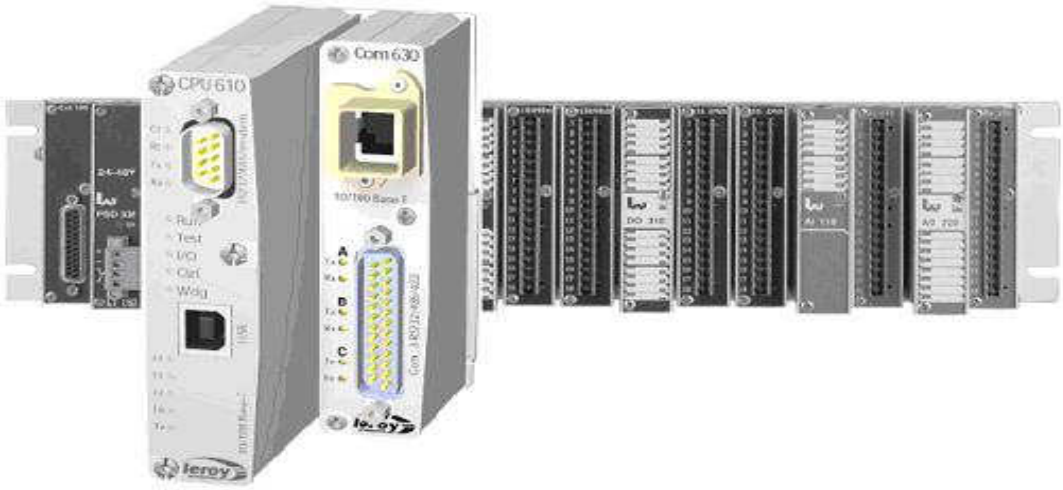
LT200



User's manual
ISaGRAF V5 programming

P DOC LT200 001 E - V04

Leroy automation
Boulevard du Libre échange
31650 Saint Orens / Toulouse France
Tel : +33 (0) 5 62 24 05 50 Fax : +33 (0) 5 62 24 05 55
e-mail : info@leroy-autom.com
web site : www.leroy-automation.com

Overview

LT200 is part of the LT family, PLC programmable in two languages :

- LT200 in language C under Linux, with Eclipse IDE.
- LT200 in languages IEC 61131-3 ISaGRAF 5 workbench from ICS Triplex

The LT200 hardware implementation is explained in the wiring manual available on our website.

Prerequisite

Developpement of applications on LT200 ISaGRAF require the knowledge of programming in IEC61131-3 languages.

The actual implementation of the LT200 requires skills in electricity and industrial automation.

The equipment required is:

- A development PC running Windows XP, with an USB port and an Ethernet network card.
- An USB cable
- An Ethernet cable

Version

This documentation describes the features in the LT200 ISaGRAF version 1.0

Property

Contact

✉ Leroy Automation
35 Boulevard du Libre Echange
31650 SAINT-ORENS
France
☎ 33 (0) 5.62.24.05.50
33 (0) 5.62.24.05.55
http://www.leroy-automation.com
support technique :
☎ 33 (0) 5.62.24.05.46
mailto:support@leroy-autom.com

Contents

Chapter

1

General Overview

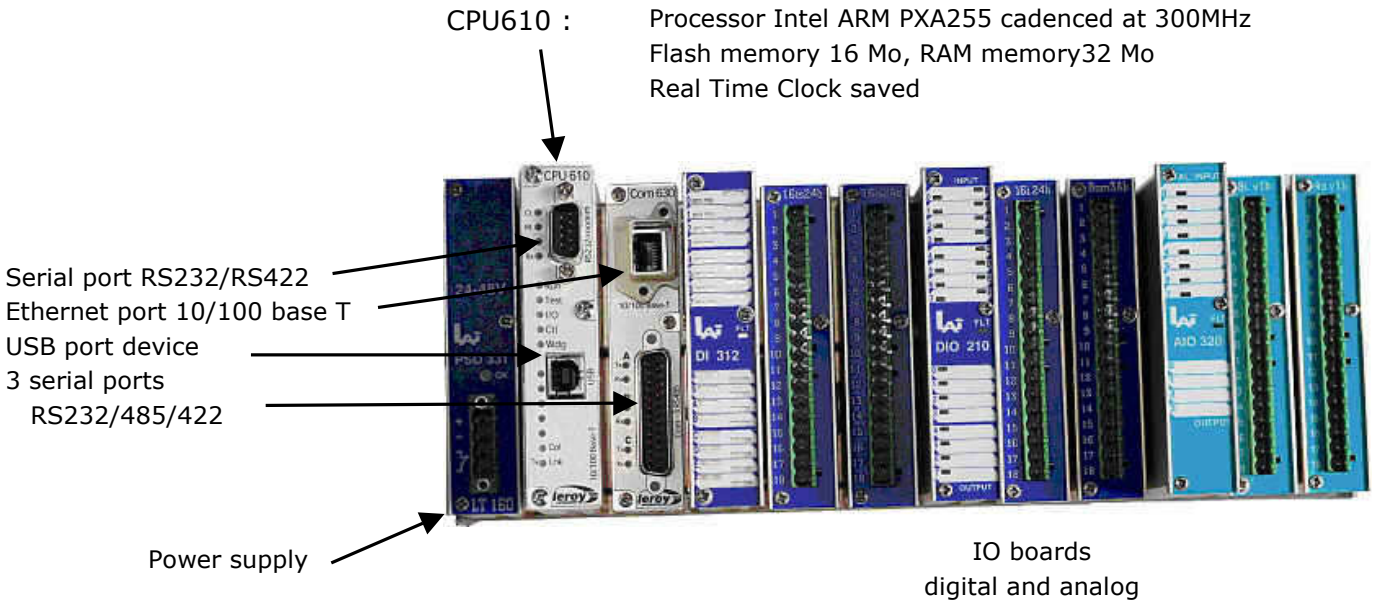
Introduction

This chapter describes the basics of LT200. We detail in this chapter :

- LT200 hardware,
- LT200 software

LT200 hardware

Application in LT200, will run and use all of its components hardware base:



LT200 embedded software

LT200 is based on Linux 2.6.12 Operating System : the ISaGRAF V5 run time was focused on this core. The BSP ISaGRAF LT200, (Board Support Package), is a specific Linux distribution.

The 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 .

LT200 embedded software is naturally programmable with the workbench ISaGRAF V5.13 or higher.

Chapter

2

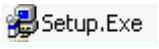
Quick start

Overview

This chapter describes all the operations necessary to implement and test a basic program for the LT200 in less than 15 minutes.
We detail in this chapter the following steps:

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

Installing the USB driver



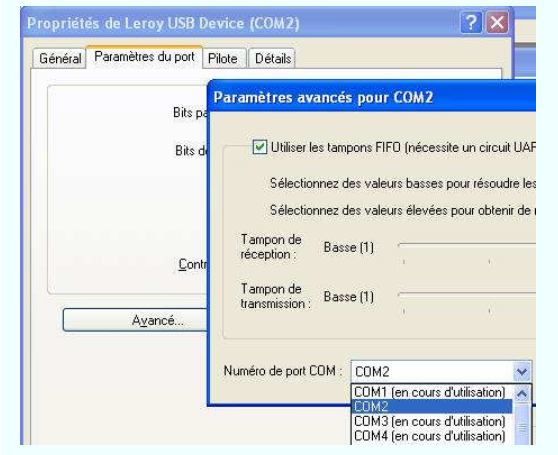
This driver enables the workshop to communicate with the LT200 USB: this driver has been developed exclusively for Windows XP.

The file to run a setup file is shipped on CD ROM in the folder "Driver-PC" file "Setup.exe"

Installation procedure:

- IMPORTANT: The LT200 should not be connected with USB to the PC, during installation of driver
- Run the file « Setup.exe »
- Connect the USB port of your PC to the LT200: a "beep" must be heard
- Identify the USB port com for LT200 in your PC: open the « configuration panel», then run the application « System », tab « Hardware », « device manager », « Ports (Com and LPT) » : note the number x of com : « Leroy USB Device (COMx)»

Note : the number of USB com port for the LT200 is assigned automatically by Windows ; ISaGRAF workbench requires to have a number strictly inferior to COM10: if it is superior or equal to COM10, you have to release a port of the first 9, and in the advanced settings of the port « Leroy USB device », change your com port number and select the one that is not used, inferior to COM10 :



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 LT200 ISaGRAF files to the workbench

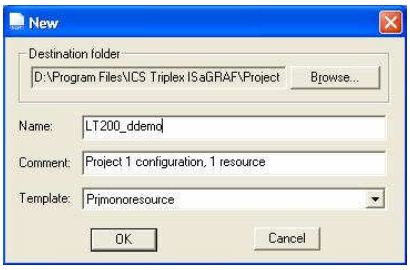
Copy from the CD Rom Leroy Automation the file « LT200_LNX-Jx.y.tdb » , in the directory « TDB » from CD, in the directory : « C:\Program Files\ICS Triplex ISaGRAF\Projects\ISaGRAF 5.x\Prj».

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

Creating a new project

Creating a new project :

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



Import the definition of the PLC :

This operation will allow recovery of the definitions of configuration LT200: the file to retrieve the text file is " LT200_LNX- Jx.y.tdb " previously copied.

Open the menu "File / Import / PLC Definition" and select the file above, in the directory before the new project: « \Prj »

Modification of target parameters of an existing project

Open your project :

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

Import the definition of the PLC :

This will allow recovery of the definitions of LT200 configuration:
The file to retrieve is the text file "LT200_LNX-Jx.y.tdb" previously copied.
Open the menu "File / Import / PLC Definition" and select the file above, in the directory before the new project: "\ Prj"

Select Target LT200 in the properties of your project

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

Cancelling the current build of the project :

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

Import a new time the definition of the PLC :

This operation is necessary for a good build : retrieve a new time the tdb file or "LT200_LNX-Jx.y.tdb" previously imported : open the menu "File / Import / PLC Definition" and select the file above, in the directory before the new project: "\ Prj".

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 CPU 6xx board:

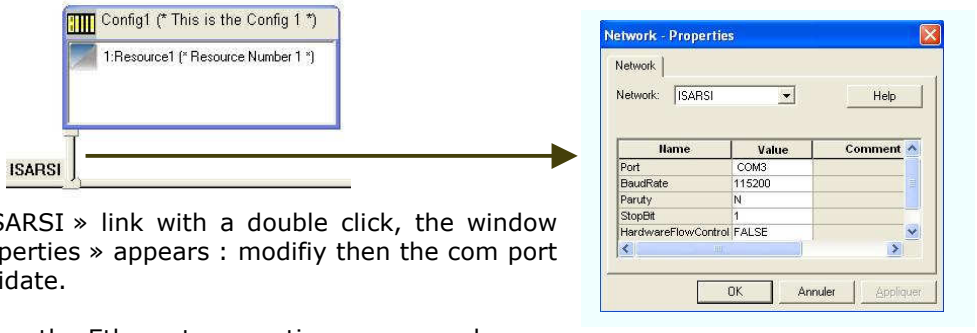
Close the I/O wiring editor.

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) ».

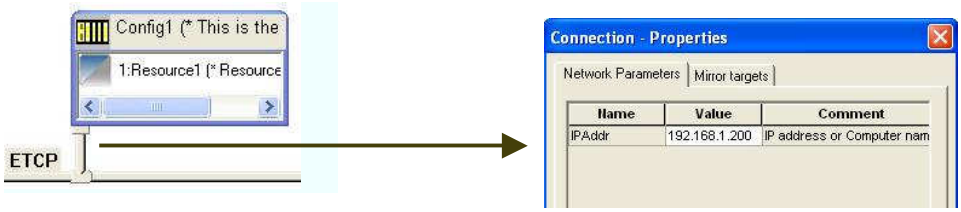
console configuration link : PC <-> LT200

Double click the Setup icon in the project tree: it becomes the physical architecture of your project:



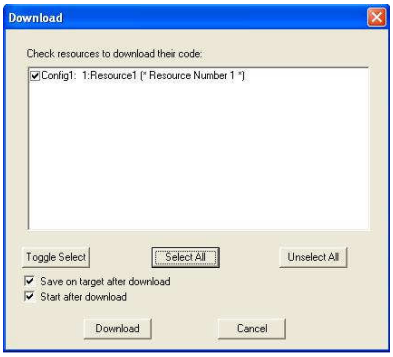
Change the « ISARSI » link with a double click, the window « Network – Properties » appears : modifiy then the com port number, and validate.

If you want to use the 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 LT200 IP address:



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

Double 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":

Chapter

3

I/O wiring

Overview

This chapter describes the board configuration of LT200.
We detail in this chapter the settings of CPU and IO boards:

- I/O wiring
- CPU 610 board
- Ethernet settings
- Digital I/O boards : DI310, DI312, DI410, DO310, DIO210
- Analog I/O boards : AI110, AI210, AO121, AIO320
- I/O Board Status

I/O wiring

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

16 boards maximum can be added.

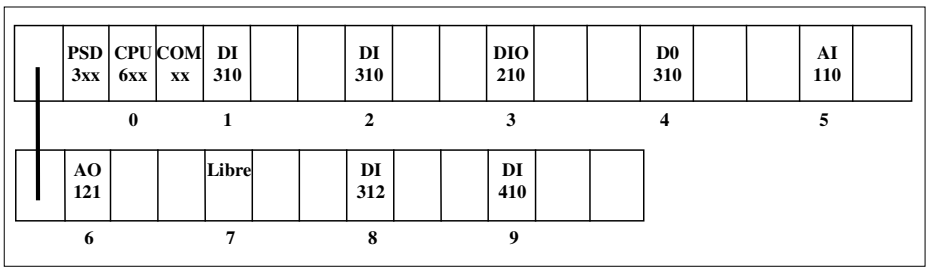
Each board will be identified with an « Device Index».

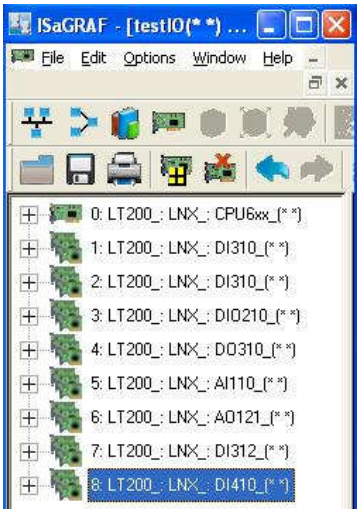
Device Index n° 0 : reserved to the CPU 610.

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

Device Index n° 1 à 15 : they are reserved to I/O boards. The device index corresponds to the physical position of these boards on the main rack and extensions racks (2 maximum).

Example : hardware configuration on 2 racks, and associated software configuration:





CPU610 board

Board parameters are:

- **CardID** (Word, readonly) : Internal identification code of the board. Read-only. Value = 610
- **SecuredStart** (BOOL) : Verification of the configuration before starting IO.
  - ➢FALSE (default) : no verification.
  - ➢TRUE : verification before startup : if the physical configuration and the software configuration do not, LT200 stay in general watchdog.

NB : The verification 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) : watchdog value in ms ; minimal value : 100ms.
  - ➢0 (default): no watchdog.
  - ➢>0 : Time maximum allocated for the cycle

  Advice : For safety, set this value to twice the average cycle time

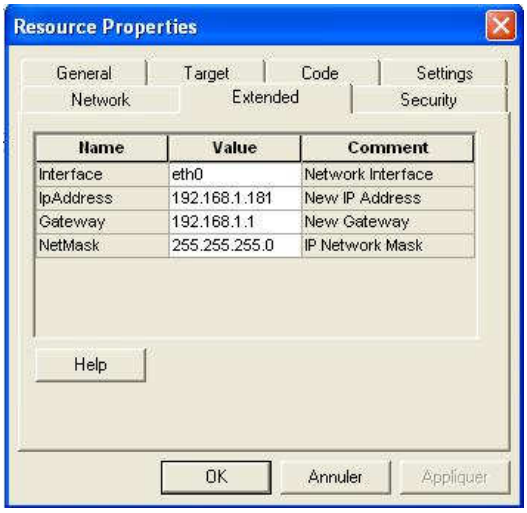
- **FailOnWdg** (BOOL)
  - ➢TRUE : FAIL led lights on exceeding the time set in WDGTimems
  - ➢FALSE : no action
- **BusWdgOnWdg** (BOOL)
  - ➢TRUE : Enabling watchdog IO bus on exceeding the time set in WDGTimems
  - ➢FALSE : no action
- **StopOnWdg** (BOOL) :
  - ➢TRUE : ISaGRAF will change to step by step mode on exceeding the time set in WDGTimems
  - ➢FALSE : no action
- **RebootOnWdg** (BOOL) :
  - ➢TRUE : LT200 will reboot on exceeding the time set in WDGTimems
  - ➢FALSE : no action

This board has a boolean output : Wdg : it drives the relay that is on the power supply board PSD3xx.

Wdg Once triggered, it is required to reboot the LT200 hardware (power cut and power up).

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 LT200 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 mask according to the following algorithm:
  Format : xxx.xxx.xxx.xxx where xxx [0..255]

DI310 board : 32 digital inputs

It is made up of 3 sub boards :

- VStatus_: board status
- V32ETOR_ : 32 digital inputs

DI410 board : 64 digital inputs

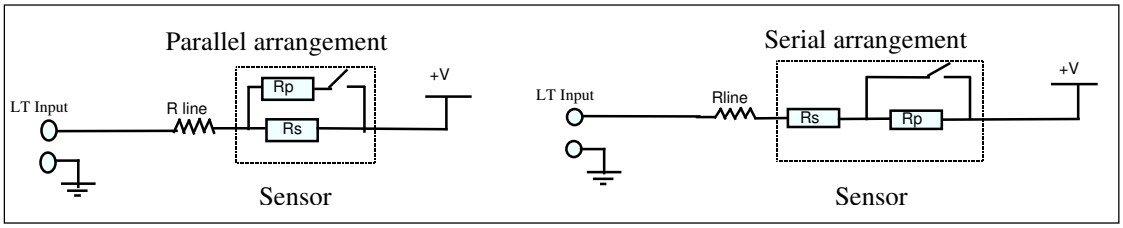
It is made up of 3 sub boards :

- VStatus_: board status
- V64ETOR_ : 64 digital inputs

DI312 board : 32 safety inputs

Principle :

DI312 modules are equipped with an adjustable comparison device used to **check the wiring** of sensors by connecting a network of 2 resistors to them: **safety inputs**. These resistor networks are of 2 types: the **serial arrangement** (i.e. the 2 serial resistors) and the **parallel arrangement** (i.e. the 2 parallel resistors). The serial resistor is always present. In the parallel arrangement, the sensor is mounted in series with Rp which it eliminates by opening. In the serial arrangement, the sensor is mounted in parallel with Rp which it eliminates by closing.



Wiring of Safety Inputs

In order to preserve the general nature of parameter setting, ISaGRAF can indicate the equivalent resistance of the resistor network when the sensor is **normally open (Rcno)** and when the sensor is **normally closed (Rcnf**). Resistance values are given **in OHMS**.

Parallel Arrangement	Serial Arrangement
Rcnf = Rs//Rp + Rline	Rcnf = Rs + Rline
Rcno = Rs + Rline	Rcno = Rs+Rp + Rline

CAUTION: Parameter setting is unique for the resistance values of a **DI312 board** and is therefore the same for all the channels of a single DI312 module.

DI312 board parameters are as follows:

- 32-bit mask for the wiring check of the 32 inputs. The wiring check is active at input n if the bit of order n is set to 1. By default, the 32 bits of the mask are set to 1.
- RCNO: only one value for all inputs.
- RCNF: only one value for all inputs.
- Rline: only one value for all inputs.

For each channel, the status bit and alarm bit encode 4 possible states:

Input Status (Green LED)	Alarm (Red LED)	Description
0 (OFF)	0 (OFF)	Sensor normally open

1 (ON)	0 (OFF)	Sensor normally closed
0 (OFF)	1 (ON)	Input not connected or short-circuit at 0V
1 (ON)	1 (ON)	Short-circuit at +V

Resistance constraints:

- Resistors must have a tolerance of no more than 1%.
- 0.7 kΩ < **Rcno** < 22 kΩ
- Rcno induces the current in the measuring device: I (mA) = 22 (V)/(1+ Rcno (kΩ)) given that I must be between 1 mA and 9.96 mA . If the calculated value of I is more than 9.96 mA, saturate it at 9.96 mA.
- (2.95(V)/I (mA)) < **Rcnf** (kΩ) < Rcno (kΩ) - Rline (kΩ) - 1.95 (V)/I (mA)
- **Rline** < 0.2 kΩ.ρ

The DI312 module is made up of 2 boards:

- V32ETOR_ : 32 Boolean inputs
- VState_ : 32 faults relating to inputs.

DO310 board : 32 digital outputs

It is made up of 3 boards :

- VStatus_: board status
- V32STOR_ : 32 digital outputs

DIO210 board : 16 digital inputs and 8 digital outputs

It is made up of 3 boards :

- VStatus_: board status
- V16ETOR_ : 16 digital inputs
- V8STOR_ : 8 digital outputs

AI110 board : 8 analog inputs

It is made up of 2 boards :

- VStatus_: board status
- V8EANA_ : 8 analog inputs

conversion table : current/voltage <=> number of points	
Current Inputs	±21,1mA => ±32767 points.
Voltage Inputs	±10,25V => ±32767 points.

AI210 board : 16 analog inputs

It is made up of 2 boards :

- VStatus_: board status
- V16EANA_ : 16 analog inputs

the conversion table is the same than for the AI110 board.

AO121 board : 8 analog outputs

It is made up of 2 boards :

- VStatus_: board status
- V8SANA_ : 8 analog outputs

conversion table : number of points <=> current/voltage	
Currents Outputs	0 / 32767 points => 4 / 20mA
Voltage Outputs	±32767 points => ±10V

AIO320 board : 8 analog inputs and 4 analog outputs

It is made up of 3 boards :

- VStatus_: board status
- V8EANA_ : 8 analog inputs
- V4SANA_ : 4 analog outputs

Current-to-Voltage Conversion Table <=> Number of Points on AIO320	
current inputs	±20mA => ±32767 points.
voltage inputs	±10V => ±32767 points.
Current outputs	0/32767 points => 4/20mA
Voltage outputs	±32767 points => ±10V
PT100 sensors inputs	-50°C=>-500 points +350°C=>+3500 points

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
position of board on the bus [0..15]				Fault	Wdg	pow er	VCC	Board Code identification							

Input/output board codes:

Board	**Status Bit 11**	**Status Bit 9**	**Board Code [0..FFh] Mask Bit 6: BFh**
DI310	1	Al Ext	03h or 43h
DI312	NS	Al Ext	14h
DI410	1	Al Ext	06h
DO310	Fault	Al Ext	05h or 45h
DIO210	Monostable	VRel	16h
AI110	0	0	80h
AI210	0	0	81h
AO121	0	0	88h
AIO320	Monostable	0	83h

Where:

- NS: not significant
- Al Ext: set to 1 if the external power supply at the terminal blocks is in the Valim ±20% range.
- (*) Al ext of DI312: External power supply equal to 24V±10%
- Fault: set to 0 in the event of overload on a digital output channel
- Monostable: set to 1 if board is correctly refreshed; for 4TCD/TSD operation is reversed.
- Vrel: set to 1 if relays are correctly powered.

Chapter

4

CPU Specific Functions

Overview

This chapter describes the specific functions of LT200 CPU.
We detail in this chapter the management of :

- Retain variables
- CPU Time
- Data storage in CPU flash memory
- PID regulation function block
- I/O boards Leds

Retain variables

To make a variable as retain, you just have to declared it as « retain » in the ISaGRAF5 dictionary. The maximum total size of the variables is 3K.

The retain variables are stored in the following three cases :

- On ISaGRAF stop/restart : retain variables are not saved at shutdown and restored in re-starting.
- On download: retain variables are saved at the time of ISaGRAF stop. They will be returned to re-start if the "retain" has not changed in the project.
- On power failure: variables are stored at the break and restored when re-booting.

At startup, if the retain variables are corrupted or unavailable (case of the first startup of the project), the Fail LED lights, but the backup service is active. If there is no major problem, the Fail LED turns off after the second startup.

In case of problems, diagnostic information is contained in the log file "isasys.txt": see Chapter 8, "Diagnostics and Troubleshooting" section "Reading File event in the LT200.

Time management

The LT200 is equipped with a Real Time Clock (RTC). This clock gives the date, time and day of the week. Theses data can be read or written using following C functions in your ISaGRAF project.

Writing the date

Function	**Date_Write()**
Action	Write the date in LT200 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 LT200 clock
Parameters	(String[10])DateToWrite : 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 LT200 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])TimeToRead : time read in the clock format : "hh:mm:ss:xxx" with 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

LT200 has flash memory available for the user project : the amount available is 1 Mo.
The user can store in this space files containing either word 16bits, or strings.
The user has to manage this space memory.

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.

File removal

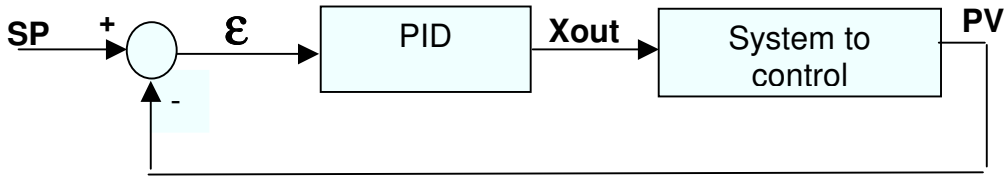
Function	**FlashDelete()**
Action	User File deletion
Parameters	(String[255]) FileName : name of file to delete.
Returned Value	DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of the document.

Reading customer files : ftp connexion

LT200 ISaGRAF has an FTP server : with a client software installed on your PC, you'll be able to connect you to LT200 FTP server, and to download all files created with ISaGRAF functions.
FTP connection parameters must be from type : « anonymous »

PID controller function block

A PID is a generic control loop feedback mechanism. The system output is controlled using the difference between the actual output of the system and the state it should have. This principle is summarized by the block diagram below.



SP = Set Point = Order, PV = Process Value

$Xout(t)=Kp(\varepsilon(t)+\frac{1}{Ti}\int_0^t \varepsilon(t)d(t)+Td\frac{d\ \varepsilon\ (t)}{dt})$ in continuous then

$Xout(k)=Kp(\varepsilon(t)+\frac{Ts}{Ti}I(k)+\frac{Td}{Ts}(\varepsilon(k)-\varepsilon(k-1)))$ in discret with $I(k)=I(k-1)+\varepsilon(k)Ts$.

Implementation

The PID calculation involves three separate actions: the proportional, the integral, the derivative. Each action can be adjust in order to control the system.
The PID implementation La mise en oeuvre d'un PID est réalisée par l'utilisation d'un bloc Functionnel C Pid_AL() :

- Declare a PID instance: in ISaGRAF dictionary / tab « Global variables», add a block function instance of **PID_AL** type ; for example "pid1",
- At each PLC cycle, call "pid1" with its parameters ; for example : pid1(Auto1, Pv1 ,Sp1, X01, Kp1, Ti1, Td1, Ts1, Min1, Max1);
- Use the instance block value : output1 is a numeric variable ; the returned value is : output1 := pid1.Xout;

Function Block	**Instance_PidAl()**
Action	PID controller
Parameters	1. Auto mode (BOOL) • TRUE : automatic • FALSE : manual 2. PV (REAL) : process measure 3. SP (REAL) : order 4. X0 (REAL) : Setting Value for manual mode 5. Kp (REAL) : proportional gain 6. Ti (REAL): intégral gain (in s) 7. Td (REAL) : derivative gain (in s) 8. TS (TIME) : sampling period (in ms) 9. Min (REAL) : output lower limit 10. Max (REAL) : output upper limit 11. Xout : output PID value 12. Iterm 13. Old_Iterm 14. memerror 15. old_date 16. last_date
Returned values	DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of the document.

It is possible to compose a P, or PI, or PD, or PID controller. To do this, simply disable the action that is not used. An action (proportional, integral or derivative) is disabled when the parameters are as following: Kp=1 or Ti=0 or Td=0.

Adjustment method

The PID controller setting is done through the choice of parameters Kp, Ti, Td.
Some experimental analysis process methods are available to determine the parameters Kp, Ti, Td.
For example, the typical specifications for the control of chemical process or thermal process are as following:

- Ti from 3 to 1000 seconds,
- Td from 3 to 150 seconds.

A method of controller setting : the method by trial and error.

The online setting can be done empirically using a procedure that can be summarized as follows:

- Start the PID controller,
- Remove the integral and derivative actions,
- Set the proportional gain Kp to a low value,
- Do a small change in the setpoint and observe the system response. As the gain is very small, the response will be very damped,
- double the gain and repeat the previous step. Continue so until the response becomes oscillatory. Call this value Kpu (ultimate Kp),
- set Kp to (Kpu / 2),
- Do the same by reducing Ti by a factor 2, to obtain an oscillatory response to a small variation of the setpoint,
- set Ti to the double of that value,

The procedure is the same for the derivative constant: increase Td until a oscillating response, then set Td to 1/3 of that value.

Functions de commande des Leds des Cartes d'entrée sorties :

Input/output LEDs are automatically refreshed by the kernel. However, the workbench provides functions to control the status of these LEDs differently (e.g. reversing the logic of digital I/Os). There is one function per board:

- LedDI310(RangCarte, Leds_1_32);
- LedDI410(RangCarte, Leds_1_32, Leds_33_64);
- LedDI312(RangCarte, Leds_1_32, Leds_33_64);
- LedDO310(RangCarte, Leds_1_32);
- LedDIO210(RangCarte, LedsI_1_16, LedsO_1_8);
- LedAI110(RangCarte, LedsV_1_8, LedsR_1_8);
- LedAI210(RangCarte, LedsV_1_16, LedsR_1_16);
- LedAO121(RangCarte, Leds_1_8);
- LedAIO320(RangCarte, Leds_1_8);

Caution : the Led command must be rewritten every cycle, otherwise it is overwritten by the ISaGRAF kernel.

Function	**LedDI310()**
Action	Controls the LEDs of a DI310 board
Parameters	• Rank (USINT) : Board order (1 to 15). • Leds_1_32 (UDINT) : state of the 32 leds (0=OFF, 1=ON)
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.

Function	**LedDI410()**
Action	Controls the LEDs of a DI410 board
Parameters	• Rank (USINT) : Board order (1 to 15). • Leds_1_32 (UDINT) : state of the first 32 leds (0=OFF, 1=ON) • Leds_33_64 (UDINT) : state of the last 32 leds (0=OFF, 1=ON)
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.

Function	**LedDI312()**
Action	Controls the LEDs of a DI312 board
Parameters	• Rank (USINT) : Board order (1 to 15). • LedsV_1_32 (UDINT) : state of the 32 green leds (0=OFF, 1=ON) • LedsR_1_32 (UDINT) : state of the 32 red leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedDO310()**
Action	Controls the LEDs of a DO310 board
Parameters	• Rank (USINT) : Board order (1 to 15). • Leds_1_32 (UDINT) : state of the 32 leds (0=OFF, 1=ON)

Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedDIO210()**
Action	Controls the LEDs of a DIO210 board
Parameters	• Rank (USINT) : Board order (1 to 15). • LedsI_1_16 (UDINT) : state of the 16 green leds (0=OFF, 1=ON) • LedsO_1_8 (UDINT) : state of the 8 red leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedAI110()**
Action	Controls the LEDs of a AI110 board
Parameters	• Rank (USINT) : Board order (1 to 15). • LedV_1_8 (UDINT) : state of the 8 green leds (0=OFF, 1=ON) • LedR_1_8 (UDINT) : state of the 8 red leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedAI210()**
Action	Controls the LEDs of a AI210 board
Parameters	• Rank (USINT) : Board order (1 to 15). • LedV_1_16 (UDINT) : state of the 16 green leds (0=OFF, 1=ON) • LedR_1_16 (UDINT) : state of the 16 red leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedAO121()**
Action	Controls the LEDs of a AO121 board
Parameters	• Rank (USINT) : Board order (1 to 15). • Leds_1_8 (UDINT) : state of the 8 green leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Function	**LedAIO320()**
Action	Controls the LEDs of a AIO320 board
Parameters	• Rank (USINT) : Board order (1 to 15). • Leds_1_8 (UDINT) : state of the 8 green leds (0=OFF, 1=ON)
Returned Value	(DINT) Operation status : • >=0 : number of elements written, operation successful, • other : operation failed, check the error code at the end of this chapter.

Error codes

Error code	Meanings
-325	LED command : error on the board position
-326	LED command : error on the board type
-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
-508	Error file deleting.

Chapter

5

bytes and ASCII serial communication

Overview

This chapter describes the serial communication functions with a simple protocol.

We detail in this chapter the management of :

- Serial communication principle
- Initialization functions and closing
- Reading and writing bytes functions
- Reading and writing characters functions

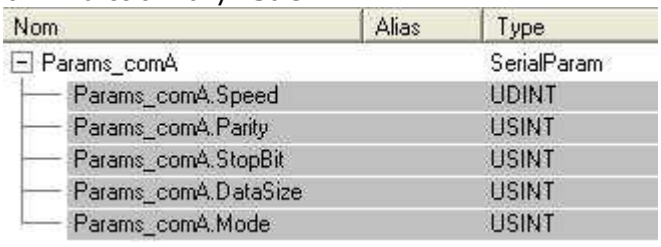
communication principle

LT200 can manage on its four serial ports a simple protocol : this simple protocol is designed to manage terminals and devices with an ASCII protocol, without the time constraints associated with byte transmission and reception.

We have defined the type SerialParam : it is a data structure within the modbus RTU, used in initialization serial ports functions, to specify the parameters setting of the port.

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 : declaration in dictionnary ISaGRAF



initialization and closing functions

Serial port opening

Function	**Serial_Open()**
Action	Serial port initialization
Parameters	• COM (USINT) : COM number : 0 to 3 • SerialParams (Serialparam) : serial port parameters. • Modem (BOOL) : TRUE : modem signals managed ; FALSE : modem signals not managed.
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Example	`(* initialization serial port com 1 *)`
	`IF Status_i_serop = 0 THEN`
	`  Status_i_ serop := Serial_Open(1, SerPar1, True);`
	`END_IF;`

Serial port closing

Function	**Serial_Close()**
Action	Serial port closing
Parameters	COM (USINT) : COM number : 0 to 3
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Sending and receiving bytes

Sending bytes

Function	**SerialByte_Write()**
Action	Sending bytes on a serial port
Parameters	• COM (USINT) : COM number : 0 to 3 • Tab (ByteTab) : variable containing the bytes to transmit. • ByteNumber (WORD) : number of bytes to transmit. • Offset (WORD) : offset in bytes of the first element to be transmitted in Tab
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Number of elements in reception queue

Function	**Serial_Number()**
Action	Number of elements in reception queue
Parameters	COM (USINT) : COM number : 0 to 3
Returned value	(DINT) Number of elements in reception queue

Reading bytes

Function	**SerialByte_Read()**
Action	Reading bytes in reception queue
Parameters	• COM (USINT) : COM number : 0 to 3 • Tab (ByteTab) : Variable receiving bytes read • ByteNumber (WORD) : number of bytes to read • Offset (WORD) : offset in bytes of the first element read in Tab
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Sending and receiving characters

Sending characters

Function	**SerialChar_Write()**
Action	Sending characters on a serial port
Parameters	• COM (USINT) : COM number : 0 to 3 • Msg (STRING[255]) : message containing the characters to transmit • CharNumber (USINT) : number of characters to transmit. • Offset (USINT) : offset in characters of the first element to transmit in Msg (0 to 254)
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

Reading characters

Function	**SerialChar_Read()**
Action	Lecture de caractères sur un port série
Parameters	• COM (USINT) : COM number : 0 to 3 • Msg (STRING[255]) : message containing the characters read • CharNumber (USINT) : number of characters to read • Offset (USINT) : offset in characters of the first element read in Msg (0 à 254)
Returned value	(DINT) operation Status : • 1 : operation successful, • other : operation failed, check the error code at the end of this chapter.

error codes

Error code	Meaning
-100	Bad COM number
-101	Unable to open the COM
-102	Bad speed settings
-103	Bad parity settings
-104	Bad number of stop bits settings
-105	Bad number of data bits

Error code	Meaning
-106	Unable to configure the COM
-107	Configuration RS232/485/422 impossible
-108	Unable to initialize the port
-109	The specified port is already open
-110	Bad mode
-111	The specified port is closed
-112	offset error or elements number error
-113	No elements or error of port access
-114	This port is busy by the console connection

Chapter

6

Modbus RTU and TCP communication

Overview

This chapter describes the modbus RTU and TCP use on LT200.
We detail in this chapter the following steps :

- Modbus protocol
- Modbus Slave protocol: RTU and TCP
- Modbus Master protocol: RTU and 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 LT200 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/

LT200 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 address	bit range	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Words 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																		

Bit at address : 16 * 2 + 12 = 75

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

Address Bit = 16 * address of word + rank of bit

The sum of all sizes tables slaves should not exceed 16384 words.

Only bits of the 4096 first words are accessible (the addressing format can not exceed 65535).

Maximum number manageable by LT200 on RTU and TCP:

- maximum number of modbus masters : 9.
- maximum number of modbus slaves : 16.

Functions modbus codes managed by the LT200 are:

- 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

We have defined the type ModbusTable : it is the type of variable created to represent the Modbus table in LT200 :

Data Structure	**ModbusTable**
(WORD)	Table of modbus words [0.. 16383]

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.

Bits can be accessed only up to the word at address 4095.

All addresses have an offset at 0 by default.

Example of declaration in ISaGRAF dictionary:

Nom	Alias	Type
⊞ ModTab		ModbusTable

Function Modbus_Init :

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

We have defined the type SerialParam : it is a 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

Nom	Alias	Type
⊟ Params_comA		SerialParam
Params_comA.Speed		UDINT
Params_comA.Parity		USINT
Params_comA.StopBit		USINT
Params_comA.DataSize		USINT
Params_comA.Mode		USINT

Data Structure	**SerialParam**
Speed (UINT)	Port Speed (unity : bauds) : 1200, 2400, 4800, 9600, 19200, 38400

Data Structure	**SerialParam**
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 := 0;(* RS232*)`
	`END_IF;`

Modbus slave protocol

Modbus service must first have been initialized.

Each open slave is working on a segment ; a segment is a sub modbus table, 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 to 255). • ComName (STRING) : « /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 structure • GeneralOffset (UINT) : offset in words to open the segment compared to the general table. • Size (UINT) : size in 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/1',Params_comA,0,100,0);`
	`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: the choice is made depending on whether the values entered for the fields « Trigger » and « Period » of the structure (see table below):

The structure of a modbus master request is as follows: a structure that brings together all information relating to a Modbus request.

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)	Case of a trigger request : trigger. The request is issued when the value change from FALSE to TRUE.
SlaveNumber (USINT)	slave number (address): used in RTU (0 to 255), 1 in TCP
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 • 23 : write n words, read m words
ReadSlaveAddress (WORD)	Read Offset address in the modbus slave table.
ReadMasterOffset (WORD)	Read Offset address in the local LT200 table for the data read in the slave.
ReadLength (WORD)	Number of elements to read : • limited to 2000 for bit (codes 1 and 2) • limited to 125 for words (codes 3, 4 and 17)
WriteSlaveAddress (WORD)	Wrinting Offset address in Modbus slave table.
WriteMasterOffset (WORD)	Offset address in the local LT200 table for the data that have to be write in the slave
WriteLength (WORD)	Number of elements to write : • limited to 800 for write bits (code 15) • limited to 100 for write words (code 16 and 17)
Status (DINT)	Request status : see table ci-dessous
NbFrameOK (UDINT)	Number of successful frames
NbError (UDINT)	Number of failed frames

Example	`(* SQ1 : read 6 words at address 10 in the slave ; 1 request / 500ms*)`
	`requ[0].Period := 500;`
	`requ[0].Trigger := FALSE;`
	`requ[0].SlaveNumber := 1;`
	`requ[0].FunctionCode := 3;`
	`requ[0].ReadSlaveAddress := 10;`
	`requ[0].ReadMasterOffset := 20;`
	`requ[0].ReadLength := 6;`

offsets units are in :

- bits for read and write bits functions
- words for read and write words functions

Adding a request :

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.

Example	`(* adding modbus master request *)`
	`if Status_o_comA_master = 1 AND Status_o_m_req0 = 0 then`
	`Status_o_m_req0 := Modbus_AddRequest(ANY_TO_USINT (1)`
	`,requ[0]);`
	`end_if;`

	`(* getting the modbus master request status for requ[0]*)`
	`ReqStat1 := requ[0].Status;`
	`ReqNbFrameOK1 := requ[0].NbFrameOK;`
	`ReqNbError1 := requ[0].NbError;`

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 to 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
200	Service already initialized
-200	service initialization impossible
-201	Master number incorrect.
-202	Modbus service not initialized.
²201	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

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

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.

Data waiting to read

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.

Reading bytes

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.

Writing bytes

Function	**Socket_ByteWrite()**
Action	Send bytes data
Parameters	• **in** (USINT) Id : Socket ID • **in** (ByteTab) Tab : Bytes table containing the data to send • **in** (WORD) ByteNumber : bytes number to send • **in** (WORD) Offset : Offset from which data will be issued from the table Tab. • **out** (STRING(20)) Dest : IP Address of remote device.
Returned value	Function status (DINT): • =1 : operation successful • <1 : operation failed, check the error code at the end of this chapter

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

Chapter

8

LT200 monitoring and diagnosis

LT200 ISaGRAF LEDs : Power Supply, CPU and I/O boards

Power supply Led

If the power supply is present and correct, the corresponding green LED lights up without flashing.

CPU leds

LED name	Color	Meaning
RUN	Green	• Flashing slowly (2s) if TIC application (ISaGRAF) is running. • Flashing quickly (5/10s) if TIC application (ISaGRAF) is in STOP • Ligth on : prm mode or step to step • Flashing alternately led the PRM: LT200 in starting
FAIL	Orange	• Light on if TIC application (ISaGRAF) is corrupted. • OFF if operation is correct
I/O	Green	Ligth on if operation is correct • flashing if an I/O board is not correctly inserted or if at least one I/O board status is incorrect while the program is running. • Flashing alternately with the Run led : LT200 starting phase
PRM	Green	lights up without flashing if the equipment is in PRM mode when the LT is booted.
WDG	red	• Light ON while WDG isn't n't • OFF if TIC ISaGRAF is in RUN mode ; the hardware watchdog is refreshed by processor.
CP	green	Ligth ON if the I/O coprocessor is runing (program downloaded).
F1	green	Not used
F2	green	Not used

communication leds : serial and Ethernet

LED name	Color	Meaning
Lk	orange	Ethernet link : Light ON if the Ethernet link is wired up to another Ethernet device
Rx	green	Light ON if byte reception on the corresponding serial port
Tx	green	Light ON if byte sending on the corresponding serial port
SM	green	Not used
Te	green	Not used

Input/Output boards leds

Input/output LEDs are automatically refreshed by the kernel. Their management is as following :

Board type	Led on front I/O board face	LED State	Meaning
All	FLT : red color	On	3 possible cases : • general WDG • internal board power supply in fault • no monitoring from CPU
DI310 DI410 DIO210	1 green LED per digital input channel	On	If input is in TRUE state
DO310 DIO210	1 orange LED per digital output channel	On	If output is in TRUE state
AI110 AI210 AIO320	1 green LED per analog input channel	On	If the CPU is monitoring correctly the channel
AO121 AIO320	1 green LED per analog output channel	On	If the CPU is monitoring correctly the channel

console link troubleshooting : PRM mode

For an LT ISaGRAF, switching to Parameter Setting Mode means running only the ISaGRAF kernel with no TIC (Target Independent Code) application downloaded.
This mode, called **PRM**, is symbolized by a LED of the same name on the CPU.
This mode is used in order to restore the console link between the PC and the LT200 ISaGRAF.

To switch to PRM mode :

- switch off the LT,
- Shunt the pins 7 and 8 of COM0
- switch on the LT,
- some seconds after startup, led **PRM** lights up, then the shunt can be removed,
- ISaGRAF start in safe mode and wait for a connection with the workbench on the Ethernet or USB console link.

LT200 log file:

In case of problems, the LED FAIL will be ON : diagnosis informations are available in a log file « isasys.txt », uploadable from the LT200 ftp server using an anonymous connection.
LT200 ISaGRAF has too a telnet server: you can connect to it with a telnet client, and read the event file « isasys.txt »: this file is present in the following repertory : « home/anonymous » ; once the telnet connection established, enter the command:

- « cd home/anonymous »
- « cat isasys.txt »

At the end of this file, a warning counter indicates a potential problem : move up in the list of messages in order to watch the problem ; the list of messages is limited to 1000, when exceeded, the file is reset.

Some information is in text, other form of warning code:

warning code	Meaning
a : 0	Normal startup of ISaGRAF kernel
b : 0	Normal stop of ISaGRAF kernel
6 : x	Corruption of retain variables. Normal in the case of first started.
7 : x	Error recovery of retain variables: check the size « Backup total size is xxx bytes » given earlier in the file. Normal in the case of first started.
10 : x 11 : x	A user function (or function block) used in the project isn't embedded in the LT200.
1C :x	Error when opening IO boards drivers