available for the user project : the amount available is 2 Mo.

The user can store in this space files containing either word 16bits, or strings.

Type WordTab

WordTab	(WORD)[0..1023]

Reading words

Function	**FlashByte_Read()**
Action	Read words in flash memory
Parameters	• (String[255]) FileName : name of the file to access.
	• (WordTab) Data : Array to receive words read.
	• (UDINT) OffsetInWords : Offset of reading in words number.
	• (UINT) SizeInWords : number of words to read (Max 1024)
Returned Value	(DINT) Operation status :
	• >=0 : number of elements read, operation successful,
	• other : operation failed, check the error code at the end of the document.

Write words

Function	**FlashByte_Write()**
Action	Write words in flash memory NB : writing is always at end of file
Parameters	• (String[255]) FileName : name of the file to access (or create if does'nt exist).
	• (WordTab) Data : array of words to write.
	• (UDINT) OffsetInWords : Offset of writing in words number.
	• (UINT) SizeInWords : Number of words to write (Max 1024)
Returned Value	(DINT) Operation status :
	• >=0 :number of elements written, operation successful,
	• other : operation failed, check the error code at the end of the document.

Reading characters

Function	**FlashChar_Read()**
Action	Reading characters in flash memory
Parameters	• (String[255]) FileName : name of the file to access.
	• (String[255]) Data : Message to receive data.
	• (DINT) OffsetInChar : Offset of reading in characters.
	• (USINT) Size : Number of characters to read. Max 255
Returned Value	(DINT) Operation status :
	• >=0 :numbers of elements read, operation successful,
	• other : operation failed, check the error code at the end of the document.

Writing characters

Function	**FlashChar_Write()**
Action	Writing characters in flash memory NB : writing is always at end of file
Parameters	• (String[255]) FileName : name of the file to access (or create if does'nt exist). • (String[255]) Data : Message to write. • (USINT) Size :Number of characters to write. Max 255 • (BOOL) CRLF : Activate / desactivate newline option when writing.
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of the document.

Error codes

Error code	Meanings
-400	Length of parameter invalid
-410	Date reading error
-411	Date conversion error
-412	Date invalide
-413	Date writing error
-420	Time reading error
-421	Time conversion error
-422	Time invalide
-423	Time writing error
-500	Length of parameter invalid
-501	Error in generating the file path
-502	Error in opening the file
-503	Error in moving up to the offset
-504	Error in reading the file
-505	Error in writing the file
-506	Parameter Size too high
-507	Error closing file

Ethernet TCP and UDP functions

Overview

This package allows to send and receive bytes on TCP and UDP

Types of variables

A new structure, named « Socket », in ISaGRAF dictionary :

Socket	
ID (USINT)	Number of sockets : between 1 and 10 (TCP and UDP together)
Protocol (WORD)	0 : UDP ; 1 : TCP
ClientOrServer (WORD)	0 : Client ; 1 : Server
Port (WORD)	Port number
Address (STRING(20))	If TCP client, Address of remote server Not used otherwise.
MulticastGroup (STRING(20))	If UDP server, Multicast group to receive
Status (WORD)	If TCP client, Socket status updated automatically : 1: server connected; 0 server disconnected

Available Functions

Opening a socket

Function	**Socket_Open()**
Action	Opening a socket
Parameters	(socket) Socket : structure description of the socket to open
Returned value	(DINT) Function status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Closing a socket

Function	**Socket_Close()**
Action	Closing a socket
Parameters	(USINT) Id : ID of the socket to close
Returned value	(DINT) Function status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

<u>Data waiting to read</u>

Function	**Socket_Number()**
Action	Whether data is waiting for reading
Parameters	**ID** (USINT) : socket ID
Returned value	Function status (DINT) : • 1 : elements pending to read, • 0 : any elements in reception.

<u>Reading bytes</u>

Function	**Socket_ReadByte()**
Action	Reading data in the reception queue
Parameters	• **in** (USINT) **ID** : Socket ID • **out** (ByteTab) **Tab** : Reception table to store received bytes • **in** (WORD) **ByteNumber** : Number of bytes to read • **in** (WORD) **Offset** : Reception address in the reception table = Initial address + offset. • **out** (STRING(20)) **Source** : If UDP server, IP address of data transmitter.
Returned value	Function status (DINT): • 1 : operation successful • <1 : operation failed, check the error code at the end of this chapter

NB : the words « **in** » and « **out** » in bold to indicate the settings if they are inputs or outputs of the function Words.

Ethernet functions error codes

Error Code	Meaning
-700	ID of socket already open
-701	Bad socket number
-702	Socket ID already closed
-703	Connection error
-704	Offset error –or- wrong number of elements
-705	Bad protocol
-706	Port Number already used
-707	TCP address server not informed
-708	ID already used

Modbus RTU and TCP communication

Overview

This chapter describes the modbus RTU and TCP use on RIOM.
We detail in this chapter the following steps :

- Modbus protocol
- Modbus Slave protocol: RTU et TCP
- Modbus Master protocol: RTU et TCP
- Modbus failure codes

Modbus protocol

Modbus is a communication protocol that allow the exchange of data between several devices. It's a master / slave protocol. The hardware link on RIOM can be either a serial link (RS232, RS485, RS422) than an Ethernet link (100Mb).

This protocol is described in several downloadables documents : http://www.modbus.org/

RIOM can handle simultaneously the following features :

- on each of its 4 serial connections : master or slave modbus RTU
- on its Ethernet link : master or slave modbus/TCP

A data table may be associated with each slave.

Data are bit and word (16 bits) type, and slaves tables have their size in word : bit and word tables are the same.

Example of modbus table:

It can be represented as an array of 16 columns, representing the 16 bits in a word, and x lines representing the words:

Word adress	Bit rank	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Decimal value
0		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	=255
1		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	=4
2		0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	$=1*2^{12}+1*2^{6}=4160$
3									1									
4						0		1		1								
5																		

For the modbus address of a bit, you must do the following calculation:

Address Bit = 16 * word address + bit rank

Example : [Word address = 2 ; bit rank = C] ➔ Bit address = 2C hexa = 2*16+12=75 dec

The sum of all sizes tables slaves should not exceed 4096 words.

Maximum number manageable by RIOM on RTU and TCP:
- maximum number of modbus masters : 9.
- maximum number of modbus slaves : 16.

Functions modbus codes managed by the RIOM are:

Code	function	description
1	read_coils	reading bits
2	read_input_discretes	reading input bits
3	read_multiple_registers	reading words
4	read_input_registers	reading input words
5	write_coil	writing a bit
6	write_single_register	writing a word
15	force_multiple_coils	writing bits
16	write_multiple_registers	writing words
23	read_write_register	reading and writing words

Structures and functions

ModbusTable : Type of variable created to represent the Modbus table in RIOM :

Data Structure	**ModbusTable**
(WORD)	Table of modbus words [0..4095]

NB: All Modbus protocols work with the same table. This is the variable ModbusTable first transmitted to a Modbus Function to be taken into account.

All addresses have an offset at 0 by default.

Example of declaration in ISaGRAF dictionary:

Modbus_Init function:

Function	**Modbus_Init()**
Action	Initialisation of Modbus service
Parameters	Table (ModbusTable): Table Modbus.
Returned Value	(DINT) Status of operation : • 1: operation successful, • autre: operation failed, check the error code at the end of this chapter.

Example	`(* initialisation modbus service *)`
	`IF Status_i_modb = 0 THEN`
	`        Status_i_modb := Modbus_Init(ModTab);`
	`END_IF;`

SerialParam : data structure within the modbus RTU, used in initialization serial ports functions, to specify the parameters setting of the port.

Example : declaration in ISaGRAF dictionary

Data Structure	**SerialParam**
Speed (UINT)	Port Speed (unity : bauds) : 1200, 2400, 4800, 9600, 19200, 38400
Parity (USINT)	Parity : 0 (none), 1 (odd), 2 (even)
StopBit (USINT)	Number of stops bits : 1 or 2
DataSize (USINT)	Number of data bits : 7 ou 8
Mode (USINT)	0 (RS232), 1 (RS485), 2 (RS422)
| Example | ```
IF Status_i_modb = 0 THEN
      Status_i_modb := Modbus_Init(ModTab);
(* initialization modbus comA parameters *)
      Params_com0.Speed := 9600;
      Params_com0.Parity := 2;
      Params_com0.StopBit := 1;
      Params_com0.DataSize := 8;
      Params_com0.mode := 1;(* RS485*)
END_IF;
``` |

Modbus slave protocol

Modbus service must first have been initialized.

Each open slave is working on a segment : a segment is a modbus subtable, part of the general table and being positioned within it with an offset and length.

Opening a modbus RTU slave port :

Function	**Modbus_OpenSerialSlave**
Action	Opening a modbus RTU slave.
Parameters	• SlaveNumber (USINT): slave address on the network (1 à 255). • ComName (STRING) : « /dev/tts/0 » for SL1, « /dev/tts/2» for SL2, (/dev/tts/1 is used for workbench communication) • Params (SerialParam) : port parameters structure • GeneralOffset (UINT) : offset in words to open the segment compared to the general table. • Size (UINT) : number of words of the segment to open • *TimeOut : not used*
Returned Value	(DINT) slave handle: • >0 : slave handle open • < 0 : operation failed, check the error code at the end of this chapter.
| Example | ```
(* initialization modbus RTU slave com A *)
IF Status_i_modb = 1 AND Status_o_modbs_comA_slave = 0  THEN
      Status_o_modbs_comA_slave := Modbus_OpenSerialSlave(1,
'/dev/tts/0',Params_comA,0,100,0,0);
END_IF;
``` |

NB1 : slave number corresponds to the address of the slave on the network.
NB2 : data bit number must be 8. Any other value will result in an error at the opening.

Opening a modbus/TCP slave port :

Function	**Modbus_OpenTCPSlave**
Action	Opening a modbus TCP slave.
Parameters	• SlaveNumber (USINT) : slave number (1 to 255). • Port (UINT) : local TCP slave port (502 by default). • GeneralOffset (UINT) : offset in words of the segment to open compared to the general table. • Size (UINT) : size in words of the segment to open. • TimeOut (UINT) : time out in milliseconds.
Returned Value	(DINT) slave handle : • >0 slave handle open • < 0 : operation failed, check the error code at the end of this chapter.
| Example | ```
(* initialization modbus TCP slave *)
IF Status_i_modb = 1 AND Status_o_modbTCP_slave = 0  THEN
     Status_o_modbTCP_slave := Modbus_OpenTCPSlave( 10, 502,
0, 1000, 100);
END_IF;
``` |

Master supervision

Function	**Modbus_AddSupervision**
Action	Monitor the activity of a master.

Parameters	• SlaveID (DINT) : slave handle, returned by the Opening Function
	• MasterName (STRING) : IP address of the master to monitor (ignored for RTU communication)
	• TimeOut (UDINT) : TimeOut in milliseconds (minimum 1000)
	• PresenceState (BOOL) : Dictionary variable that will automatically receive the status of the monitored master presence.
	  ➢ False : master absent,
	  ➢ True : master present.

Returned Value	(DINT) Function status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.
| Example | ```
(* master supervision *)
IF Status_addsup1 = 0 THEN
     Status_addsup1:= Modbus_AddSupervision(
Status_o_modbTCP_slave, '192.168.1.10', 5000, Supervis1);
END_IF;
``` |

Master filtering

Function	**Modbus_CreateTCPFilter**
Action	The slave will respond to requests only for masters identified with this function. NB : Function as this is not called, the slave accepts the requests of all masters. The number of masters is limited to 8
Parameters	SlaveID (DINT) : slave handle, returned by the Opening Function MasterName (STRING) : IP address of the master autorized
Returned Value	(DINT) Operation status : 1 : operation successful, other : operation failed, check the error code at the end of this chapter.

Example	```
	(* master filter *)
	IF Status_filtsup1 = 0 THEN
	    Status_filtsup1:= Modbus_CreateTCPFilter(
	Status_o_modbTCP_slave, '192.168.1.10');
	END_IF;
	``

Closing a modbus slave port

Function	**Modbus_CloseSlave**
Action	Close a modbus slave port.
Parameters	SlaveID (DINT) : slave handle, returned by the Opening Function
Returned Value	(DINT) Function Status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Example	```
	(* Close modbus/TCP slave connection *)
	IF dem_ferm AND Status_ferm_esclave = 0 THEN
	    Status_ferm_esclave :=
	Modbus_CloseSlave(Status_o_modbTCP_slave);
	END_IF;
	``

Modbus master protocol

A modbus RTU master works with one COM port only. If there are several slaves (network 485 or 422), it is the slave number (in the structure Modbus Request) that differentiates.
A modbus/TCP master can have only a single TCP slave (single IP address provided to initialize the master).
The total number of masters is 10 maximum.

Opening a master modbus RTU connection :

Function	**Modbus_OpenSerialMaster**
Action	Opening a modbus RTU master
Parameters	• MasterNumber (USINT): master number (0 to 50). • ComName (STRING): name of the com to use: « /dev/tts/0 » for COM0, « /dev/tts/1 » for COM A, « /dev/tts/2 » for COM B, « /dev/tts/3 » for COM C • Params (SerialParam): port parameters • TimeOut (UINT): timeout in milliseconds.
Returned Value	(DINT) operation status : • 1: operation successful, • other : operation failed, check the error code at the end of this chapter.

Example	```(* initialization modbus master comA*)```
	```IF Status_o_modbs_comA_master = 0 THEN```
	``` Status_o_modbs_comA_master :=```
	```Modbus_OpenSerialMaster(1,'/dev/tts/1',Params_comA,100);```
	``` END_IF;```

NB1 : master number is a unique identifier, used to obtain the diagnosis of the communication. The number of master must be different for each new opening.

NB2 : data bit number must be 8. Any other value will result in an error at the opening.

Opening a master modbus/TCP connection :

Function	**Modbus_OpenTCPMaster**
Action	Opening a modbus/TCP master.
Parameters	• MasterNumber (USINT): master number (0 à 50). • TargetAddress (STRING): IP address of the slave Format : « 192.168.1.156 » • Port (UINT): number TCP port of the slave (standard port is 502). • TimeOut (UINT): timeout in milliseconds
Returned Value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Adding a request :

The number of request per master is limited to 16.

There are 3 types of Modbus requests:
- Periodic request : the request is constant emission period,
- Trigger request : the request is issued by event (booleans triggers)
- OneShot request : the request is issued only once

NB: Choice by « Period » field in the "ModbusRequest" structure below.

Function	**Modbus_AddRequest**
Action	Store a request for a modbus master request.
Parameters	• MasterNumber (USINT): master number (0 to 50). • Request (ModbusRequest): modbus request to add
Returned Value	(DINT) operation status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Data structure **"ModbusRequest"**	
Period (UINT)	Sending period in milliseconds : • 0: request One Shot • 65535: Request on trigger • other: period of the periodic request
Trigger (BOOL)	If trigger request, the request starts when the value change from FALSE to TRUE.
SlaveNumber (USINT)	slave number (address): If RTU [0 … 255], 0 = broadcast If TCP [1]
FunctionCode (USINT)	Function Modbus Code: • 1 ou 2: read n bits • 3 ou 4: read n words • 6: write 1 words • 15: write n bits • 16: write n words • 17: write n words, read m words
ReadSlaveAddress (WORD)	Read Offset address in the modbus slave table.
ReadMasterOffset (WORD)	Read Offset address in the local RIOM table for the data read in the slave.
ReadLength (WORD)	Number of elements to read : • If Function code = 1 or 2 , number = [1.. 2000] bits • If Function code = 3,4,17, number = [1.. 125] words
WriteSlaveAddress (WORD)	Writing Offset address in Modbus slave table.
WriteMasterOffset (WORD)	Offset address in the local RIOM table for the data that have to be write in the slave
WriteLength (WORD)	Number of elements to write : • If Function code = 15, number = [1.. 800] bits • If Function code = 16,17, number = [1.. 100] words
Status (DINT)	Request status : see table ci-dessous
NbFrameOK (UDINT)	Number of successful frames
NbError (UDINT)	Number of failed frames

offsets units are in :
- bits for read and write bits functions
- words for read and write words functions

modbus request status

The status of modbus request is encoded on 32 bits (type DINT) :
- the error code is coded on 16 bits of high part,
- the error code is coded on 16 bits of low part.

Error value	Meaning
0	No error
64 (0x40)	internal error
65 (0x41)	internal error
66 (0x42)	opening port error
67 (0x43)	Serial Port already open
68 (0x44)	TCP connection error
69 (0x45)	Connection closed by slave
70 (0x46)	internal error
71 (0x47)	internal error
72 (0x48)	internal error
73 (0x49)	internal error
74 (0x4A)	Access to port impossible
75 (0x4B)	Port TCP not available
128 (0x80)	internal error
129 (0x81)	Checksum error
130 (0x82)	Frame error
131 (0x83)	invalid response
132 (0x84)	Response Time out
133 (0x85)	Sending Time out
161 (0xA1)	Exception Illegal Function Response
162 (0xA2)	Exception Illegal Address Response
163 (0xA3)	Exception Illegal Data Value Response
164 (0xA4)	Exception Slave Device Failure Response

communication shutdown :

Function	**Modbus_CloseMaster**
Action	Close a master and delete the associated requests.
Parameters	MasterNumber (USINT): number of master [0 .. 50].
Returned Value	(DINT) operation status: • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

modbus error codes

Error codes	Meaning
-100	com number incorrect
-101	Opening com impossible
-102	Speed incorrect
-103	Parity incorrect
-104	Number of stop bits incorrect
-105	Number of data bits incorrect
-106	COM configuration impossible
-107	Configuration RS232/485/422 impossible
-108	Port initialization impossible
-109	port specified doesn't exist
-110	Wrong mode
-111	port specified closed
-112	offset error or size of elements incorrect
-113	No element or access port error
-114	This port is already token for the console link

Error codes	Meaning
200	Service already initialized
-200	service initialization impossible
-201	Master number incorrect.
-202	Modbus service not initialized.
201	Master numberalready open.
-204	Master opening error.
-205	Adding request Impossible.
-206	Master number not open.
-207	Master closing impossible.
-208	Slave number incorrect
-214	Slave opening impossible
-216	Slave not open
-217	Slave can't be closed
-218	Impossible to add the segment
-219	offset error or size
-220	maximal number of request exceeded for this master
-221	mode incorrect
-222	maximal number of instances exceeded
-223	Impossible to add this filter
-224	Impossible to add the supervision
-225	Number of supervision exceeded for this slave
-226	invalid function code

Chapter

7

CAN Bus communication

Overview

This chapter describes the Can Bus 2.0A use on RIOM.

Types of variables

Message CAN

This structure describes a CAN message.
NB: This variable is not directly usable. You must use the variable **CanMsgTab** which is an array of 2048 variables *CanMsg*.

CanMsg		
Period	UINT	Period in ms : • of production for a produced message, • of monitoring for a consumed message, • 0 if aperiodic or not monitored.
ConsOrProd	USINT	• 0 : Consumed message, • 1: Produced message.
TrigInhib	BOOL	• for an aperiodic produced message : sending trigger. • for a periodic message : inhibition of sending.
Data	BYTE[0..7]	Message data
Length	USINT	message length : [1 .. 8] bytes
Status	USINT	message status: • 0 : status OK, • 1 : timeout (the message consumed was not received by the controller for at least Period) • 2 : Never received (the consumed message is never arrived to the controller) • 3 : message writing error.

Status CAN

CanStatus		
DriverMode	USINT	Mode of operation of driver : 1 : Initializing, 2 : Operational , 3 : Stopping.
ControlerMode	USINT	Mode of operation of the CAN Controller : 0 : initialized, 1 : Operational
CanState	USINT	CAN network state : 0: Error Active, 1: Error Passive,

		2: Bus Off
Cycle	UINT	current cycle time of the controller (in ms).
MinCycle	UINT	least cycle time of the controller
MaxCycle	UINT	Maximum cycle time of the controller
Qemit	UINT	Number of frames in transmission queue
QemitMin	UINT	Least number of frames in transmission queue
QemitMax	UINT	Maximum number of frames in transmission queue.
QnoEmit	UINT	Number of frames lost in transmission.
Qrecep	UINT	Number of frames in reception queue
QrecepMin	UINt	Least number of frames in reception queue
QrecpMax	UINT	Maximum number of frames in reception queue.
QnoRecep	UINT	Number of frames lost in reception.

Functions

Initialization

Function	**CAN_Init()**
Action	Service initialization.
Parameters	None
Returned value	(DINT) operation status : • 1 : operation successful, • <1 : operation failed, check the error code at the end of this chapter.

Startup

NB: it must have filled out the table CANMsgTab before to use the CAN_Start() function. After this call, no further configuration changes are possible.

Function	**CAN_Start()**
Action	Start the CAN service
Parameters	• (CANMsgTab) Tab : CAN messages array. • (CANStatus) Status : CAN status, automaticaly refreshed thereafter. • (UINT) WatchPeriod : period in ms of monitoring (0 if any si no monitoring) of application by the CAN controller. If the application doesn't refresh its variables with the controller in the specified WatchPeriod, it returns in the state initialized.
Returned value	(DINT) operation status : • 1 : operation successful, • <1 : operation failed, check the error code at the end of this chapter.

Stop

Function	**CAN_Stop()**
Action	CAN service stop
Parameters	None
Returned value	(DINT) operation status : • 1 : operation successful, • <1 : operation failed, check the error code at the end of this chapter.

CAN error codes

Error codes	Meaning
-900	Opening driver error
-901	CAN already used
-902	Error when message insertion
-903	Memory allocation error
-904	CAN already started
-905	CAN not initialized
-906	CAN startup error
-907	CAN not started
-908	Error when CAN stoped.
-909	Startup error of low level CAN