Gamme LT

MANUEL
DE MISE EN ŒUVRE LOGICIELLE
ISAGRAF V3
POUR CPU3XX

P DOC LT 001F IsagrafV3 V2.3

FOURNITURES

Le Kit **LT ISaGRAF** comprend :

- Le présent manuel utilisateur
- 1 câble RS232 permettant la connexion du PC au LT.
- 1 disquette : "Librairies LT ISaGRAF".

Le manuel de mise en oeuvre matérielle est disponible sur notre site web.

Cette documentation présente les fonctionnalités :

- Du logiciel embarqué (noyau) version 4.74
- Des librairies LT ISaGRAF : version 3.1

SUPPORT TECHNIQUE :
Tél : +33 (0) 562 240 546
e-mail : support@leroy-autom.com

Pré-requis : La mise en œuvre du LT ISaGRAF nécessite les compétences suivantes :
connaissance de l'atelier ISAGRAF et des langages d'automatisme IEC 61131-3.

Leroy Automation
250 rue Max Planck
31670 LABEGE – France

Tél : +33(0) 562 240 550

www.leroy-automation.com

SOMMAIRE

1 PRÉSENTATION GÉNÉRALE 7

1.1 RESSOURCES MATÉRIELLES DU LT 9
1.2 CYCLE DES TRAITEMENTS EFFECTUÉS 10

2 PREMIÈRE MISE EN OEUVRE 11

2.1 INSTALLATION DE L'ATELIER ISAGRAF V3 11
2.2 INTÉGRATION DES LIBRAIRIES ET PROJETS SPÉCIFIQUES AU LT DANS L'ATELIER ISAGRAF 11
2.2.1 Décompression des fichiers sources 11
2.2.2 Intégration des librairies spécifiques au LT dans l'atelier ISaGRAF 11
2.2.3 Intégration des projets spécifiques au LT dans l'atelier ISaGRAF 12
2.3 CRÉER UN PREMIER PROJET 13
2.4 CONFIGURATION MATÉRIELLE 13
2.5 COMPILER LE PROJET 14
2.6 CONNECTER L'ATELIER À L'AUTOMATE 14
2.7 TÉLÉCHARGER L'APPLICATION 14
2.8 MISE AU POINT D'UNE APPLICATION 15

3 LA CARTE CPU 16

3.1 LE PARAMÉTRAGE DE LA CARTE CPU 16
3.1.1 Paramètre « mode » : modes de fonctionnement disponibles 17
3.1.2 Paramètre « params_com » : paramètres de la liaison console 18
3.2 SAUVEGARDE DES DONNÉES 20
3.2.1 Accès à la mémoire EEPROM 20
3.2.2 Variables non volatiles 21
3.2.3 L'horloge secourue 21
3.3 FONCTION DE RÉGULATION : PID 23

4 GESTION DES COMMUNICATIONS 25

4.1 PRINCIPE DE COMMUNICATION SUR LES PORTS DE COMMUNICATION 26
4.2 LES PROTOCOLES SUR RÉSEAU RS232/RS485 30
4.2.1 Le protocole Jbus Esclave 30
4.2.2 Le protocole Jbus Maître 32
4.2.3 Le protocole d'émission/réception de caractères 34
4.2.4 Le protocole d'émission/réception d'octets 36
4.2.5 Signaux de contrôle de la liaison RS232 36
4.3 LES PROTOCOLES SUR RÉSEAU ETHERNET 37
4.3.1 Le protocole Telnet 38
4.3.2 Le protocole Modbus/TCP 39
4.3.3 Le protocole SNMP V1 42
4.3.4 Le protocole SMTP : envoi de courrier électronique 45

5 LES CARTES D'ENTRÉES/SORTIES 47

5.1 CARTE DI310 47
5.2 CARTE DI410 47
5.3 CARTE DO310 47
5.4 CARTE AI110 47
5.5 CARTE AO121 47
5.6 CARTE AI210 47
5.7 EQUIPEMENT DI312 47
5.8 EQUIPEMENT DIO210 49
5.9 EQUIPEMENT AIO320 49
5.10 EQUIPEMENT DI130 49
5.11 EQUIPEMENT DIO130 49

6 DIAGNOSTIC ET DÉPANNAGE 50

6.1 LECTURE DU STATUS DES CARTES CPU ET D'ENTRÉES/SORTIES 50
6.2 ERREURS REMONTÉES À L'ATELIER 52
6.3 DÉPANNAGE DE LA LIAISON CONSOLE : PASSAGE DU LT EN MODE PARAMÉTRAGE 52
6.4 LES LEDS DU LT ISAGRAF 54
6.4.1 Led de l'alimentation 54
6.4.2 Leds de la CPU 54
6.4.3 Pilotage des leds des cartes d'entrées/sorties 54
6.5 IDENTIFICATION DU LT 57
6.5.1 Version du noyau embarqué sur la cible LT 57
6.5.2 Type de CPU montée sur le LT 57
6.5.3 Numéro de série du LT 57

1 Présentation générale

Les blocs d'unités centrales LUC3xxx composés de cartes CPU3xx de la gamme LT de Leroy Automation sont équipées d'un noyau permettant leur programmation avec l'atelier ISaGRAF Version 3.

Cette documentation présente la mise en oeuvre de cet atelier sur la cible LT.

Dans la suite du document la mention « LT Isagraf » signifie une CPU33x ou CPU35x équipé d'un kernel Isagraf.

Par ailleurs, l'installation de l'atelier ISaGRAF proprement dit ainsi que son utilisation (création d'un projet, programmation...) sont à lire dans le "Guide utilisateur" d'ISaGRAF fourni avec l'atelier.

Le schéma suivant offre une vue d'ensemble de l'architecture ISaGRAF sur le LT :

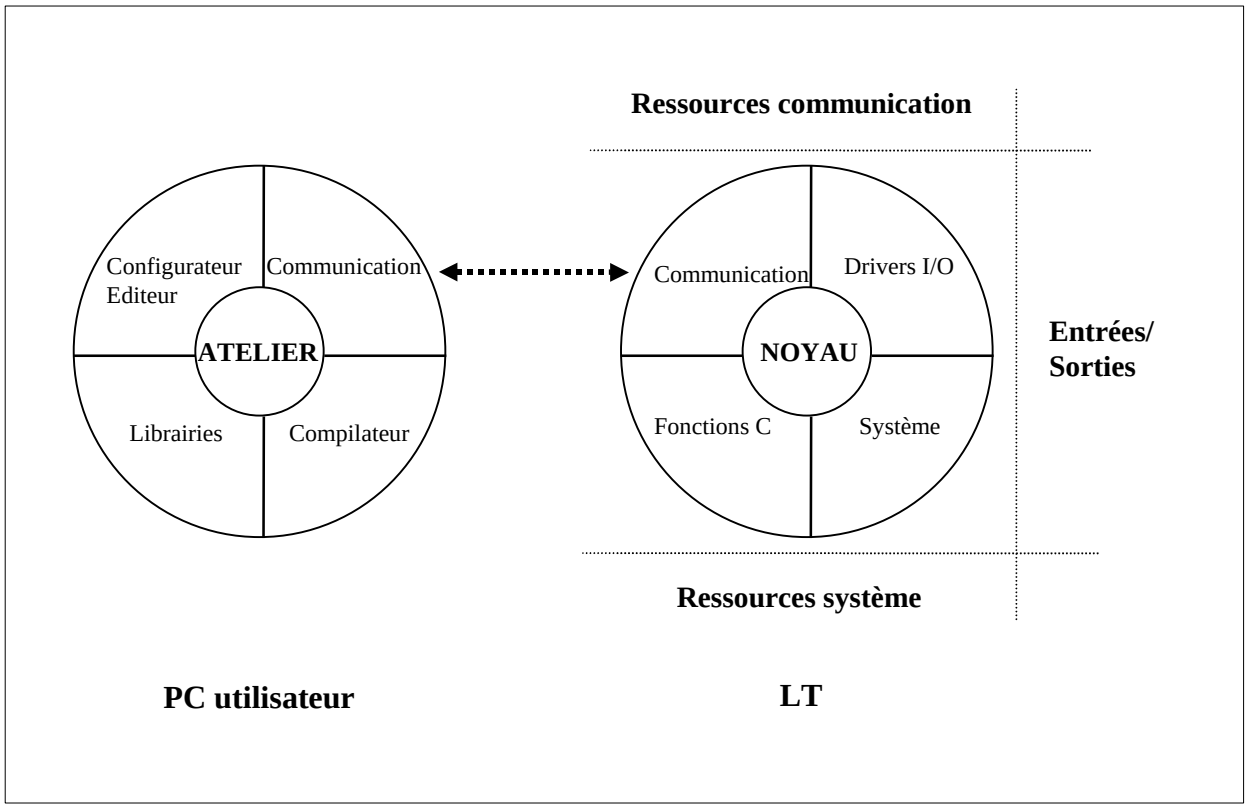


Figure 1 : architecture ISaGRAF sur le LT

La **cible LT** supporte le **noyau ISaGRAF** et permet l'accès aux ressources suivantes :

- système : horloge, mémoire…
- drivers d'entrées/sorties
- drivers de communication.

L'atelier ISaGRAF permet de créer et modifier de nouveaux projets à destination de la cible. Avec l'utilitaire "**Librairies**", l'utilisateur dispose de fonctions permettant de gérer le LT :

- insertion de cartes d'entrées/sorties,
- insertion de drivers de communication (Modbus, Jbus, Ethernet),

- fonctions de pilotage des entrées/sorties (leds),
- fonctions de diagnostic du LT (lecture du status des cartes...).

1.1 Ressources matérielles du LT

Le LT est une base matérielle de travail. Sur cette base, le noyau ISaGRAF va s'exécuter et utiliser les ressources matérielles disponibles. C'est pourquoi, une présentation, même succincte du matériel, peut aider à la compréhension des modes de fonctionnement du LT et des fonctions associées.

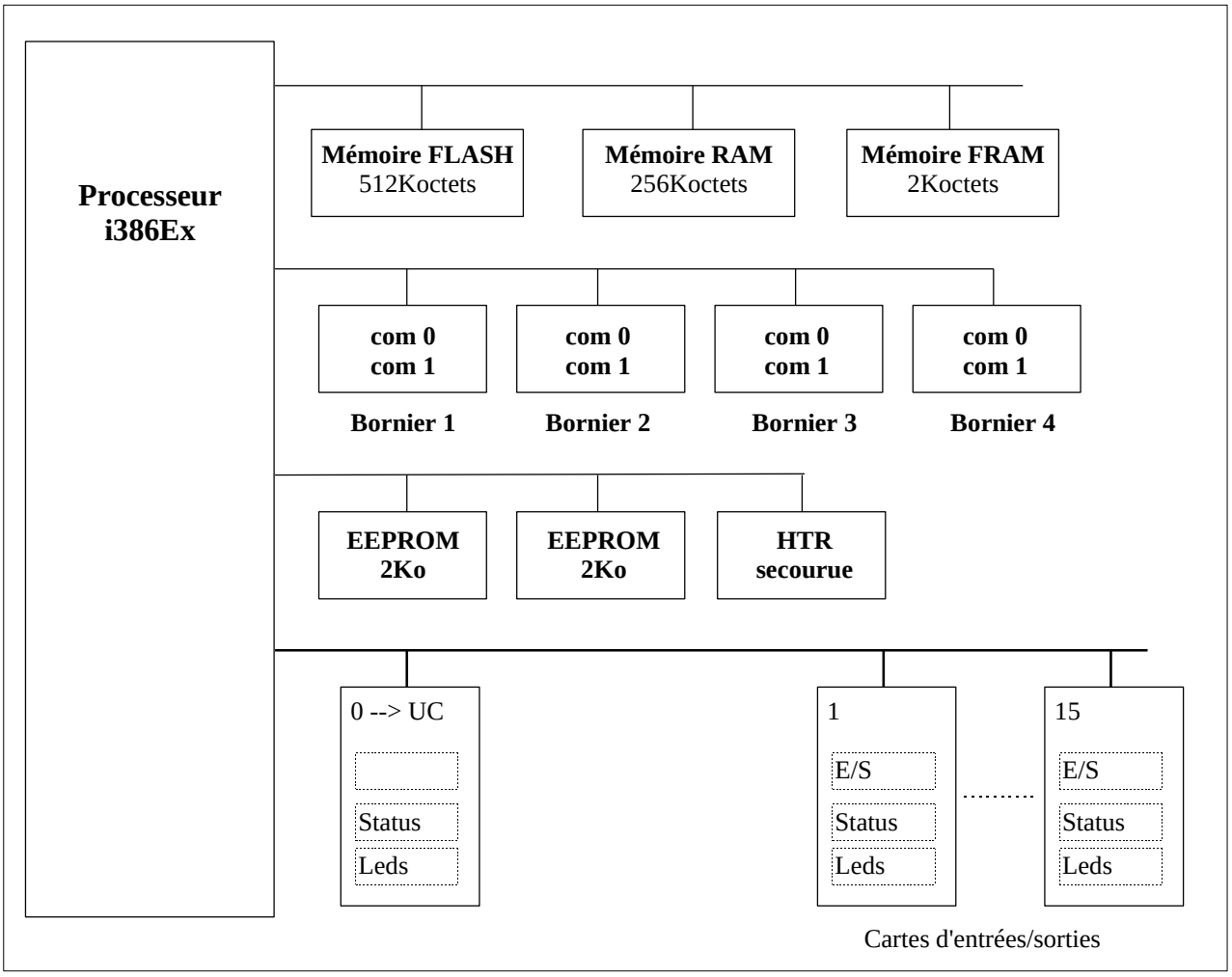


Figure 2 : ressources matérielles

1.2 Cycle des traitements effectués

Suivant le schéma bien connu d'un automate, le noyau ISaGRAF exécute le cycle de traitements suivant:

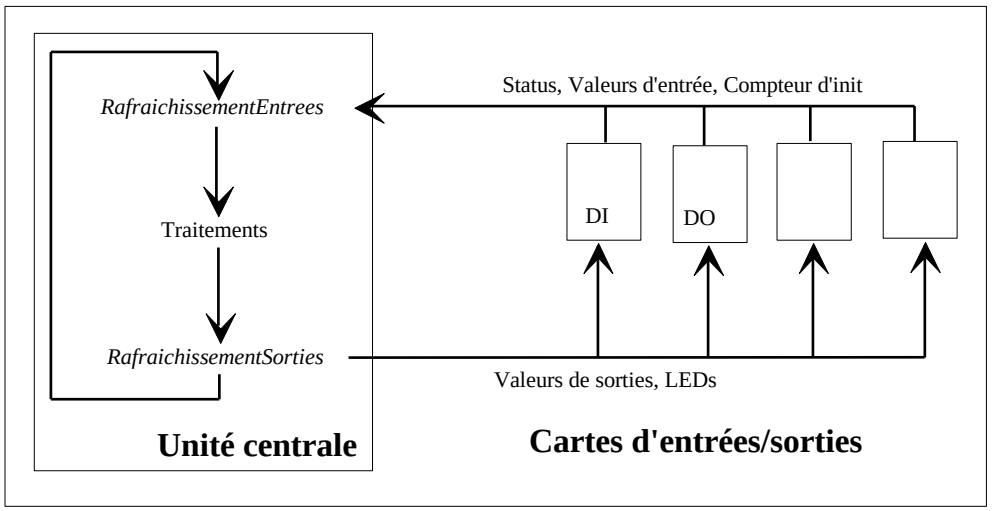


Figure 3 : cycle de traitements du LT ISaGRAF

2 Première mise en oeuvre

2.1 Installation de l'atelier ISaGRAF V3

A partir du "Guide utilisateur" d'ISaGRAF, suivez les instructions du chapitre "Mise en route". La configuration matérielle et logicielle requise par l'atelier ISaGRAF est suffisante pour fonctionner avec la cible LT.
Lancer l'atelier ISaGRAF.

2.2 Intégration des librairies et projets spécifiques au LT dans l'atelier ISaGRAF

Leroy Automation fournit
- des librairies (ou bibliothèques) spécifiques au LT,
- des exemples de projets.

2.2.1 Décompression des fichiers sources

Exécuter le fichier « LibrairiesLtIsagraf.exe » fourni :
- choisir un nouveau répertoire sur votre disque dur, par exemple « C:\Isawin\Lai »
- lancer la décompression.

Le répertoire que vous avez choisi contient alors des fichiers correspondant à des objets ISaGRAF :

Nature	Type de l'objet	Extension de fichier
Librairies	Configurations d'E/S	*.ria,
	Cartes d'E/S	*.bia
	E/S complexes	*.xia
	Fonctions	*.iia
	Blocs fonctionnels	*.aia
	Fonctions C	*.uia
	Blocs fonctionnels C	*.fia
Projets		*.pia

2.2.2 Intégration des librairies spécifiques au LT dans l'atelier ISaGRAF

Lancer le gestionnaire de librairies.
- Dans le menu "**Fichier**"/"**Autres librairies**", sélectionner un type de librairie : « **Cartes d'E/S** » par exemple.
- Dans le menu "**Outils**"/"**Archiver**" : sélectionner chaque élément contenu dans la liste "**Archive**"
- Sélectionner dans l'unité de sauvegarde le répertoire « C:\Isawin\Lai » ou celui que vous aviez choisi lors de l'étape de décompression.
- cliquer sur le bouton "**Restituer**".

Répéter cette opération pour chaque type d'objet en les sélectionnant tour à tour, à partir du menu "**Fichier**"/"**Autres librairies**", ou de la liste déroulante, sous la barre de menu, et en répétant l'opération de restitution précédente.

Une fois tous les objets intégrés dans l'atelier, toutes les librairies spécifiques au LT pourront être utilisées comme des librairies ISaGRAF standard.
L'utilitaire "Librairies" fourni avec l'atelier ISaGRAF et utilisé pour générer ces fonctions permet de visualiser la "**Fiche technique**" spécifique à chaque fonction. Elle contient tous les renseignements nécessaires à l'utilisation des fonctions ou cartes d'entrées/sorties (paramètres, code de retour, restrictions, exemples...).

2.2.3 Intégration des projets spécifiques au LT dans l'atelier ISaGRAF

Lancer le gestionnaire de projets.
Réaliser la même opération de restitution que pour les Librairies :

- dans le menu "**Outils**"/"**Archiver**"/"**Projets**", sélectionner chaque élément contenus dans la liste "**Archive**"
- cliquer sur le bouton "**Restituer**".

Liste des projets disponibles :

- Lt232sig : mise en œuvre des signaux RS232 des borniers Com311
- Lte2prom : mise en œuvre de la lecture et de l'écriture des E2prom
- LThtr : mise en œuvre de l'horloge sauvegardée
- Ltjbusm1 : mise en œuvre du protocole modbus asynchrone maître
- Ltjbusm2 : mise en œuvre du protocole modbus asynchrone maître avec utilisation de grafcet fils
- Ltjbuss : mise en œuvre du protocole modbus asynchrone esclave
- Ltledio : mise en œuvre du pilotage des leds des cartes d'entrées et sorties
- Ltio : mise en œuvre du câblage des cartes d'entrées et sorties, lecture de status, compteurs d'initialisation
- LtNulcar : mise en œuvre d'une communication simple avec traitement du caractère nul
- LtNulpro : mise en œuvre d'une communication simple : émission et réception d'octets.
- Ltpid : mise en œuvre d'un bloc fonctionnel pid
- Ltsaveva : mise en œuvre des variables non volatiles
- LtTCPe : mise en œuvre du protocole modbus/TCP esclave
- LtTCPm : mise en œuvre du protocole modbus/TCP maître
- Lttemail : mise en œuvre du protocole SMTP : envoi d'email
- LTxmulti : exemple de mise en œuvre simultanée d'un ensemble de protocoles de communication : modbus/TCP (maître et esclave), modbus asynchrone (maître et esclave).
- LTxomod : exemple de la fonction passerelle réseau Ethernet / réseau asynchrone : protocoles modbus/TCP esclave et modbus asynchrone maître

2.3 Créer un premier projet

On peut réaliser un premier projet « minimal » sans traitement de la manière suivante :

- Du Gestionnaire de projets de l'atelier ISaGRAF, **créer un nouveau projet** : cliquer menu "**Fichier**"/"**Nouveau Projet**"

✓ La fenêtre « **Créer un nouveau projet** » apparaît.

- Indiquer le nom de votre projet (inutile de choisir la configuration d'entrées sorties si elle ne correspondant pas à votre configuration) et appuyez sur le bouton OK

✓ Votre projet apparait dans la liste de projets de votre gestionnaire de projets.

- **Cliquer sur ce nouveau projet** pour l'ouvrir.

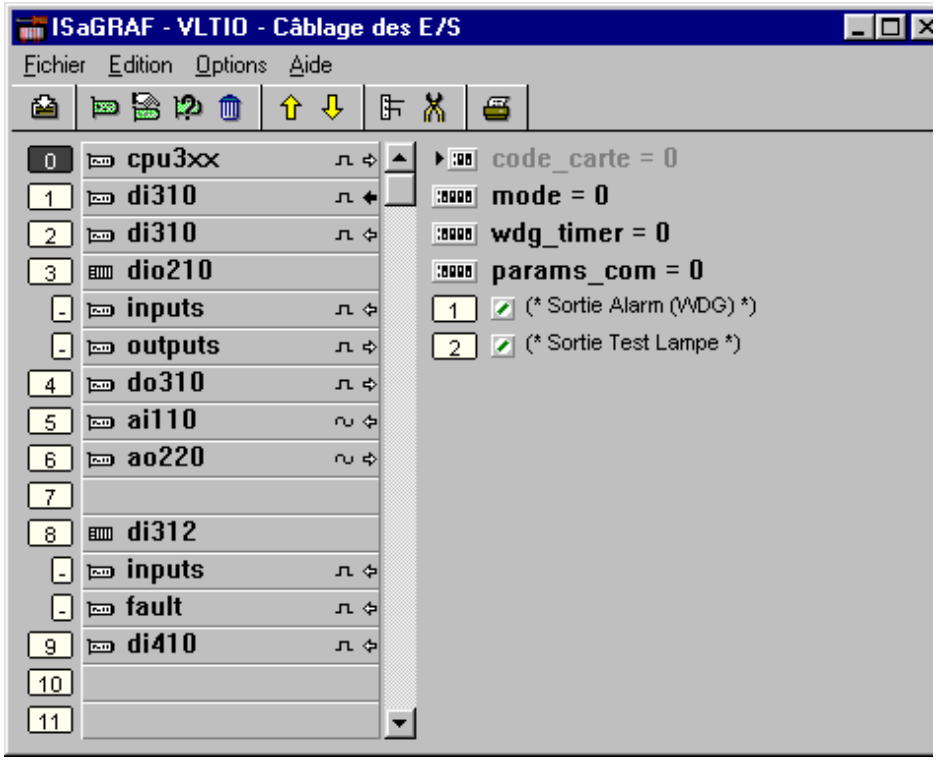
2.4 Configuration matérielle

- Cliquer sur le menu "**Projet**" /"**Câbler les E/S**" ou sur l'icône correspondante.

Une fenêtre propose 256 emplacements (slots) de cartes d'entrées/sorties. Seuls les 16 premiers peuvent être utilisés pour programmer un LT.

Slot 0 : il est réservé à la CPU : double cliquer sur le slot 0 : La librairie des cartes apparaît. Choisir le bloc CPU (CPU3xx ou CPUeth pour une CPU Ethernet) correspondant à votre configuration. Nota : les ports de communications associés à une CPU sont déclarés dans le programme ISaGRAF à l'aide de fonctions spécifiques.

Slots 1 à 15 : ils sont réservés à la déclaration des cartes d'entrée et de sortie. Selon l'ordre de présence des blocs suivant la CPU, double cliquez sur le slot n°x pour déclarer le bloc à l'emplacement logique n° x sur le rack. La librairie des cartes apparaît. Si le bloc n'est pas dans la liste, choisir la librairie des Equipements en cochant la case correspondante. Choisir le bloc dans la liste.
Voici un exemple de configuration et sa correspondance sur le LT :



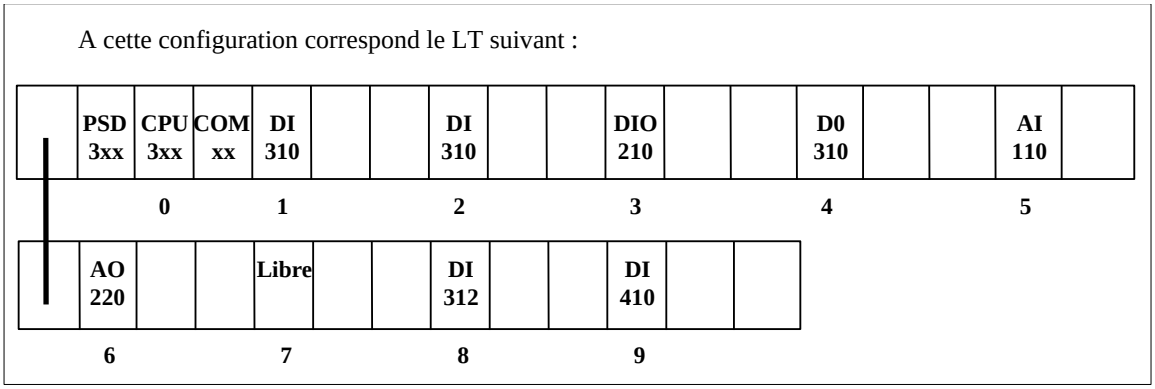


Figure 4 : principe de configuration des cartes CPU et d'entrées/sorties

Sous chaque carte du LT est noté son rang logique : CPU = 0, 1°E/S = 1, ... , 15° E/S = 15. Noter que ce rang correspond à la numérotation de l'éditeur de câblage ISaGRAF. La carte DIO210 est un équipement composé de plusieurs cartes. Il ne prend qu'un slot sur la liste des cartes déclarées. De même pour la carte DI312.
Nota : les blocs d'extension ne se configurent pas. Dans l'atelier de câblage, un LT est vu comme un seul rack.

2.5 Compiler le projet

- Dans le menu "**Codage**"/"**Options de compilation**" , libérer le choix Simulate et sélectionner l'option : "**ISA86M : TIC code for INTEL**".
- Générer l'application : menu "**Codage**" / "**Générer l'application**"

2.6 Connecter l'atelier à l'automate

- raccorder le port série du PC au port série Bornier 1 Com1 de l'automate avec un câble RS232.

Lors de la 1ère utilisation, il faut passer l'automate en mode PRM :
- Mettre l'automate hors tension
- Lancer l'utilitaire SSTB (disponible dans les librairies LT ISaGRAF) :
  - exécuter le fichier SSTBSetup pour installer SSTB.
- Sur l'onglet Mode, cliquer sur le bouton « Run PRM function »
- Mettre le LT sous tension immédiatement après : la led Prm s'allume.

2.7 Télécharger l'application

- Cliquer le menu "**Test**" / "**Tester**"
- ✓ La fenêtre du débogueur apparaît avec le message suivant : « **Pas d'application** »
- Choisir le menu "**Fichier**" / "**Transférer**" **/ ISA86M : TIC Code for Intel**
- Cliquer le bouton « **Transfère** »

✓ L'ascenseur horizontal donne l'avancement du chargement,

A la fin du chargement, l'automate reboote, le bandeau indique alors « Déconnecté » avec les messages BREAK et FRAME, puis apparaît le message « Nom de l'application active », ainsi que « MARCHE et les temps de cycle du projet ».

Après le transfert de cette première application, l'automate se réinitialise et exécute la nouvelle application : la led Run clignote lentement, la led Prm est éteinte.
Si une erreur est détectée, elle sera remontée vers l'atelier sous la forme d'un code erreur (cf. paragraphe Erreurs 6.1).

Lorsque vous mettez un LT ISaGRAF sous tension, il exécute l'algorithme suivant :

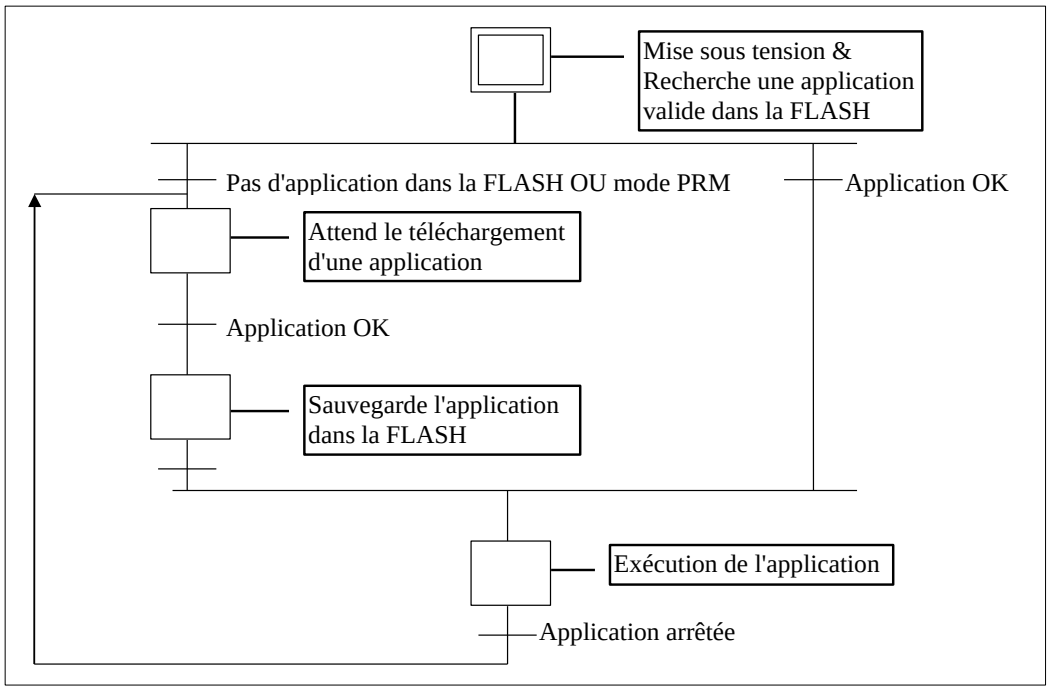


Figure 5 : principe d'exécution d'un LT ISaGRAF

Suivant le grafcet vu précédemment deux cas se présentent :

- une application n'a pu être récupérée dans la FLASH : le bandeau du debogueur affiche "Pas d'application",
- une application a été récupérée dans la FLASH : le bandeau du debogueur affiche " 'nom de l'application active", le temps de cycle et l'état de cette application.

Consulter le chapitre "**Mise au point**" du "Guide utilisateur" ISaGRAF afin d'utiliser correctement le debogueur.
Important : la récupération d'une application contenue dans le LT ISaGRAF n'est pas possible.

2.8 Mise au point d'une application

Une application peut être mise au point de deux manières :

- sur le PC avec le simulateur accessible par le menu "Test"/"Simuler",
- sur le LT avec le débogueur accessible par le menu "Test"/"Tester".

L'utilisation de ces deux modes de mise au point est détaillée dans le "Guide utilisateur" d'ISaGRAF.
La taille du code TIC généré par l'atelier ISaGRAF correspond à la taille du fichier **appli.x8m**. Ce fichier se trouve dans le répertoire \isawin\apl\"nom application"\.

La taille d'une application est limitée à 64Ko : fichier appli.x8m < 64Ko.

3 La carte CPU

3.1 Le paramétrage de la carte CPU

La carte CPU se décline en 2 versions :

- **cpu3xx = module CPU33x sans port Ethernet.**
- **cpueth = module CPU35x avec un port Ethernet.**

Elle doit obligatoirement se trouver sur le **premier slot** de l'éditeur de câblage des E/S.

Paramètres communs à ces deux versions :

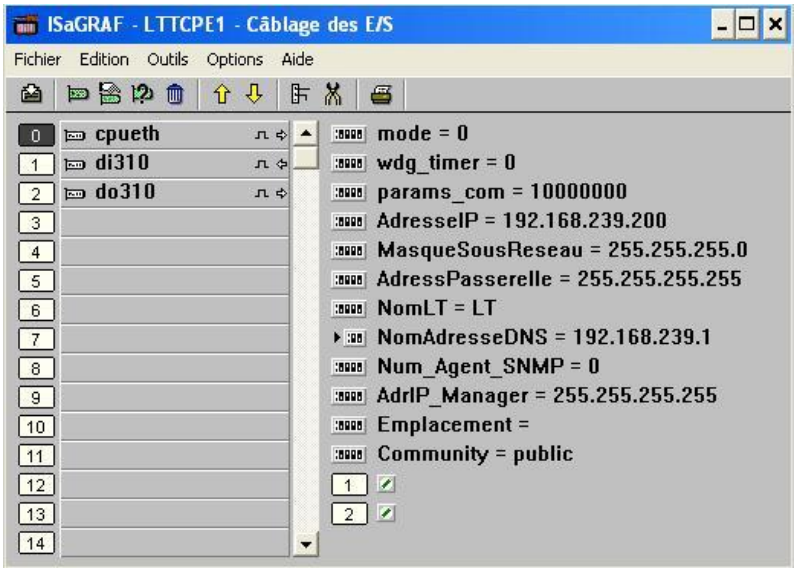
- **code_carte** : il est figé. Il est égal à 0 pour la cpu3xx, non visible pour la cpueth.
- **mode** : permet de configurer des fonctionnements différents du noyau (type Word, 0 par défaut),
- **wdg_timer** : permet de borner le temps de cycle du LT (en millisecondes). (type Word, 0 par défaut),
- **params_com** : paramètres de communication de la liaison console. (type Long Hexa, 0 par défaut).

Cette carte dispose également de **deux sorties booléennes** représentant :

- la **sortie Alarme** de l'alimentation. Le pilotage de la sortie Alarme inhibe la gestion et le rafraîchissement des E/S,
- la **sortie Test Lampe** permettant de vérifier si les leds des cartes entrées/sorties fonctionnent. Le pilotage de cette sortie allume toutes les leds des cartes d'entrées/sorties.

Exemple de paramétrage de la carte cpu3xx :

Exemple de paramétrage de la carte cpueth :



Paramètres supplémentaires liés à la version cpueth :

- **Adresse IP** : identifie le réseau et l'équipement (l'automate LT) sur un réseau TCP/IP.

Par défaut l'adresse IP vaut 255.255.255.255. Dans ce cas le LT va ignorer les autres paramètres et utiliser un serveur d'adresse BOOTP : celui-ci renverra au LT une adresse IP disponible sur le réseau.

Cas particulier : adresse IP : 0.0.0.0 : le LT va démarrer avec l'adresse IP paramétrée avec le logiciel SeeQ (disponible en téléchargement sur notre site WEB) : ce logiciel permet de modifier l'adresse IP, sans modifier le code TIC ISaGRAF.

Format : xxx.xxx.xxx.xxx avec xxx [0..255]

- **Masque Sous Réseau** : masque d'adresse permettant de mettre en évidence la division de l'adresse IP en adresse de réseau et sous-réseau et adresse de l'équipement sur ce sous-réseau. Ce masque de 32 bits comporte des 1 pour les parties de l'adresse de réseau et sous-réseau et des 0 pour les parties de l'adresse de l'équipement.

Format : xxx.xxx.xxx.xxx avec xxx [0..255]

- **Adresse Passerelle** : adresse IP de la passerelle se trouvant sur le réseau. Si le LT veut communiquer en dehors du réseau auquel il appartient, il doit s'adresser à cette passerelle. Par défaut, cette adresse vaut 127.0.0.1 et identifie le LT lui-même (pas de passerelle).

Format : xxx.xxx.xxx.xxx avec xxx [0..255]

- **Nom LT** : nom symbolique du LT (à utiliser pour la connexion au serveur SMTP) :

Format : 10 caractères alphanumériques maximum.

- **Adresse DNS** : Adresse IP du serveur DNS (*Domain Name Server*). Ce serveur retourne une adresse IP à partir d'un nom symbolique désignant un équipement ou un serveur sur un réseau TCP/IP. Il est utile pour les connexions au serveur SMTP et dans la gestion du protocole modbus / TCP maître.

Format : xxx.xxx.xxx.xxx avec xxx [0..255]

- **Num_Agent** : Numéro d'agent SNMP dans la sous-branche LAI (4273) ; par défaut à 0 : service SNMP inactif

- **AdrIP_Manager** : Adresse IP du manager SNMP : seules les requêtes de ce manager sont traitées ; par défaut à 255.255.255.255 : les requêtes de tout manager SNMP sont traitées

- **Emplacement** : renseigne le champs « location » dans la MIB II du LT

Community : « public » par défaut ; il peut être personnalisé : le LT répondra uniquement aux requêtes envoyées par un manager de sa communauté.

3.1.1 Paramètre « mode » : modes de fonctionnement disponibles

- bit 0 : **mode chien de garde (WDG)**
  - à 0 : le chien de garde est géré par le noyau (défaut)
  - à 1 : le chien de garde est géré par l'application ISaGRAF (code TIC)

    la sortie 1 sur la carte cpu doit être cablée : dans le programme applicatif, si elle est forcée à :
    - false : pas de Wdg
    - true : les cartes d'entrées/sorties ne sont plus rafraîchies.
- bit 1 : **mode Secure**
  - à 0 : aucun contrôle de présence (défaut)
  - à 1 : toute carte présente sur le rack doit se trouver dans l'atelier de câblage sinon une erreur IO est signalée et le WDG n'est pas désactivé.

- bit 2 : **mode FreeIO**
  - à 0 : une carte d'E/S se trouvant dans l'atelier de câblage et absente sur le rack génère une erreur IO (défaut)
  - à 1 : une carte d'E/S se trouvant dans l'atelier de câblage peut être absente sur le rack sans générer d'erreur IO.

3.1.2 Paramètre « params_com » : paramètres de la liaison console

Par défaut, pour les 2 versions cpu3xx et cpueth, la liaison console est sur le port série du bornier 1 com1 ; Par défaut, params_com=0 et les paramètres de communication sont :

- bornier 1
- com 1
- esclave 1
- vitesse 19200 bauds
- parité sans
- bit stop 1
- donnés 8 bits

la liaison console peut être paramétrée à n'importe quel autre emplacement des borniers liés à la CPU, aussi bien sur un port série que sur le port Ethernet.

Pour **modifier les paramètres de communication** il faut changer la valeur de params_com. Chaque paramètre est codé sur 1 quartet de params_com, le quartet 1 étant le quartet de poids faible :

Bornier	Com	N° esclave		Données	BitsStop	Parité	Vitesse
Quartet 8	Quartet 7	Quartet 6	Quartet 5	Quartet 4	Quartet 3	Quartet 2	Quartet 1

Le **paramétrage par défaut** (params_com = 0) correspond aussi à **params_com = 1101810A**

3.1.2.1 Liaison console sur un port série RS232

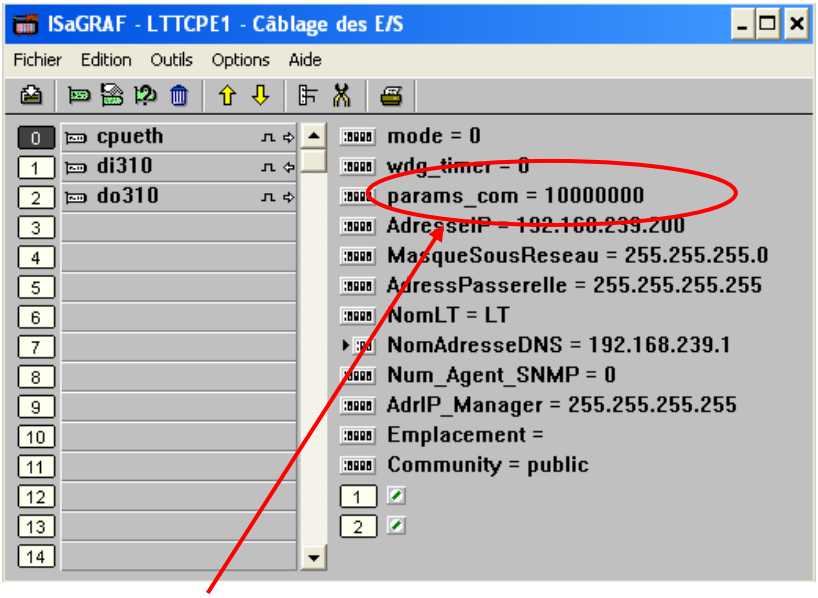
Le codage des paramètres qui correspondent aux paramètres de l'atelier ISaGRAF est le suivant :

Numéro du Quartet	Paramètre	Valeur du Quartet	Valeur du paramètre associé
1	Vitesse	5	600 Bauds
		6	1200 Bauds
		7	2400 Bauds
		8	4800 Bauds
		9	9600 Bauds
		A	19200 Bauds
2	Parité	0	Pas de parité
		1	Parité paire
		2	Parité impaire
3	Bits de stop	1	1 bit de stop
		2	2 bit de stop
4	Bits de données	7	7 bits
		8	8 bits
5 et 6	No esclave	[0..FF]	[0..255]
7	Port de com	0	Com0
		1	Com1
8	No Bornier	1 à 4	Bornier 1 à 4

3.1.2.2 Liaison console sur le port Ethernet

Ce paramétrage n'est possible que sur la version cpueth.

- Modifier le paramètre params_com dans la fenêtre de câblage comme indiqué ci-après :

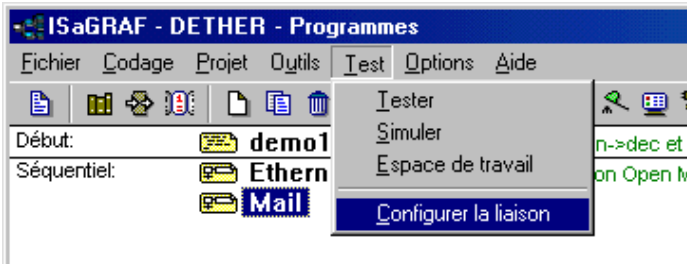


params_com=10000000 désigne le com0 du bornier1 (port Ethernet) comme liaison console.

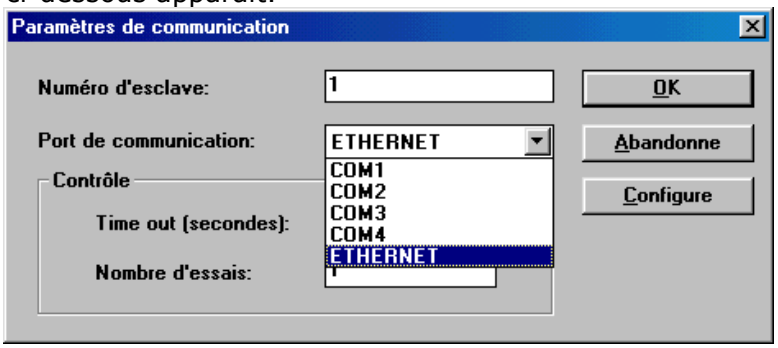
Le numéro d'esclave est remplacé par l'adresse IP du LT.
Le format de communication est imposé par la norme IEEE 802.3 (10Base-T).

Après avoir modifié le params_com qui agit sur le LT, généré et téléchargé l'application dans l'automate, il faut sélectionner le port de communication correspondant dans l'atelier ISaGRAF.

- Cliquer sur « Configurer la liaison » du menu « Test » dans la fenêtre « Programmes »

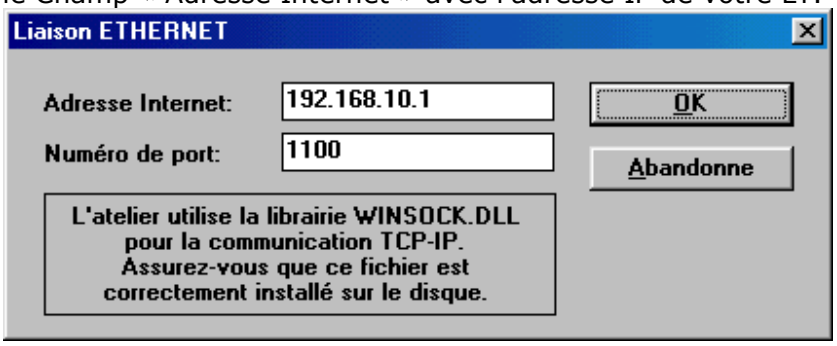


✓ La fenêtre ci-dessous apparaît.



- Sélectionner « Ethernet » dans le menu déroulant « Port de communication ». Le numéro d'esclave de ce port doit être « 1 ».
- Cliquer sur le bouton « Configure ».

- Remplir le Champ « Adresse Internet » avec l'adresse IP de votre LT.



La led Com0 bornier 1 du module CPU signale la surveillance sur la connexion avec l'atelier ISaGRAF.

- Allumée fixe lorsque l'atelier ISaGRAF est connecté sur le LT
- Clignotante lorsque la liaison est rompue après un timeout de 2 secondes.

3.2 Sauvegarde des données

3.2.1 Accès à la mémoire EEPROM

Le LT dispose de deux mémoires EEPROM. Chaque mémoire est adressable par son numéro qui est respectivement 4 et 5. La taille d'une EEPROM est de 1kmots (1024 mots). Elles sont accessibles en lecture et en écriture d'entiers (mots de 16 bits).
Exemple d'écriture puis de lecture d'un mot dans l'EEPROM n° 4 : **voir l'exemple de projet « lte2prom »**

- *Status := E2p_W(4, 2, ValeurAEcrire);*
- *ValeurLue = E2p_R(4, 2*);

Status est une variable booléenne, *ValeurAEcrire* et *ValeurLue* sont des variables entières (seuls les 16 bits de poids faible sont significatifs).
Nota : la variable entière écrite sera bornée à [-8000h..7FFFh].

E2p_R : Lit 1 mot dans une EEPROM du LT

Donnée := *E2p_R(NumE2p, AdrLecture);*

Argument	*type*	*Valeurs ou usage*
Donnée	entier	valeur entière lue [0..FFFFh] ; si erreur : 10000h
NumE2p	Entier	Numéro de l'EEPROM [4, 5]
AdrLecture	entier	[0..1023]

E2p_W : Ecrit 1 mot dans une EEPROM du LT

Status := *E2p_W(NumE2p, AdrEcriture, Donnée);*

Argument	*type*	*Valeurs ou usage*
Status	booleen	True : écriture faite ; False : écriture impossible
NumE2p	Entier	Numéro de l'EEPROM [4, 5]
AdrEcriture	entier	[0..1023]
Donnée	entier	valeur entière à écrire [0..FFFFh]

3.2.2 Variables non volatiles

Le LT dispose d'une **mémoire secourue** d'une taille de **2048 octets**.
Pour secourir une variable en cas de coupure d'alimentation du LT, il suffit de **cocher la case "non volatile"** lors de la déclaration de la variable. Il n'est pas nécessaire de configurer les paramètres d'exécution d'une application ISaGRAF comme spécifié dans le Guide Utilisateur ISaGRAF.

Sur les 2048 octets de la mémoire secourue, **1980 sont réservés à la sauvegarde des données non volatiles**. L'espace occupé par type de variable est le suivant :

- 1 octet par variable booléenne,
- 4 octets par variable entière + 4 octets pour l'ensemble des variables entières
- 5 octets par variable temporisation
- 1 octet par caractère d'une variable message + 3 octets par variable message

La seule contrainte est : si 1 variable non volatile est cochée, il est nécessaire d'en cocher une de chaque type. 4 types de variables peuvent être non volatiles : booléen, entier, temporisation et message.

Le principe de sauvegarde et récupération d'une variable secourue est le suivant :

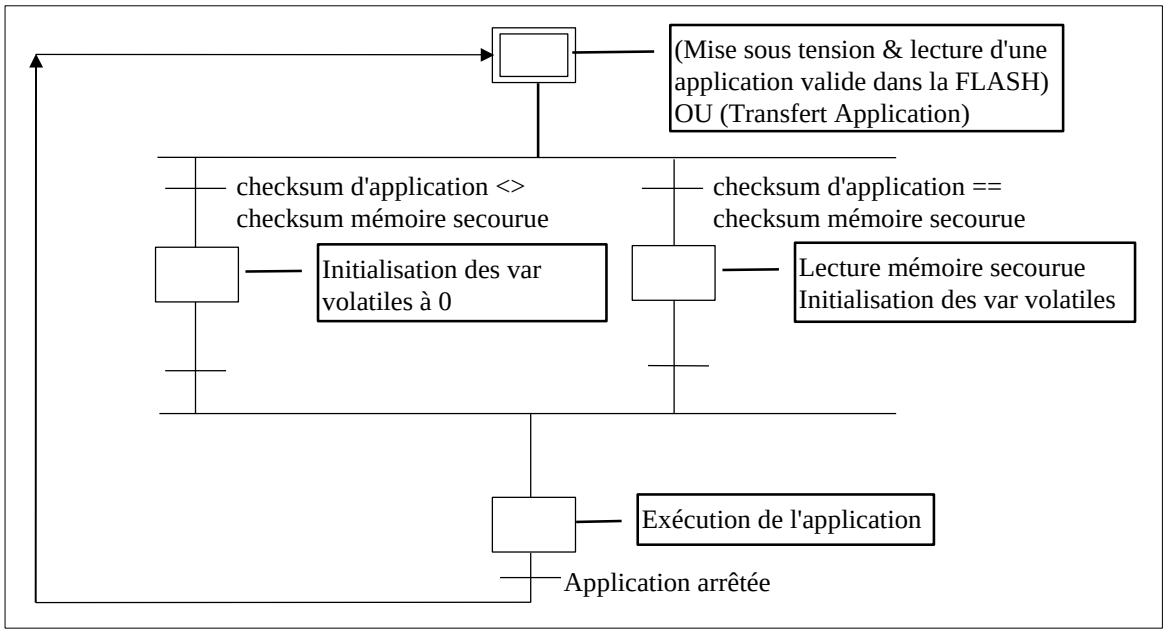


Figure 6 : Principe de traitement des variables non volatiles

3.2.3 L'horloge secourue

Le LT dispose d'une **horloge logicielle secourue**. Cette horloge **fournit la date, l'heure et le jour de la semaine**.

DayTim_O : Initialise l'accès à l'horloge sur LT

Status := DayTim_O ( );

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : erreur d'initialisation ; TRUE : initialisation correcte Une seule initialisation est nécessaire pour un projet.

DayTim_W : mise à l'heure (date et heure) de l'horloge sur LT

Status := DayTim_W (TypeInfo, chaine);

Status	booléen	FALSE : erreur d'initialisation ; TRUE : initialisation correcte Une seule initialisation est nécessaire pour un projet.
TypeInfo	Entier	0 pour une date 1 pour une heure 2 pour un jour
chaine	message	Le message date sous la forme « AAAA/MM/JJ ». Exemple '2003/12/31' Le message heure sous la forme « HH:MM:SS.CC». Exemple '23:58:10.02' Le message heure sous la forme « X» : « 0 » pour le dimanche, « 1 » pour le lundi, « 2 » pour le mardi, « 3 » pour le mercredi, « 4 » pour le jeudi, « 5 »pour le vendredi « 6 » pour le samedi

DayTime : Donne la date ou l'heure ou le jour sous forme de chaîne de message

C'est une fonction standard de l'atelier Isagraf . Voir le manuel d'Isagraf

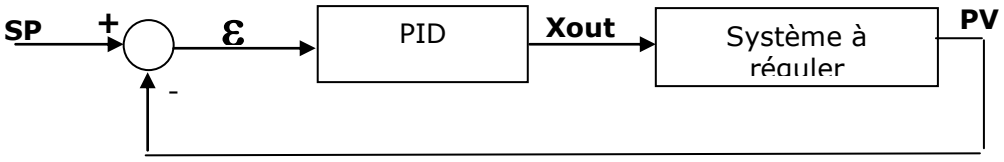
chaine := DayTime (TypeInfo);

Chaine	message	Voir manuel isagraf
Typeinfo	Entier	Voir manuel isagraf

Exemple : voir l'exemple de projet « lthtr ». Egalement, un bloc fonctionnel « horloge » permet de régler l'horloge à partir d'un maître modbus(voir la fiche d'aide de ce bloc).

3.3 Fonction de régulation : PID

Un PID est un régulateur de système fonctionnant sur le principe du retour d'état. La sortie du système est régulée en utilisant la différence entre la sortie réelle du système et l'état qu'elle devrait avoir. Ce principe est résumé par le schéma bloc ci-après.



SP = Set Point = Consigne, PV = Process Value = Mesure

$Xout(t)=Kp(\varepsilon(t)+\frac{1}{Ti}\int_0^t \varepsilon(t)d(t)+Td\frac{d\ \varepsilon\ (t)}{dt})$ en continue soit

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

3.3.1.1 Mise en oeuvre

Dans le dictionnaire / onglet « Instance BF », déclarer une instance de bloc fonctionnel de type **PID_AL**. Cette instance initialise, paramètre et rafraîchit un bloc PID.

Le traitement se décompose en trois actions: proportionnelle, intégrale, dérivée. Chaque action est réglable séparément ainsi on peut composer une régulation à partir des trois actions.

La mise en oeuvre d'un PID est réalisée par l'utilisation d'un bloc fonctionnel C Pid_AL() :

- déclarer une instance de PID : par exemple "pid1" dans le dictionnaire,
- A chaque période d'échantillonnage, appeler l'instance avec ses paramètres,
- exemple : pid1(Auto1, Pv1 ,Sp1, X01, Kp1, Ti1, Td1, Ts1, Min1, Max1);
- sortie1 est une variable numérique,
- la valeur de sortie renvoyée est dans : sortie1 := pid1.Xout;

Voir le projet de démonstration : «pid_dem.pia »

PidAl

Instance_PidAl (Mode, Pv, Sp, X0, Kp, Ti, Td, Ts, Min, Max);
Commande :=Instance_PidAl.Xout;

Argument	*type*	*Valeurs ou usage*
Mode	Booleen	TRUE = automatique ; FALSE = manuel Doit être à FALSE à l'initialisation
PV	reel	Process Value : mesure venant du process
Sp	Reel	Set point : Consigne
X0	Reel	valeur de réglage sortie pid pour le mode manuel
Kp	Reel	Gain proportionnel général à toutes les actions
Ti	Reel	constante d'intégration en secondes
Td	Reel	constante de dérivation en secondes
Ts	Temporisation	période d'échantillonnage en ms
Min	Reel	Valeur limite basse pour la sortie
Max	reel	valeur limite haute pour la sortie

Il est possible de composer un régulateur P, PI, PD, PID. Pour cela, il suffit de désactiver l'action qui n'est pas utilisée. Une action (proportionnelle, intégrale ou dérivée) est désactivée lorsque les paramètres dont elle dépend sont : Kp=1 ou Ti=0 ou Td=0.

3.3.1.2 Méthode de réglage

Le réglage du régulateur PID passe par le choix des paramètres Kp, Ti, Td. Pour déterminer les paramètres Kp, Ti, Td des méthodes d'analyse expérimentale du procédé sont possibles.

Par exemple, les spécifications typiques pour les appareils conduisant des processus chimiques ou thermiques sont les suivants:

- Ti de 3 à 1000 secondes,
- Td de 3 à 150 secondes.

Une méthode de réglage en ligne : la méthode par essai-erreur.

Le réglage en ligne peut se faire de façon empirique en utilisant une procédure qu'on peut résumer ainsi:

- mettre la régulation en place,
- enlever l'action intégrale et dérivée,
- mettre le gain Kp à une faible valeur,
- faire une petite variation de la consigne et observer la réponse du système. Comme le gain est très petit, la réponse sera très amortie,
- doubler le gain et refaire l'étape précédente. Continuer ainsi de suite jusqu'à ce que la réponse devienne oscillante. Appelons cette valeur Kpu (ultimate Kp),
- mettre Kp à (Kpu / 2),
- faire la même opération en réduisant Ti par un facteur 2, jusqu'à obtenir une réponse oscillante pour une petite variation de la consigne,
- mettre Ti au double de cette valeur,
- procéder de même pour la constante dérivée: augmenter Td jusqu'à obtenir une réponse oscillante, puis mettre Td à 1/3 de cette valeur.

4 Gestion des communications

Les blocs LUC3xxx peuvent accepter de 1 à 4 borniers de communication.

- 1er bornier COM301 Série + 3 borniers max parmi COM312 ou COM311
- 1er bornier COM303 Ethernet + 3 borniers max parmi COM312 ou COM311

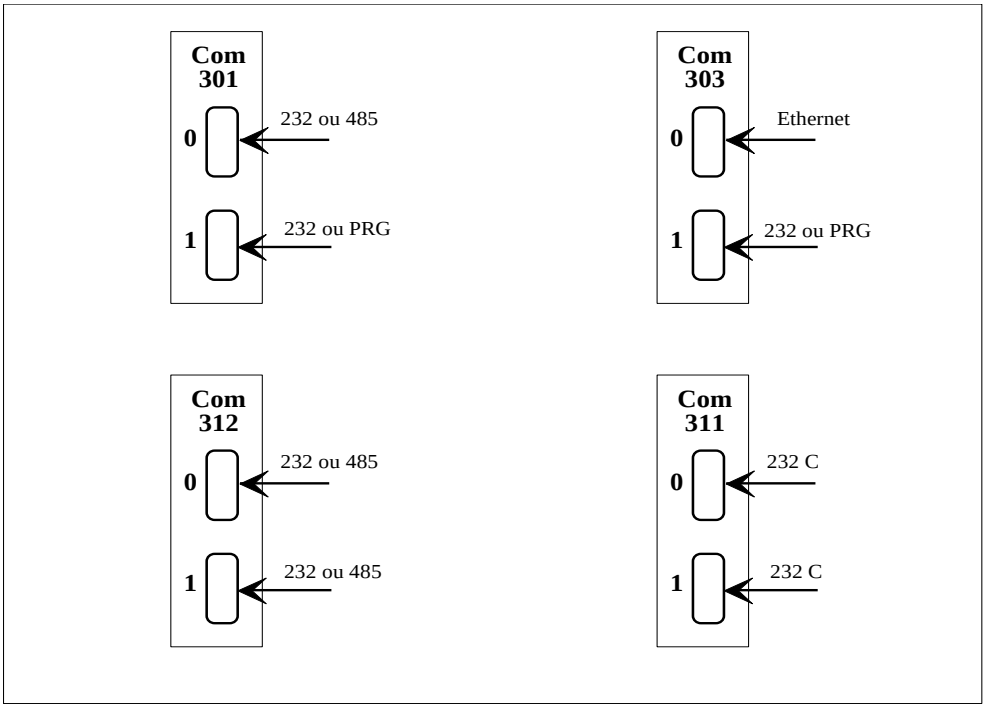


Figure 7 : borniers de communication du LT

Le **com1 du bornier 1** (com1 des borniers Com301 et Com303) **est par défaut la liaison console**. Cette liaison console peut cependant être placée sur un autre port. La liaison console permet le dialogue avec l'atelier ISaGRAF (téléchargement, débogueur...) ou l'accès aux variables du dictionnaire par un maître Modbus/Jbus (cf. paragraphe suivant). **Le com1 du bornier 1 supporte seulement une liaison RS232.** Une voie de communication est obligatoirement dédiée à la liaison console.

Le **com0 du bornier Com303 est une liaison Ethernet** et ne peut pas être configuré autrement. C'est la seule liaison Ethernet possible.

Les autres liaisons RS232 ou RS485 (com1 des Com301 et Com303, com0 et com1 des borniers Com311 et Com312) peuvent supporter soit un protocole **JBus maître ou esclave** , soit un protocole spécifique **d'émission/réception d'octets**, soit la **liaison console** si nécessaire.

Sur le bornier 1, on trouve obligatoirement un Com301 ou un Com302. Un seul Com311 peut être utilisé et se trouve obligatoirement sur le bornier 2.
Sur le LT, la communication est gérée à travers **2 couches logicielles** :
La **couche basse** est particulière à chaque liaison série. Elle stocke les octets reçus, détecte une fin de trame par dépassement du temps de silence, émet la réponse éventuelle du LT. Elle est réalisée sous interruption (spécifique à la liaison série) et transparente vis à vis du programme utilisateur.

La **couche haute** est indépendante des liaisons série. Elle analyse la trame reçue, effectue éventuellement le travail demandé par le maître et prépare la réponse à émettre. Cette couche est traitée par les fonctions utilisateur spécifiques à chaque protocole.

Protocoles disponibles :

Protocoles sur réseau RS232 et RS485 :

- Jbus Esclave,
- Jbus Maître,
- Protocole simple d'émission/réception.

Attention : le choix de la liaison série (RS232 ou RS485) dépend uniquement du câblage réalisé.

Protocoles sur réseau Ethernet :

- ModBus/TCP :
  - esclave : interrogation par 1 ou plusieurs maîtres ModBus/TCP
  - maître : interrogation d'un ou plusieurs esclaves Modbus/TCP
- SMTP : envoi de courrier électronique

4.1 Principe de communication sur les ports de communication

Les ports série RS232/485 ou RS232C supportent

- soit le **protocole Modbus/Jbus,**
- soit un protocole spécifique bâti sur **l'émission/réception d'octets**,
- soit la **liaison console** si nécessaire.

Le principe d'utilisation du protocole Modbus/Jbus sur un port du LT est le suivant :

initialiser (ou déclarer) une voie de communication Modbus/Jbus Esclave ou Maître revient à utiliser une fonction C spécifique depuis l'atelier. Cette fonction va définir les paramètres de communication sur la voie et y associer une table d'échange de n mots.

Les trames venant d'un maître ou esclave rafraîchiront cette table d'échange. Les données de chaque table peuvent être utilisées depuis l'atelier (variables du dictionnaire) par des fonctions spécifiques :

- ***Word_R()*** : lit sous forme non signée un mot d'une table vers une variable entière du dictionnaire,
- ***Word_W()*** : écrit une variable entière du dictionnaire vers un mot d'une table,
- ***Bit_R(***) : lit un bit d'une table vers une variable booléenne du dictionnaire,
- ***Bit_W()*** : écrit une variable booléenne du dictionnaire vers un bit d'une table,
- ***DWord_R()*** : lit deux mots d'une table vers une variable entière du dictionnaire,
- ***DWord_W()*** : écrit une variable entière du dictionnaire vers deux mots d'une table,
- ***WordS_R*** : lit sous forme signée un mot d'une table vers une variable entière du dictionnaire.

Exemple pour un protocole Jbus Esclave :

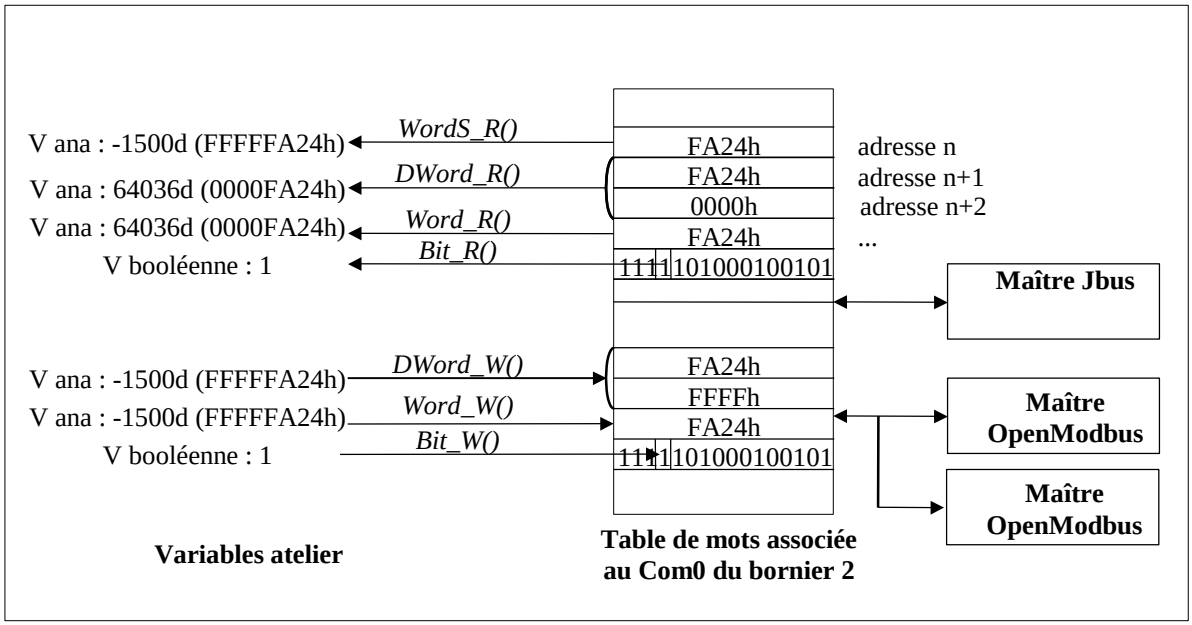


Figure 8 : Principe de communication Jbus esclave

Taille et forme des variables entières et des mots contenus dans les table d'échange :

Les variables de type entier sont codées sur 32 bits. Elles peuvent être représentées sous forme

décimale signée [-2147483648..2147483647] ou
hexadécimale non signée [00000000..FFFFFFFF].

Les variables de type mot (16 bits) contenues dans une table d'échange sont lues sous leur forme non signées par la fonction Word_R(). Pour obtenir ces variables sous leur forme signées, il faut utiliser la fonction WordS_R().*Voir les exemples présentés figures 8 et 9.*

Exemple pour un protocole Jbus Maître :

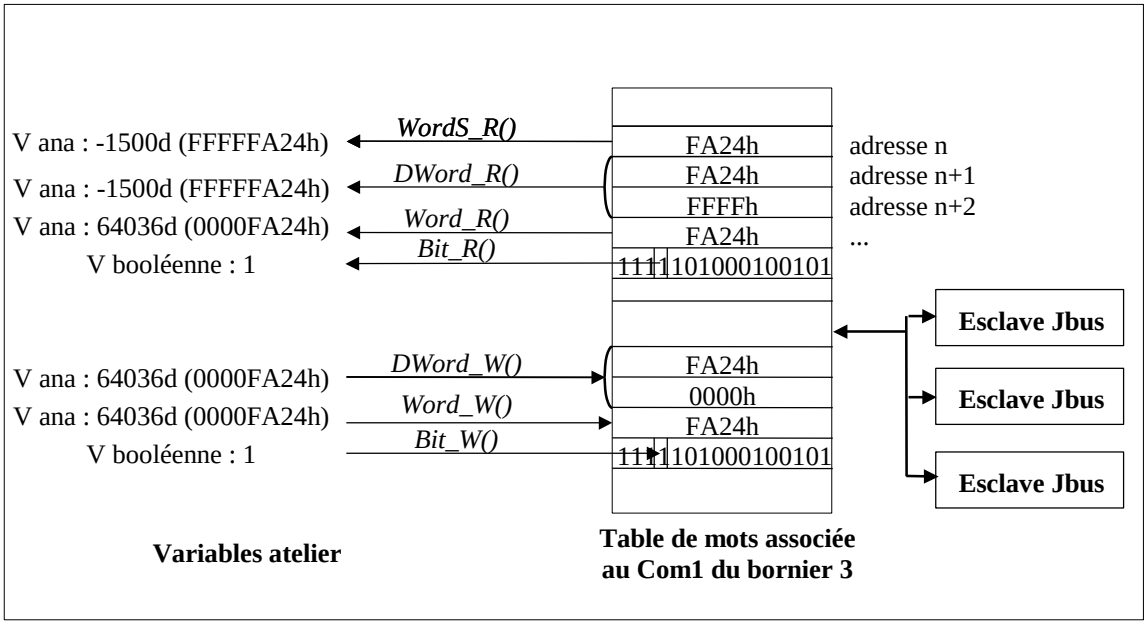


Figure 9 : principe de communication Jbus maître

Les paragraphes suivants détaillent les étapes à suivre afin de mettre en oeuvre chaque protocole.

Important : une table peut être associée à plusieurs ports de communication. Le principe est de déclarer une table par son numéro (de 1 à 7) puis d'utiliser son numéro lors de la déclaration d'un autre port. Si l'on utilise une table déjà déclarée il est obligatoire que la longueur passée en paramètre soit équivalente à la table initiale.

Les ordres Jbus/Modbus reconnus et traités par le LT ISaGRAF sont les suivants :

Fonction	Code Fonction
lecture de plusieurs bits	1 et 2 *(1)*
lecture de plusieurs mots	3 et 4 *(2)*
écriture d'un bit	5
écriture d'un mot	6
écriture de plusieurs mots	16

(1) Le LT ne fait pas la différence entre les bits de sorties et les bits d'entrées

(2) Le LT ne fait pas la différence entre les mots d'entrées et les mots de sorties

Word_R: Lit 1 mot, sous forme non signée, dans une table réseau

donnee := Word_R (NumTable, AdrMot);

Argument	*type*	*Valeurs ou usage*
Donnee	Entier	[0000 0000 .. 0000 FFFF]h ou [0..65535]d Si erreur = 0
NumTable	entier	[1..7]
AdrMot	entier	[0.."valeur max saisie lors de l'ouverture du Com - 1"]

DWord_R: Lit 2 mots dans une table réseau

donnee := DWord_R (NumTable, AdrMot);

Argument	*type*	*Valeurs ou usage*
Donnee	Entier	[0..2^32] initialisée à la valeur : 0
NumTable	entier	[1..7]
AdrMot	entier	[0.."valeur max saisie lors de l'ouverture du Com - 2"]

WordS_R: Lit 1 mot, sous forme Signée, dans une table réseau

donnee := Word_R (NumTable, AdrMot);

Argument	*type*	*Valeurs ou usage*
Donnee	Entier	Valeur signée lue (cf schéma plus haut)
NumTable	entier	[1..7]
AdrMot	entier	[0.."valeur max saisie lors de l'ouverture du Com - 1"]

Word_W: Ecrit 1 mot dans une table réseau

status := Word_W (NumTable, AdrMot, Donnee);

Argument	*type*	*Valeurs ou usage*
status	Booléen	True : écriture faite ; False : écriture impossible
Donnee	Entier	valeur entière à écrire [0..65535] L**e mot de poids fort n'est pas pris en compte.**
NumTable	entier	[1..7]
AdrMot	entier	Adresse Mot dans la table :[0.."valeur max saisie lors de l'ouverture du Com - 1"]

DWord_W: Ecrit 2 mots dans une table réseau

status := DWord_W (NumTable, AdrMot, Donnee);

Argument	*type*	*Valeurs ou usage*
status	Booléen	True : écriture faite ; False : écriture impossible
Donnee	Entier	valeur entière à écrire [0000 0000 .. FFFF FFFF]h ou [0 .. 2^32]
NumTable	entier	[1..7]
AdrMot	entier	Adresse Mot dans la table :[0.."valeur max saisie lors de l'ouverture du Com - 2"]

Exemple : Mot := 16#FF008800;
Status := DWord_W(1, 2, Mot); (* Ecriture de la variable entière Mot aux adresses 2 et 3 de la table 1 *)

Bit_R : Lit 1 bit dans une table réseau

donnee := Bit_R (NumTable, AdrMot, RangBit);

Argument	*type*	*Valeurs ou usage*
Donnee	Entier	0 => bit à 0 ; 1 =< bit à 1 ; si erreur : 0
NumTable	entier	[1..7]
AdrMot	entier	Adresse Mot dans la table :[0.."valeur max saisie lors de l'ouverture du Com - 1"]
RangBit	entier	[0..Fh]

Exemple : *Donnee := Bit_R(1, 2, 5);* (* lecture du bit 5 du mot 2 de la table 1 *)
Conseil : pour avoir un bit en valeur de retour, il suffit de faire une comparaison du résultat de la fonction par rapport à la valeur 1 : Bit := *Bit_R(1, 2, 5)=1;*

Bit_W : Ecrit 1 bit dans une table réseau

status := Bit_W (NumTable, AdrMot, RangBit, donnée);

Argument	*type*	*Valeurs ou usage*
Status	booléen	True : écriture faite ; False : écriture impossible
NumTable	entier	[1..7]
AdrMot	entier	Adresse Mot dans la table :[0.."valeur max saisie lors de l'ouverture du Com - 1"]
RangBit	entier	[0..Fh]
Donnee	booléen	Variable booléenne [TRUE, FALSE]

4.2 Les protocoles sur réseau RS232/RS485

4.2.1 Le protocole Jbus Esclave

Pour utiliser sur un port de communication le protocole Jbus Esclave, 3 fonctions sont disponibles :

JbusS_O : Ouvre une communication Jbus esclave sur un port

Status := JbusS_O (Bornier, Com, NumEsclave, NumTable, LongTable);

Argument	*type*	*Valeurs ou usage*
Status	Booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
NumEsclave	Entier	Numéro esclave entre [1..255]
NumTable	Entier	Numéro de la table entre [1..7]
LongTable	Entier	Longueur de la table en nombre de mots de 16 bits [1..4095] L'adresse Jbus de début de table est 0.

Exemple : Ouvrir une connexion JbusEsclave sur le bornier 2 com 1 . A ce port est associée une table no 3 de 100 mots accessibles en lecture/écriture. Le n° d'esclave est 12
Status := JbusS_O(2, 1, 12, 3, 100);

JbusS_C : ferme une communication Jbus esclave sur un port

Une commande "StopApplication" depuis le débogueur ferme tous les ports de communication ouverts.

Status := JbusS_C (Bornier, Com);

Argument	*Type*	*Valeurs ou usage*
Status	Booléen	FALSE : Fermeture non effectuée ; TRUE : fermeture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1

JbusS_P : Configure les paramètres de communication d'un port Jbus Esclave.

Status := JbusS_P (Bornier, Com, Vitesse, Parite, BitStop, Donnees,Acces,Silence,ARB);

Argument	type	Valeurs ou usage
Status	booléen	FALSE : Configuration non effectuée ; TRUE : Configuration correctement effectuée
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
Vitesse	entier	Voir tableau ci-dessous
Parite	Entier	Voir tableau ci-dessous
BitStop	entier	Nombre de bits de stop [1,2]
Données	entier	Nombre de bits de données [5,6,7,8]
Acces	Entier	0=Lecture/Ecriture ; 2=Lecture ; 1=Ecriture ; 3=Accès bloqué
Silence	Entier	[0..7FFFh]
ARB	entier	Adresse Réseau de Base [0..FFFFh]

Vitesse :	1 = 75 Bauds 2 = 110 Bauds 3 = 150 Bauds 14 = 200 Bauds (exclusivement sur bornier 1 com1) 4 = 300 Bauds 5 = 600 Bauds 6 = 1200 Bauds 7 = 2400 Bauds 8 = 4800 Bauds 9 = 9600 Bauds 10 = 19200 Bauds 11 = 38400 Bauds 12 = 76800 Bauds (exclusivement sur bornier 1 com1) 13 = 115000 bauds (exclusivement sur bornier 1 com1)
Parité :	0 = Pas de parité 1 = Paire 2 = Impaire 3 = Forcée à 0 4 = Forcée à 1

Exemple: voir l'exemple de projet « Ltjbuss »

4.2.1.1 Communication sur la liaison console

Le **liaison console (com1 du bornier 1 par défaut)** supporte le protocole **Modbus esclave d'ISaGRAF**. Ce protocole permet d'accéder aux variables des applications ISaGRAF par leur adresse réseau. Cette adresse réseau est définie dans le dictionnaire de l'atelier.

Seules les variables de type ***Boolean*** ou ***Analog*** sont accessibles. Les fonctions Modbus reconnues par le protocole ISaGRAF sont les suivantes :

1	Lecture n bits
3	Lecture n mots
5	Ecriture 1 bit
6	Ecriture 1 mot
16	Ecriture n mots

Attention : le protocole Modbus ISaGRAF ne gère pas les codes d'erreurs comme "adresse Modbus inconnue".

Les paramètres de communication de cette liaison console sont par défaut :

- n° esclave : 1,
- vitesse : 19200 bauds,
- parité : sans,
- données : 8 bits,
- bits stop : 1.

Ces paramètres peuvent être modifiés en utilisant le paramètre **params_com de la carte cpu3xx** (cf 3.1.2).

4.2.2 Le protocole Jbus Maître

Pour utiliser sur un port de communication le protocole Jbus Maître, 4 fonctions sont disponibles:

JbusM_O : Ouvre une communication Jbus maitre sur un port

Status := JbusS_M (Bornier, Com, NumTable, LongTable, TimeOut, NbEssais);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
NumTable	Entier	Numéro de la table entre [1..7]
LongTable	entier	Longueur de la table en nombre de mots de 16 bits [1..4095] L'adresse Jbus de début de table est 0.
TimeOut	Entier	Time out en ms [1..7FFFh]
Nbessais	entier	nombre d'essais supplémentaires sur réponse esclave absent : [0..9]

Exemple : Déclarer une connexion Jbus Maître sur le bornier 3 com 1. A ce port est associée une table d'échange no 1 de 100 mots. Le time-out est de 500ms. Le nombre de tentatives sur esclave absent est 3.
Status := JbusM_O(3, 1, 1, 100, 500, 3);

JbusM_C : ferme une communication Jbus maitre sur un port

Une commande "StopApplication" depuis le débogueur ferme tous les ports de communication ouverts.

Status := JbusM_C (Bornier, Com);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Fermeture non effectuée ; TRUE : fermeture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1

JbusM_P : Configure les paramètres de communication d'un port Jbus maître

Status := JbusM_P (Bornier, Com, Vitesse, Parité, BitStop, Données, TSA, TSAD, BusyRet);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Configuration non effectuée ; TRUE : Configuration correctement effectuée
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
Vitesse	entier	Idem **JbusS_P**
Parité	Entier	Idem **JbusS_P**
BitStop	entier	Nombre de bits de stop [1,2]
Données	entier	Nombre de bits de données [5,6,7,8]
TSA	Entier	Temps de Silence Additionnel aux 3 caractères de détection de fin d'une trame : [0..7FFFh] en millisecondes ,0 par défaut,
TSAD	Entier	Temps Silence Additionnel aux 3 caractère de fin d'émission de trame sur Diffusion : [0..7FFFh] en millisecondes , 0 par défaut
BusyRet	entier	nombre d'essais supplémentaires sur réponse esclave non prêt : [0..9] , 0 par défaut

JbusM_T : Envoi d'une trame Jbus Maître sur un port et lecture du status de communication
Cette fonction a 2 usages à utiliser successivement selon la valeur du booléen EnvoiTrame.

Status := JbusM_T (Bornier, Com, NumEsclave, CodeFonction, AdresseEsclave, Longueur, AdresseDonnees, EnvoiTrame);

Argument	type	Valeurs ou usage
Status	entier	Si Envoi trame = TRUE : • 0 : Trame non envoyée (paramètres incorrects) • 1 : Trame envoyée. Si Envoi trame = FALSE : le status donne l'échange de l'échange en cours ex : • 0 = échange en cours • 256d = échange terminé correct • 1280d = esclave absent • 770d = adresse incorrecte dans l'esclave • 772d = esclave non prêt Voir annexe : liste des codes d'erreurs
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
NumEsclave	Entier	Numéro Esclave : [1 ..255] , 0 pour les trames de diffusion
Code fonction	Entier	• 1 ou 2 : lecture n bits • 3 ou 4 : lecture n mots • 5: écriture 1 bit • 6: écriture 1 mot • 15 : écriture n bits • 16 : écriture n mots
AdresseEsclave	Entier	adresse des données dans l'esclave : [0..65535]
Longueur	Entier	Nombre d'objets à transmettre en mots ou bits selon la fonction :[1..128]
AdresseDonnees	Entier	[0.. "valeur max saisie lors de l'ouverture du Com"]
EnvoiTrame	booleen	**TRUE pour envoyer la trame** **FALSE pour tester la fin de l'échange**

Exemple : voir les exemples de projet « Ltjbusm1 » et « Ltjbusm2 »

4.2.3 Le protocole d'émission/réception de caractères

Il est possible d'implanter un protocole d'émission/réception d'octets sur les liaisons séries disponibles (hormis la liaison console). Les fonctions fournies permettent d'installer et de gérer des files d'attente de type FIFO : une en émission et une en réception. Une liaison série est gérée indifféremment en RS232 ou en RS485. Ce protocole simple permet de gérer des terminaux, des appareils avec un protocole ASCII sans avoir les contraintes temporelles liées à l'émission et à la réception d'octets. La gestion bas niveau d'un port série est effectuée par le LT sous interruption.
Après une initialisation de la liaison série, l'utilisateur peut lire ou écrire des octets dans les files d'émission et de réception. C'est le LT qui se charge, sous interruption, d'émettre ou de recevoir les octets sur la ligne.

NulPro_O : Ouvre une communication simple sur un port du LT

Status := NulPro_O (Bornier, Com, LongTabRec, LongTabEmi, Mode);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
LongTabrec	Entier	[1..1020] longueur de la file de réception (1020 = 4 messages)
LongTabEmi	Entier	[1..510] longueur de la file d'émission (510 = 2 messages)
Mode	Entier	=1 obligatoirement

NulPro_C : Ferme une communication simple sur un port du LT

Status := NulPro_C (Bornier, Com);

Status	booléen	FALSE : Fermeture non effectuée ; TRUE : Fermeture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1

NulPro_S : Ecrit dans la file d'émission sur un port du LT

Status := NulPro_S (Bornier, Com, ARM , NbCar);

Status	booléen	FALSE : Ecriture non effectuée ; TRUE : Ecriture correctement effectuée
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
ARM	Entier	Adresse Réseau du Message à émettre qui DOIT être déclarée dans le dictionnaire Isagraf et supérieure à 0 [1…65535]
Nbcar	entier	Nombre de caractères à écrire [0..255] Si = 0 alors tous les caractères du message sont envoyés

NulPro_R : Lit dans la file de réception sur un port du LT

Code := NulPro_R (Bornier, Com, ARM , NbCar);

Code	Entier	code>0 : la fonction a lu correctement « code » caractères code = 0 : lecture du message non réalisée code = -1: lecture du message réalisée mais incorrecte : le message ne permet pas de stocker tous les caractères code = -2: lecture du message réalisée mais incorrecte : un caractère n'a pu être lu dans la file de réception ou la demande de lecture de caractères porte sur un nombre de caractères supérieur à celui du message. Code = -3 : pas de caractères dans la file de réception
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
ARM	Entier	Adresse Réseau du Message de réception qui DOIT être déclarée dans le dictionnaire Isagraf et supérieure à 0 [1…65535]
Nbcar	Entier	Nombre de caractères à lire dans la file [0..255] Si = 0 alors lire tous les caractères de la file

NulPro_N : Lit le NOMBRE de caractères présents dans la file de réception sur un port du LT

NbCar := NulPro_N (Bornier, Com);

NbCar	entier	nombre de caractères dans la file de réception
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1

NulPro_P : Configure une communication simple sur un port du LT.

Status := NulPro_P (Bornier, Com, Vitesse, Parité, BitStop, Données);

Argument	type	Valeurs ou usage
Status	booléen	FALSE : Configuration non effectuée ; TRUE : Configuration correctement effectuée
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
Vitesse	entier	Idem fonction **JbusS_P**
Parité	Entier	Idem fonction **JbusS_P**
BitStop	entier	Nombre de bits de stop [1,2]
Données	entier	Nombre de bits de données [5,6,7,8]

Exemple : voir l'exemple de projet « Ltnulpro »

Exemple : Gestion d'imprimante
Les fonctions d'émission/réception d'octets sur une liaison série permettent de piloter simplement une **imprimante série**. Tout port série du LT peut être employé pour gérer une imprimante série. Les liaisons RS232C permettent en plus de gérer les signaux de contrôle comme DTR ou XON/XOFF.
Seuls les messages possédant une adresse réseau dans le dictionnaire peuvent être imprimés.
Envoi d'un message sur une imprimante connectée au com0 du bornier 2 :

- Ouvrir le port de communication : NulPro_O(2, 0, 1020, 510, 1);
- Configurer si nécessaire les paramètres de communication NulPro_P(...);
- Ecrire les caractères du message se trouvant à l'adresse réseau 20h du dictionnaire NulPro_S(2,0, 16#20, 0);

4.2.4 Le protocole d'émission/réception d'octets

Ces fonctions permettent d'écrire et de lire directement des octets. Ceci est intéressant lorsque l'on doit émettre ou recevoir au milieu d'une chaîne des octets nuls, qui avec les fonctions précédentes sont considérés comme un caractère de fin de chaîne, alors que la chaîne de données à émettre ou recevoir n'est pas terminée.

BinPro_S : Ecrit des octets dans la file d'émission sur un port de LT

Status := BinPro_S (Bornier, Com, Variable , NbOct);

Status	Booléen	FALSE : Ecriture non effectuée ; TRUE : Ecriture correctement effectuée
Bornier	Entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
Variable	Entier	Variable entière 32 bits contenant les octets à émettre
NbOct	Entier	Nombre d'octets à écrire [1..4]

BinPro_R : Lit des octets dans la file de réception sur un port de LT

Variable := BinPro_R (Bornier, Com, NbOct);

Variable	entier	Variable contenant les octets lus dans la file de réception.
Bornier	entier	1 à 4
Com	entier	0= com0 ; 1 = com1
NbOct	entier	Nombre d'octets à lire dans la file de réception [1..4]

BinPro_N : Lit le nombre d'octets présents dans la file de réception sur un port du LT

NbCar := BinPro_N (Bornier, Com);

NbCar	entier	nombre d'octets dans la file de réception
Bornier	entier	1 à 4
Com	entier	0= com0 ; 1 = com1

4.2.5 Signaux de contrôle de la liaison RS232

Hormis la liaison console par défaut (bornier 1, com1), le LT dispose de deux types de liaisons RS232 :

- liaison RS232 comprenant les signaux de contrôle RTS et CTS,
- liaison RS232C comprenant en plus les signaux de contrôle DTR, DSR et DCD.

Ces signaux peuvent être pilotés avec deux fonctions de lecture ou d'écriture :

RS232_R : Lit un signal de la liaison RS232

Status := RS232_R(Bornier, Com, TypeSignal);

Variable	entier	Variable contenant les octets lus dans la file de réception.
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1
TypeSignal	entier	0=CTS (brin 8) ; 1=DSR (brin 6) ; 2=DCD (brin 1)

RS232_W : Pilote (écrit) un signal de la liaison RS232

Status := RS232_W(Bornier, Com, TypeSignal, Etat);

Status	Booleen	TRUE= signal correctement écrit ; FALSE= erreur
Bornier	entier	1 à 4
Com	Entier	0= com0 ; 1 = com1

TypeSignal	entier	0=RTS (brin 7) ; 1=DTR (brin 4)
Etat	entier	TRUE=état haut (+10V) ; FALSE=état bas (-10V)

Utilisation dans un projet ISaGRAF : voir l'exemple de projet « Lt232sig »
Nota : en mode half-duplex le RTS est géré automatiquement (activé à l'émission puis désactivé).

Application : gestion modem
Le modem peut être sélectionné par un signal de la liaison RS232C et la procédure de connexion réalisée avec le protocole simple. Ensuite, les données émises peuvent l'être soit en protocole simple, soit en protocole Modbus/Jbus maître.

4.3 Les protocoles sur réseau Ethernet

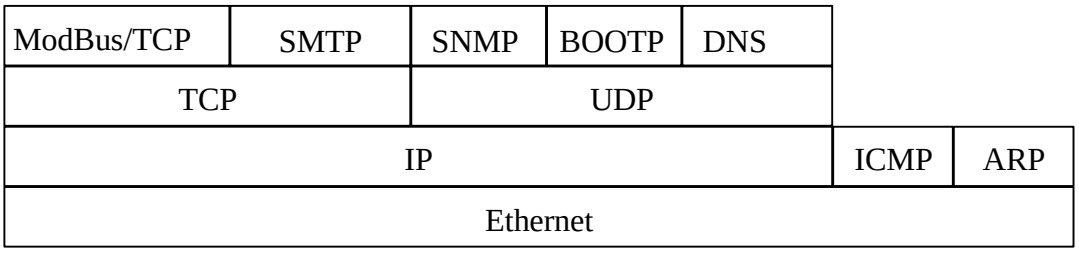
Une fois les paramètres réseau (paragraphe 3.1) correctement saisis , la liaison Ethernet supporte:

Protocoles de la couche liaison d'Internet :

- **IP** : Internet Protocol, ensemble de protocoles standards de l'industrie permettant la communication dans un environnement hétérogène : il fournit un protocole de gestion de réseau d'entreprise routable ainsi que l'accès à Internet.
- **ARP** (Address Resolution Protocol) : ARP fournit une correspondance dynamique entre une adresse IP connue et l'adresse matérielle lui correspondant.
- **ICMP** (Internet Control Message Protocol). Le protocole d'interconnexion ICMP permet aux passerelles et aux équipements d'échanger des informations relatives aux conditions anormales.

Protocoles de la couche transport d'internet :

- **TCP :** (Transmission Control Protocol) : TCP procure un service de flux d'octets orienté connexion : les applications dialoguant à travers TCP sont considérées l'une comme un serveur et l'autre comme un client : elles vont établir une connexion avant de pouvoir dialoguer.
- **UDP** : (User Datagram Protocol) : UDP est un mode non connecté : UDP n'utilise pas d'accusé de réception et ne peut donc pas garantir que les données ont bien été reçues : c'est à l'application qui utilise UDP de gérer cet aspect.



Cette suite de protocoles au dessus d'Ethernet détermine le mode de communication des ordinateurs et la procédure de connexion inter-réseaux.
Nota : la fonction ping, du protocole ICMP vous permettra de vérifier la présence d'un équipement sur le réseau en tapant directement son adresse IP ;
exemple d'utilisation : du menu Démarrer de Windows, choisissez la commande exécuter, et taper « ping 192.168.10.1 » pour vérifier la présence du LT connecté, qui porte cette adresse.
Attention : Pour un même projet, dans le cas de l'utilisation simultanée de plusieurs types de protocoles, nous vous conseillons de suivre l'exemple de projet « **ltxmulti** » fourni sur la disquette, et en particulier la méthode d'initialisation séquentielle des différents protocoles, lors de la mise sous tension du LT Ethernet.

Identification du LT sur Ethernet :
La fonction suivante permet de connaître dans le programme application l'adresse IP du LT :

AddrIP : Retourne l'adresse IP du LT
Donnée= AddrIP();

Donnée	Message	Adresse IP du LT

4.3.1 Le protocole Telnet

Ce protocole consiste à encapsuler des émissions d'octets dans des trames IP et décapsuler des réceptions d'octets de trames IP. Il utilise le mode connecté TCP. 4 connexions telnet peuvent être activées simultanément.

Cinq fonctions C sont disponibles pour gérer ce protocole :

NulTCP_O : Ouvre un canal de type telnet client sur une machine distante
Status := NulTCP_O (NumVoie, AddrIP, NumPort, Login, Password);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
NumVoie	entier	numéro de voie: 0 a 3
Addrip	message	Adresse IP ou adresse symbolique (nom DNS) de la machine distante
Numport	entier	Numéro du port sur la machine distante
Login	message	Nom d'identification pour accéder au port sur la machine distante
Password	message	Mot de passe pour accéder au port sur la machine distante

Exemple : Ouvrir un canal telnet avec l'équipement ayant l'adresse IP 192.168.239.210, sur son port 2100
Status := NulTcp_O(1,'192.168.239.210', 2100, 'root', 'admin');

NulTCP_C : Ferme un canal de type telnet client sur une machine distante.
Status := NulTCP_C (NumVoie);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Fermeture non effectuée. TRUE : Fermeture correctement effectuée
NumVoie	Entier	numero de voie: 0 a 3

NulTCP_S : Emet un message sur un canal type Telnet
Status := NulTCP_S (NumVoie, Msg, CRLF);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
NumVoie	entier	numero de voie: 0 a 3
Msg	message	Message à émettre
CRLF	booléen	False : message émis tel quel True : émission CRLF en fin de message

NulTCP_R : Lit dans la file de réception du canal telnet

Status := NulTCP_R (NumVoie, ARm);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Lecture non effectuée ; TRUE : Lecture correctement effectuée
NumVoie	entier	numero de voie: 0 a 3
ARm	entier	Adresse Réseau de la variable message dans le dictionnaire

NulTCP_N : renvoie le nombre de caractères contenus dans le buffer de réception

NbCar := NulTCP_N (NumVoie);

Argument	*type*	*Valeurs ou usage*
NbCar	entier	nombre de caractères dans le buffer de réception
NumVoie	entier	numero de voie: 0 a 3

4.3.2 Le protocole Modbus/TCP

Ce protocole consiste à encapsuler des échanges Modbus dans des trames IP. Il utilise le mode connecté TCP. Il offre les mêmes fonctionnalités que les voies "modbus" sur les liaisons asynchrones du produit. Les différences avec le protocole Modbus sur voie asynchrone sont les suivantes :

- pas de numéro d'esclave, car l'adressage s'effectue avec l'adresse IP
- utilisation du mode connecté TCP. Ce qui permet la connexion simultanée avec 4 maîtres maximum sur le réseau.
- pas de diffusion possible sur le réseau.

4.3.2.1 Le protocole Modbus/TCP Esclave

TCPMbs_O : Ouvre le port Modbus/TCP Esclave

ATTENTION : le port Modbus/TCP esclave ne peut être ouvert qu'une seule fois. Si il est ouvert une deuxième fois, le port sera refermé. **4** maîtres peuvent accéder "simultanément" à la table déclarée.

Status := TCPMbs_O (NumTable, LongTable, Acces);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
NumTable	entier	[1..7]
LongTable	entier	Longueur de la table entre [1..4095] mots (16 bits)
Acces	entier	0 = Lecture et Ecriture, 1= Lecture uniquement, 2= Ecriture uniquement

TCPMbs_S : Surveillance de la présence de 1 à 4 maîtres sur le port ModBus/TCP esclave

Status := tcpmbs_S(AdresseIP, TimeOut);

Argument	*Type*	*Valeurs ou usage*
Status	Entier	-1 : Erreur 1: Maître présent 0 : Maître absent 2 : Enregistrement de la surveillance en cours
AdresseIP,	message	adresse IP du maître à surveiller
TimeOut	entier	[0..FFFF] en millisecondes

Exemple : Surveiller la présence du maître dont l'adresse IP est contenue dans la variable « adresse_M ». Si ce maître est absent au moins 5 secondes, le status passera de 1 à 0.
Status := TCPMbS_S (adresse_M, 5000);

TCPMbs_T : Fixe la valeur du time-out de fermeture des connections TCP ouvertes par un client et non refermées, sur non activité ; par défaut la valeur de ce time-out est fixée à 5 minutes

Status := tcpmbs_t(TimeOut);

Argument	*Type*	*Valeurs ou usage*
Status	Booléen	FALSE : time out non fixé ; TRUE : time out fixé
TimeOut	Entier	[0..FFFF] en secondes

Exemple de mise en oeuvre dans un projet ISaGRAF : voir le projet « lttcpe »

4.3.2.2 Le protocole Modbus/TCP Maître

TCPMbM_O : Ouvre le port Modbus/TCP Maître sur Ethernet

ATTENTION : le port Modbus/TCP maître ne peut être ouvert que trois fois : trois connexions simultanées sont possibles

Status := TCPMbM_O (NumVoie, NumTable, LongTable, Acces);

Argument	*Type*	*Valeurs ou usage*
Status	Booléen	FALSE : Ouverture non effectuée ; TRUE : Ouverture correctement effectuée
NumVoie	Entier	[1..3] . Maximum 3 voies Modbus/TCP Maître sur un port Ethernet.
NumTable	Entier	[1..7]] Affecter l'une des 7 tables disponibles sur LT
LongTable	Entier	Longueur de la table entre [1..4095] mots (16 bits)
Acces	Entier	0 = Lecture et Ecriture, 1= Lecture uniquement, 2= Ecriture uniquement

TCPMbM_T : Envoi d'une trame Modbus TCP Maître sur une voie du port Ethernet

Status := TCPMbM_T (NumVoie, NumEsclave, CodeFonction, AddresseEsclave, Longueur, AdresseDonnees);

Argument	***type***	***Valeurs ou usage***
Status	entier	Réponse de l'esclave au standard Modbus TCP : Voir annexe : liste des codes d'erreurs
NumVoie	entier	[1..3] voie modbus TCP Maître

NumEsclave	message	Adresse IP ou adresse symbolique (Adresse DNS) du produit esclave modbus/TCP à accéder
Code fonction	Entier	1 ou 2 : lecture n bits 3 ou 4 : lecture n mots 5 : écriture 1 bit 6 : écriture 1 mot 15 : écriture n bits 16 : écriture n mots 17 : écriture puis lecture de n mots
AdresseEsclave	Entier	adresse des données dans l'esclave : [0..65535]
Longueur	Entier	Nombre d'objets à transmettre en mots ou bits selon la fonction :[1..128]
AdresseDonnees	Entier	[0..65535] adresse dans la table d'échange définie lors de l'appel à la fonction TCPMbM_O

TCPMbM_S : Envoi d'une trame Modbus TCP Maître de type **IO scanning** sur une voie du port Ethernet

Status := TCPMbM_S (NumVoie, NumEsclave, AdrEsclEcr, LongEcr, AdrDonEcr, AdrEsclLec, LongLec, AdrDonLec);

Argument	***type***	***Valeurs ou usage***
Status	entier	Réponse de l'esclave au standard Modbus TCP : Voir annexe : liste des codes d'erreurs modbus
NumVoie	entier	[1..3] voie modbus TCP Maître
NumEsclave	message	Adresse IP ou adresse symbolique (Adresse DNS) du produit esclave modbus/TCP à accéder
AdrEsclEcr	Entier	Adresse dans l'esclave des données à écrire: [0..65535]
LongEcr	Entier	Nombre d'objets à écrire en mots
AdrDonEcr	Entier	Adresse dans la table d'échange du LT, des données à écrire : [0..65535]
AdrEsclLec	Entier	Adresse dans l'esclave des données à lire: [0..65535]
LongLec	Entier	Nombre d'objets à lire en mots
AdrDonLec	Entier	Adresse dans la table d'échange du LT, des données lues : [0..65535]

Utilisation dans un projet ISaGRAF : voir l'exemple de projet « lttcpm »

TCPMbM_B : Fixe la valeur du time out de fermeture de la connexion avec l'esclave sur non activité ; par défaut, la connexion en cours reste ouverte 5 minutes, tant qu'un autre échange n'est pas demandé avec une fonction tcpmbm_t ou tcpmbm_s.

Status := TCPMbM_B(NumVoie, TimeOut);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : time out non fixé ; TRUE : time out fixé
NumVoie	entier	[1..3] voie modbus TCP Maître
TimeOut	entier	[0..FFFF] en secondes 0 : fermeture systèmatique après l'échange modbus >0 : fermeture après le timeout de non activité

4.3.3 Le protocole SNMP V1

SNMP : *Simple Network Management Protocol* : Protocole standard d'administration des hôtes, routeurs et autres appareils sur le réseau IP.
SNMP est basé sur le **modèle Agent/Station de gestion** (ou Agent/Manager). Un agent est un veilleur capable de répondre à des requêtes provenant d'une station de gestion. SNMP utilise UDP comme protocole de transport. Le numéro de port identifiant le protocole application SNMP est 161. Le LT est agent SNMP et les seules opérations supportées sont :

- Get : permet à une station de gestion d'extraire la valeur d'un objet (variable SNMP) d'un agent (LT).
- GetNext : permet à une station de gestion d'extraire la valeur de l'objet suivant (variable SNMP) d'un agent (LT).
- Set : permet à une station de gestion de modifier la valeur d'un objet (variable SNMP) d'un agent (LT).

Les variables SNMP appartenant au LT sont accessibles en lecture/écriture par le manager SNMP : voir le paragraphe 3.1 pour le paramétrage de l'adresse IP du manager.
Les variables gérées par le protocole SNMP appartiennent à une structure unique appelée MIB (Management Information Base) : la MIB est une base de données définie de façon formelle en langage ASN1 (Abstract Syntax Notation 1).

4.3.3.1 MIB II

La MIB II est une MIB standard que tous les agents possèdent. Les différents élements qu'elle comporte sont renseignés ou pas.
L'ID de la MIB II est : 1.3.6.1.2.1…
La MIB II est composée de 10 sous ensembles :
Le LT a uniquement le premier sous-ensemble renseigné : sous ensemble System(1)
System contient les informations de base pour la reconnaissance de l'agent :
Nb : la notation est la suivante : nom-objet(position, type, accès).

- **sysDescr**(1, octet string, read-only): Description de l'agent.
- **SysObjectID**(2, object identifier, read-only) : Identification : Pointe sur la branche du produit (1.3.6.1.4.1.4273 pour LAI)
- **SysUpTime**(3, Time Ticks, read-only) : Temps écoulé (en centièmes de seconde) depuis la réinitialisation.
- **SysContact**(4, octet string, read-write): Personne à contacter.
- **SysName**(5, octet string, read-write) : Nom du nœud.
- **SysLocation**(6, octet string, read-write): Emplacement physique de l'agent.
- **SysServices**(7, integer, read-only): Niveau de services possibles (entre 1 et 7 :couches OSI).

4.3.3.2 MIB LAI

Le protocole SNMP permet d'accéder aux variables du LT définies par ISaGRAF dans la MIB.
Leroy Automation a obtenu auprès de « Internet Assigned Numbers Authority – MIB » une branche identifiée dans la branche Enterprises (4273) : « LAI ».
Dans la branche LAI (4273), une sous branche est définie par le numéro d'agent, identifiant le LT (paramétré dans le câblage de la carte CPU : paramètre « Num_Agent_SNMP » : voir le paragraphe 3.1), et dans cette sous branche, sont définies les variables ISaGRAF.
2 types de variables ISaGRAF peuvent être utilisés par une station de gestion SNMP: variable de type Entier et variable de type Message.
Pour qu'une station de gestion puisse utiliser une variable du dictionnaire comme variable SNMP et donc réaliser un Get, GetNext ou un Set, il faut ajouter cette variable ISaGRAF à la MIB *(Management Information Base)* SNMP. Pour cela 2 fonctions sont disponibles, respectivement pour chaque type de variable :

Les fonctions **SnmpVA_C(), SnmpVM_C()** permettent de créer respectivement les variables SNMP de types entiers signés sur 32 bits et messages sous forme de chaîne de caractères dont on définit la longueur lors de la création de la variable :

SnmpVA_C : Définit le rang SNMP, dans la branche 1.3.6.1.4.1.4273 d'une variable de type entier du dictionnaire ISaGRAF.

Status := SnmpVA_C(Flag, OID, AddrVarAna);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Opération non effectuée ; TRUE : Opération correctement effectuée
Flag	entier	Réservé : 0 par défaut
OID	entier	[1..32767] rang de la variable Snmp permettant de l'identifier dans la branche 1.3.6.1.4.1.4273.Num_Agent.
AddrVarAna	entier	[0..FFFF] adresse réseau dictionnaire(hexa) de la variable de type entier

Exemple : La variable de type entier ayant l'adresse 16#0020 dans le dictionnaire est accessible par SNMP ; son identificateur est le suivant : *1.3.6.1.4.1.4273.2.4.0.*

Status := SnmpVA_C(0, 4, 16#20);

SnmpVM_C : Définit le rang SNMP, dans la branche 1.3.6.1.4.1.4273 d'une variable de type Message du dictionnaire ISaGRAF

Status := SnmpVM_C(Flag, OID, AddrVarMess);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Opération non effectuée ; TRUE : Opération correctement effectuée
Flag	entier	Réservé : 0 par défaut
OID	entier	[1..32767] rang de la variable Snmp permettant de l'identifier dans la branche 1.3.6.1.4.1.4273.Num_Agent.
AddrVarMess	entier	[0..FFFF] adresse réseau dictionnaire(hexa) de la variable de type Message

Chacune de ces fonctions associe l'adresse réseau du dictionnaire, qu'il faut préalablement déclarer, à un index numérique de la variable dans la MIB.
Exemple : voir le projet « LTsnmp »
Adresse de la variable SNMP « entier4 » :
iso.org.dod.internet.private.enterprises.lai.NumeroAgentSNMP.entier4
son identificateur, appelé OID, s'écrit *1.3.6.1.4.1.4273.2.4.0* (0 étant l'instance de la variable portant ce nom).
Enrichissement de la MIB du manager en langage ASN1 :

```
LAI DEFINITIONS ::= BEGIN

        IMPORTS
              enterprises
                    FROM RFC1155-SMI
          OBJECT-TYPE
            FROM RFC-1212;

lai     OBJECT IDENTIFIER ::= { enterprises 4273 }
```

```
agent       OBJECT IDENTIFIER ::= { lai 2 }

      entier4     OBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
            "test variable entière avec LT Isagraf"
          ::= { agent 4 }

      message4  OBJECT-TYPE
          SYNTAX  OCTET STRING (SIZE (0..255))
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
            "test variable message avec LT Isagraf"
          ::= { agent 14 }

END
```

4.3.3.3 Traps SNMP V1 :

4.3.3.3.1 Trap standard

Un trap de démarrage à froid est émis automatiquement vers le manager à chaque démarrage du produit.

4.3.3.3.2 Traps Spécifiques

Il est également possible d'émettre vers le Manager des traps spécifiques à l'aide de deux fonctions :

TrapInt : Envoi un trap avec une valeur associée

Status := TrapInt (Code, Value, OID);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Envoi non effectué. TRUE : Envoi correctement effectué
Code	*entier*	valeur du code du trap
Value	*entier*	Entier contenant la valeur à renvoyer
OID	*entier*	[1..32767] rang de la variable Snmp permettant de l'identifier dans la branche 1.3.6.1.4.1.4273.Num_Agent

TrapStr : Envoi un trap avec un message associé

Status := Trapstr (Code, Mess, entier OID);

Argument	*type*	*Valeurs ou usage*
Status	booléen	FALSE : Envoi non effectué. TRUE : Envoi correctement effectué
Code	*entier*	valeur du code du trap
Mess	message	Message à émettre
OID	*entier*	[1..32767] rang de la variable Snmp permettant de l'identifier dans la branche 1.3.6.1.4.1.4273.Num_Agent

Exemple : voir le projet « Ltsnmp »

4.3.4 Le protocole SMTP : envoi de courrier électronique

SMTP : *Simple Mail Tranfer Protocol :* Protocole standard sur Internet d'envoi de courriers électroniques
Le nombre de tentatives d'envois de courrier est infini tant que le LT ne parvient pas à atteindre le serveur SMTP. La tentative d'envoi de courrier est avortée si le serveur refuse d'accepter le courrier.
L'objet du courrier électronique envoyé depuis le LT est le suivant : « Message du xxxx » avec xxxx=nom du LT défini dans les paramètres de la carte UCR (voir paragraphe **Erreur ! Source du renvoi introuvable.**).
.
Les fonctions Email_I() et Email_S() permettent respectivement d'initialiser l'adresse du serveur du courrier sortant (SMTP) et d'envoyer un courrier électronique.

Email_I : Initialise l'adresse du serveur SMTP.

Cette adresse peut être une adresse IP ou le nom du serveur (ex : smtp.free.fr). **ATTENTION : Le serveur de courrier doit être unique ; par conséquent, il est interdit de l'initialiser plusieurs fois.**

Status := Email_I (Adresse);

Argument	*Type*	*Valeurs ou usage*
Status	booléen	FALSE : Opération non effectuée. TRUE : Opération correctement effectuée.
Adresse	message	Variable contenant l'adresse du serveur SMTP

Email_S : Envoie un courrier électronique via le serveur SMTP.

Cette adresse peut être une adresse IP ou le nom du serveur (ex : smtp.free.fr). **ATTENTION : Le serveur de courrier doit être unique ; par conséquent, il est interdit de l'initialiser plusieurs fois.**
Les adresses de l'expéditeur et des destinataires peuvent être sous la forme IP ou littérale (ex : 192.168.0.2 ou nom.prénom@provider.fr).

Status := EMail_S(Destinataire, Expediteur, Message Contenu**);**

Argument	*Type*	*Valeurs ou usage*
Status	booléen	FALSE : Opération non effectuée. TRUE : Opération correctement effectuée.
Destinataire	message	adresse du destinataire du courrier électronique.
Expéditeur	message	adresse de l'expéditeur du courrier électronique.
Contenu	message	corps du courrier électronique.

voir l'exemple de projet « lttemail »

Deux fonctions, ci-dessous, permettent de vérifier la présence du serveur SMTP :

US_PING : Initialise la fonction de vérification de la présence du serveur SMTP.
Status := US_PING (Adr_serv, Periode, Delai);

Argument	*Type*	*Valeurs ou usage*
Status	booléen	FALSE : Opération non effectuée. TRUE : Opération correctement effectuée.
Adr_serv	message	Variable contenant l'adresse du serveur SMTP
Periode	Entier	Période de scrutation du serveur SMTP en minutes
Delai	Entier	Délai de réponse du serveur SMTP en secondes

PING_V : Rafraîchit la fonction de vérification de la présence du serveur SMTP.
Status := PING_V ();

Argument	*Type*	*Valeurs ou usage*
Status	entier	256 : serveur présent 1280 : serveur absent

5 Les cartes d'entrées/sorties

Pour chaque carte d'entrées/sorties, une **fiche technique** (menu Aide) est disponible dans l'atelier ISaGRAF.
La **led Flt** de chaque carte d'entrées/sorties est gérée par l'électronique de la carte : si celle-ci n'a pas été rafraîchie **au bout de 1 seconde, un monostable pilote cette sortie**. Ce monostable est activé tant que la carte n'a pas été initialisée.
Nota :ce monostable est égal à 100 millisecondes sur les DIO130 et 200 millisecondes sur les DI130.

5.1 Carte DI310

La carte DI310 est composée de 32 entrées de type booléen.

5.2 Carte DI410

La carte DI410 est composée de 64 entrées de type booléen.

5.3 Carte DO310

La carte DO310 est composée de 32 sorties de type booléen.

5.4 Carte AI110

La carte AI110 est composée de 8 entrées de type entier (16 bits signés).

5.5 Carte AO121

La carte AO121 est composée de 8 sorties de type entier (16 bits signés).

5.6 Carte AI210

La carte AI210 est composée de 16 entrées de type entier (16 bits signés).

Table de conversion courant/tension <=> nombre de points	
Entrées courant non isolées	±21,1mA => ±32767 points.
Entrées tension non isolées	±10,25V => ±32767 points.
Entrées tension isolées	0 à 10,25V => 32767 points.
Entrées courant isolées	0 à 21,1mA => 32767 points.
Sorties courant	0 / 32767 points => 4 / 20mA
Sorties tension	±32767 points => ±10V

5.7 Equipement DI312

L'équipement DI312 est composé de 2 cartes :

- Inputs: 32 entrées de type booléen
- Fault : 32 défauts associés aux entrées

Les **modules DI312** ont un dispositif paramétrable de comparaison pour **contrôler la filerie** des capteurs en leurs connectant un réseau de 2 résistances : **entrées de**

sécurité. Les réseaux de résistances se ramènent à 2 types : le **montage série** (les 2 résistances en série) et le **montage parallèle** (les 2 résistances en parallèle).La résistance série est toujours présente. Dans le montage parallèle, le capteur est en série avec Rp qu'il supprime par ouverture. Dans le montage série, le capteur est en parallèle avec Rp qu'il supprime par fermeture.

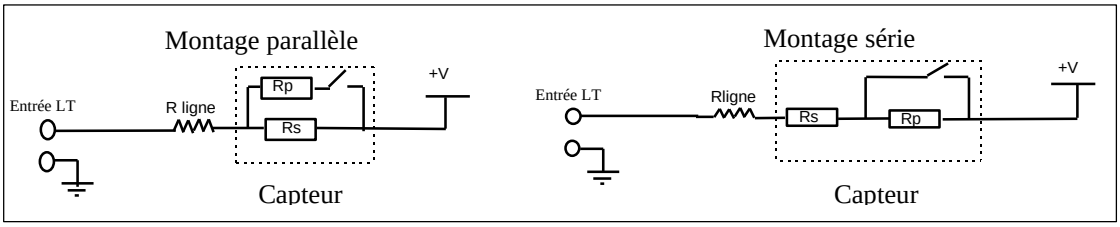


Figure 10 : Câblage des entrées de sécurité

Pour garder la généralité du paramétrage, ISaGRAF propose d'indiquer la résistance équivalente du réseau de résistances lorsque le capteur est **normalement ouvert (Rcno)** et lorsque le capteur est **normalement fermé (Rcnf**).Les valeurs des résistances sont **en OHMS**.

Montage parallèle	Montage série
Rcnf = Rs//Rp + Rligne	Rcnf = Rs + Rligne
Rcno = Rs + Rligne	Rcno = Rs+Rp + Rligne

ATTENTION : Le **paramétrage est unique** pour les valeurs de résistance d'une **carte DI312** donc le même pour toutes les voies d'un même module DI312.

Les paramètres de la carte DI312 sont les suivants :

- Masque de 32 bits du contrôle filerie des 32 entrées. Contrôle de filerie actif sur l'entrée n si le bit de rang n est à l'état 1. Par défaut, les 32 bits du masque sont à l'état 1.
- RCNO : valeur unique pour toutes les entrées.
- RCNF : valeur unique pour toutes les entrées.
- Rligne : valeur unique pour toutes les entrées.

Pour chaque voie, le bit d'état et le bit d'alarme codent 4 états possibles :

Etat Entrée LED verte	Alarme (LED rouge)	Description
0 (éteinte)	0 (éteinte)	capteur normalement ouvert
1 (allumée)	0 (éteinte)	capteur normalement fermé
0 (éteinte)	1 (allumée)	entrée non connectée ou court-circuit au 0V
1 (allumée)	1 (allumée)	court circuit au +V

Contraintes sur les résistances :

- Les résistances doivent avoir une tolérance de 1% max.
- 0.7 kΩ < **Rcno** < 22 kΩ
- Rcno induit le courant dans le dispositif de mesure : I (mA) = 22 (V) / (1+ Rcno (kΩ)) sachant que I doit être compris entre 1 mA et 9.96 mA . Si I calculé est supérieur à 9.96 mA, le saturer à 9.96 mA.
- (2.95(V) / I (mA)) < **Rcnf** (kΩ) < Rcno (kΩ) - Rligne (kΩ) - 1.95 (V) / I (mA)
- **Rligne** < 0.2 kΩ

5.8 Equipement DIO210

L'équipement DIO210 est composé de 2 cartes :

- 16 entrées de type booléen
- 8 sorties de type booléen

5.9 Equipement AIO320

L'équipement AIO320 est composé de 3 cartes :

- 8 entrées de type entier
- 16 entrées de type booléen (les entrées [1..8] et [9..16] sont respectivement dédiées aux dépassements du seuil haut et au dépassement du seuil bas des 8 entrées analogiques)
- 4 sorties de type entier

Table de conversion courant/tension <=> nombre de points sur AIO320	
Entrées courant non isolées	±20mA => ±32767 points.
Entrées tension non isolées	±10V => ±32767 points.
Sorties courant	0 / 32767 points => 4 / 20mA
Sorties tension	±32767 points => ±10V
Entrées sondes PT100	-50°C=>-500 points +350°C=>+3500 points

5.10 Equipement DI130

L'équipement DIO210 est composé de 2 cartes :

- 16 entrées de type booléen
- 16 entrées inverses de type booléen

5.11 Equipement DIO130

L'équipement DIO130 est composé de 3 cartes :

- 8 entrées de type booléen
- 16 sorties de type booléen pour commander 8 relais à commande double.
- 4 sorties de type booléen pour commander les led en face avant

6 Diagnostic et dépannage

6.1 Lecture du status des cartes CPU et d'entrées/sorties

IOStatus : Lit le status des cartes CPU et d'E/S.

Status := IOStatus(RangCarte);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[0..15]
Status	*entier*	valeur entière lue [0..FFFFh] Seuls les status de cartes correctement initialisées sont remontés Status = 0 : carte paramétrée dans l'atelier mais inaccessible sur le bus d'E/S Status = -1 : RangCarte erroné ou carte non paramétrée dans l'atelier

Exemple : *Status := IOStatus(1);* (* lecture du status de la première carte d'E/S *)

Signification des bits du status de la carte CPU :

• **bit 0**	à 1 a chaque fin de cycle
• **bit 1**	à 1 si Led WDG allumée
• **bit 2**	à 0 si défaut I/O (led I/O allumée)
• **bit 3**	à 1 si initialisations terminées (à 1 à chaque fin de cycle)
• **bit 4**	à 0 si erreur Test (led Test allumée)
• **bit 5**	à 1 si PRM détecté
• **bits 6-15**	Réservé

Signification des bits du status des cartes d'entrées/sorties :

• **bits 0-7**	**Code de la carte [0..FFh]**
• **bit 8**	à 1 si alimentation interne de la carte correcte
• **bit 9**	cf. tableau ci-dessous
• **bit 10**	FAULT : bit à 0 si carte non rafraîchie par la CPU (led FLT allumée)
• **bit 11**	cf. tableau ci-dessous
• **bits 12-15**	**Position de la carte sur le bus [0..15]**

Code des cartes d'entrées/sorties :

Carte	Status Bit 11	Status Bit 9	Code de la carte [0..FFh]
DI310	1	AI Ext	03h ou 43h
DI312	NS	AI Ext	14h
DI410	1	AI Ext	06h
DO310	Surcharge	AI Ext	05h ou 45h
DIO210	Monostable	VRel	16h
AI110	0	0	80h
AI210	0	0	81h
AO121	0	0	88h
AIO320	Monostable	0	83h
DIO130	Monostable	VRel	58h
DI130	Monostable	0	59h

Avec :

- NS : non significatif
- Al Ext : à 1 si alimentation extérieure présente sur les borniers et comprise entre Valim ±20%.
- (*) Al ext de la DI312 : Alimentation extérieure entre 24V±10%
- Surcharge : à 0 si surcharge sur une voie de sortie TOR.
- Monostable : à 1 si carte correctement rafraîchie; pour la DIO130 le fonctionnement est inversé.
- VRel : à 1 si les relais sont correctement alimentés.

IOInits : Lit le compteur d'initialisation des cartes d'E/S.

Status := IOInits(RangCarte);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Status	*entier*	valeur entière lue [0..FFFFh] Seuls les compteurs de cartes correctement initialisées sont remontés Le compteur d'init vaut 1 lors d'une initialisation correcte (unique) de la carte.

Exemple : *Status := IOInits(1);* (* lecture du compteur d'initialisation de la première carte d'E/S *)

6.2 Erreurs remontées à l'atelier

Les erreurs remontées à l'atelier en mode débug sont de deux sortes :

- **erreurs codées par l'atelier ISaGRAF** : texte expliquant l'erreur avec un numéro allant de 0 à 99. Elles sont détaillées dans le guide de l'utilisateur de l'atelier.
- **erreurs codées par LAI** : numéro allant de 100 à 255.

Les erreur codées par LAI sont :

- 100 : type de flash (non AMD512K Bottom). La taille du code TIC est alors limitée à 128Ko.
- 101 à 105 : erreur durant la sauvegarde de l'application TIC en FLASH.
  - 101 : erreur de lecture dans la FLASH.
  - 102 : erreur d'écriture dans la FLASH des infos sur les espaces réservés par une application.
  - 103 : erreur d'accès à 1 secteur de la FLASH.
  - 104 : erreur d'effacement d'un secteur de la FLASH.
  - 105 : erreur de lecture des infos sur les espaces réservés par une application.
- 110 à 112 : erreur durant la lecture de l'application TIC en FLASH.
  - 110 : erreur de lecture du type de FLASH.
  - 111 : erreur d'allocation mémoire pour les espaces réservés par une application.
  - 112 : erreur d'accès à 1 secteur de la FLASH.
- 113 : erreur sur le checksum de l'application TIC lue en FLASH.
- 120 et 121: mémoire dynamique du LT altérée, reboot automatique du LT.
- 140 à 143 : erreur dans la sauvegarde de variables non volatiles :
  - 140 : erreur : si 1 ou plusieurs variables non volatiles sont cochées, il est nécessaire d'en avoir au minimum une de chaque type.
  - 141 : erreur d'allocation mémoire LT.
  - 142 : mémoire secourue pleine.
  - 143 : erreur lecture mémoire secourue.
- 150 : erreur initialisation horloge secourue.
- 151 : erreur écriture horloge secourue.
- 160 : erreur lecture signaux de contrôle RS232.
- 161 : erreur écriture signaux de contrôle RS232.
- 170 : erreur de paramétrage de la communication du com1 du bornier 1 : paramètre params_com de la carte cpu3xx.

6.3 Dépannage de la liaison console : passage du LT en mode paramétrage

Le mode paramétrage est à utiliser pour rétablir le dialogue entre le PC et le LT ISaGRAF, dans le cas de perte de celle-ci suite au changement des paramètres de la liaison série dédiée au dialogue sur le LT ISaGRAF.
Pour un LT ISaGRAF, passer en mode paramétrage, consiste à n'exécuter que le noyau ISaGRAF sans application TIC téléchargée (Target Independant Code).
Ce mode appelé **PRM** est symbolisé par la led du même nom sur la CPU.

Pour passer en PRM :

- mettre le LT hors tension,
- relier le LT (Bornier 1, com1) et le PC (com1 ou com2) avec un câble RS232,
- exécuter le logiciel "**SSTB**" fourni sur la disquette "Librairies". Ce logiciel attend de reconnaître un LT.
- Choisir le port de communication du PC [1..4],
- Choisir le bouton "**PRM**",

- mettre le LT sous tension,
- dans sa séquence d'initialisation le LT passe en mode PRM et allume sa led PRM en fixe. Le PC affiche "**LT mis en paramètrage par défaut**"
    - Le LT ISaGRAF est alors en mode PRM.

Seule la led PRM allumée en fixe garantit un passage réussi en mode PRM.

6.4 Les leds du LT ISaGRAF

6.4.1 Led de l'alimentation

L'alimentation comporte une led verte allumée en fixe si la tension est présente et correcte.

6.4.2 Leds de la CPU

- **LED RUN verte** :
  - ◊ clignote lentement (1s) si l'application TIC (ISaGRAF) est exécutée correctement.
  - ◊ clignote rapidement (1/10s) si on est en PRM, ou l'application est arrêtée par ISaGRAF --> le noyau tourne mais n'exécute pas d'application TIC.
- **LED TEST rouge** :
  - ◊ éteinte si fonctionnement correct.
  - ◊ allumée fixe si le programme lu dans la Flash n'est pas correct ou insertion carte d'E/S non correcte.
- **LED I/O rouge** :
  - ◊ éteinte si fonctionnement correct.
  - ◊ allumée fixe si insertion carte incorrecte ou si un status carte d'E/S au moins est incorrect au cours de l'exécution du programme.
- **LED PRM verte** :
  - ◊ allumée fixe si mode PRM au boot du LT. Ne s'éteint qu'au prochain reboot du LT sans PRM.
  - ◊ éteinte sinon.
- **LED PRG verte** : non gérée par ISaGRAF.
  - ◊ allumée fixe si mode PRG au boot du LT : pont entre les broches 5 et 6 du com 1.1. Ne s'éteint qu'au prochain reboot du LT sans PRG.
  - ◊ éteinte sinon
- **LED WDG rouge** :
  - ◊ allumée fixe par défaut.
  - ◊ éteinte dès que le noyau ISaGRAF tourne ET que le programme lu dans la flash est correct. Ceci est vrai si Le WDG est géré automatiquement par le noyau (cf III.5).
- **LEDs comx verte** :
  - ◊ la led comx de la liaison console est allumée fixe dès qu'un com est correctement initialisé.

6.4.3 Pilotage des leds des cartes d'entrées/sorties

Les leds correspondant aux entrées/sorties sont rafraîchies automatiquement par le noyau. Cependant des fonctions sont proposées dans l'atelier pour piloter de manière différente l'état de ces leds (ex : inverser la logique des E/S TOR). Il existe une fonction par carte :

- *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);*
- *LedDIO21(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);*
- *LedAIO32(RangCarte, Leds_1_8);*

Attention : il faut réécrire la commande des leds à chaque cycle sinon elle est écrasée par le noyau du LT : commande des sorties TOR.

LedDI310 : Pilote les Leds d'une carte DI310.

Status := LedDI310(RangCarte, Leds_1_32);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_32	*entier*	état des 32 premières leds (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status := LedDI310(1, 16#FFFFFFFF);* (* allume toutes les leds d'une DI310 à l'emplacement 1 *)

LedDI410 : Pilote les Leds d'une carte DI410.

Status := LedDI410(RangCarte, Leds_1_32, Leds_33_64);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_32	*entier*	état des 32 premières leds (0 éteinte, 1 allumée).
Leds_33_64	*entier*	état des 32 dernières leds (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status := LedDI410(1, 16#FFFFFFFF, 16#FFFFFFFF);* (* allume toutes les leds d'une DI410 à l'emplacement 1 *)

LedDI312 : Pilote les Leds d'une carte DI312.

Status := LedDI312(RangCarte, Leds_1_32, Leds_33_64);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_32	*entier*	état des 32 leds ver'tes(0 éteinte, 1 allumée).
Leds_33_64	*entier*	état des 32 leds rouges(0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status := LedDI312(1, 16#FFFFFFFF, 16#FFFFFFFF);* (* allume toutes les leds d'une DI312 à l'emplacement 1 *)

LedDO310 : Pilote les Leds d'une carte DO310.

Status := LedDI3O0(RangCarte, Leds_1_32);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_32	*entier*	état des 32 premières leds (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status := LedDO310(1, 16#FFFFFFFF);* (* allume toutes les leds d'une DO310 à l'emplacement 1 *)

LedDIO21 : Pilote les Leds d'une carte DIO210.

Status := LedDIO21(RangCarte, Ledsi_1_16, Ledso_1_8);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Ledsi_1_16	*entier*	état des 16 leds vertes (0 éteinte, 1 allumée).
Ledso_1_8	*entier*	état des 8 leds rouges (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status LedDIO21(1, 16#FFFF, 16#FF);* (* allume toutes les leds d'une DIO210 à l'emplacement 1 *)

LedAI110 : Pilote les Leds d'une carte AI110.

Status := LedAI110(RangCarte, LedsV_1_8, LedsR_1_8);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
LedsV_1_8	*entier*	état des 8 leds vertes (0 éteinte, 1 allumée).
LedsR_1_8	*entier*	état des 8 leds rouges (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status LedAI110(1, 16#FF, 16#FF);* (* allume toutes les leds d'une AI110 à l'emplacement 1 *)

LedAI210 : Pilote les Leds d'une carte AI210.

Status := LedAI110(RangCarte, LedsV_1_16, LedsR_1_16);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
LedsV_1_16	*entier*	état des 16 leds vertes (0 éteinte, 1 allumée).
LedsR_1_16	*entier*	état des 16 leds rouges (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status LedAI210(1, 16#FF, 16#FF);* (* allume toutes les leds d'une AI210 à l'emplacement 1 *)

LedAO121 : Pilote les Leds d'une carte AO121.

Status := LedAO121(RangCarte, LedsV_1_8);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_8	*entier*	état des 8 leds vertes (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status LedAO121(1, 16#FF);* (* allume toutes les leds d'une AO121 à l'emplacement 1 *)

LedAIO32 : Pilote les Leds d'une carte AIO320.

Status := LedAIO32 (RangCarte, LedsV_1_8);

Argument	*type*	*Valeurs ou usage*
RangCarte	*entier*	[1..15]
Leds_1_8	*entier*	état des 8 leds vertes (0 éteinte, 1 allumée).
Status	*entier*	FALSE : Rafraîchissement non effectué. TRUE : Rafraîchissement correctement effectué L'état de chaque led est représenté par un bit.

Exemple : *Status LedAIO32(1, 16#FF);* (* allume toutes les leds d'une AIO320 à l'emplacement 1 *)

6.5 Identification du LT

6.5.1 Version du noyau embarqué sur la cible LT

LTVer : Lit la version du noyau embarqué sur la cible.

Donnee := LTVer ();

Argument	*type*	*Valeurs ou usage*
Donnee	*entier*	Exemple : version 1.04 : 0104h : PF = 1 et pf =4 version 1.1 : 0110h : PF = 1 et pf =10h

Exemple : *VersionLT := LTVer();*

6.5.2 Type de CPU montée sur le LT

LTType : Lit le type de CPU montée sur le LT et le type de carte fille.

Donnee := LTType ();

Argument	*type*	*Valeurs ou usage*
Donnee	*entier*	**Octet de poids faible : type de LT** 2,3 : LT 4,5 : ACS autre : cpu non identifiée **Octet de poids fort : type de carte fille** 13 : 1 seul bornier de communication : Com301 11 : Com301 + 1 bornier de communication 7 : Com301 + 2 à 3 borniers de communication 14 : Com303

Exemple : *TypeLT := LTType();*

6.5.3 Numéro de série du LT

LTSerial : Lit le numéro de série du LT

Donnee := LTSerial ();

Argument	*type*	*Valeurs ou usage*
Donnee	*entier*	Numéro de série : [0..FFFFh]

Exemple : *NumSerie := LTSerial();*

ANNEXE : Liste des codes d'erreur (status de communication) Modbus/Jbus asynchrone et modbus/TCP maître

Décimal	Hexa	Commentaire
0	**0**	**échange en cours**
256	**100**	**échange correct**
769	301	code d'exception : fonction inconnue
770	302	code d'exception : adresse incorrecte
771	303	code d'exception : donnée invalide
772	304	code d'exception : esclave non prêt
773	305	code d'exception : acquittement
774	306	code d'exception : non acquittement
775	307	code d'exception : erreur d'écriture
776	308	code d'exception : chevauchement de zones
896	380	erreur de connexion
897	381	avertissement de connexion
1024	400	numéro d'esclave incorrect
1025	401	code fonction incorrect
1026	402	longueur incorrecte
1027	403	code sous-fonction incorrect
1028	**404**	**adresse incorrecte**
1029	405	donnée incorrecte
1030	406	longueur de trame incorrecte
1280	**500**	**esclave absent**
1281	501	erreur de CRC
4096	1000	à l'émission : trame en cours
4097	1001	à l'émission : erreur de diffusion
4099	1004	à l'émission : longueur erronée
4100	1005	à l'émission : erreur d'offset
4101	1006	à l'émission : erreur de fonction
4102	1007	à l'émission : erreur de sous-fonction
4103	1008	à l'émission : erreur de donnée sous-fonction
4104	1009	à l'émission : erreur de stockage

en gras les codes d'erreur, les status de communication les plus fréquemment rencontrés.

TABLE DES FIGURES

FIGURE 1 : ARCHITECTURE ISAGRAF SUR LE LT 7
FIGURE 2 : RESSOURCES MATÉRIELLES 9
FIGURE 3 : CYCLE DE TRAITEMENTS DU LT ISAGRAF 10
FIGURE 4 : PRINCIPE DE CONFIGURATION DES CARTES CPU ET D'ENTRÉES/SORTIES 14
FIGURE 5 : PRINCIPE D'EXÉCUTION D'UN LT ISAGRAF 15
FIGURE 6 : PRINCIPE DE TRAITEMENT DES VARIABLES NON VOLATILES 21
FIGURE 7 : BORNIERS DE COMMUNICATION DU LT 25
FIGURE 8 : PRINCIPE DE COMMUNICATION JBUS ESCLAVE 27
FIGURE 9 : PRINCIPE DE COMMUNICATION JBUS MAÎTRE 27
FIGURE 10 : CÂBLAGE DES ENTRÉES DE SÉCURITÉ 48

INDEX DES FONCTIONS C

AddrIP 37
BinPro_N 35
BinPro_R 35
BinPro_S 35
Bit_R 28
Bit_W 28
DayTim_O 20
DayTim_W 21
DayTime 21
DWord_R 27
DWord_W 28
E2p_R 19
E2p_W 19
Email_I 44
Email_S 44
IOInits 50
IOStatus 49
JbusM_C 31
JbusM_O 31
JbusM_P 31
JbusM_T 32
JbusS_C 29
JbusS_O 29
JbusS_P 29
LedAI110 55
LedAI210 55
LedAIO32 56
LedAO121 55
LedDI310 54
LedDI312 54
LedDI410 54
LedDIO21 55
LedDO310 54
LTSerial 56
LTType 56
LTVer 56
NulPro_C 33
NulPro_N 34
NulPro_O 33
NulPro_P 34
NulPro_R 34
NulPro_S 33
NulTCP_C 37
NulTCP_N 38
NulTCP_O 37
NulTCP_R 38
NulTCP_S 37
PidAl 23
PING_V 45
RS232_R 35
RS232_W 35
SnmpVA_C 42
SnmpVM_C 42
TCPMbM_B 40
TCPMbM_O 39
TCPMbM_S 40
TCPMbM_T 39
TCPMbs_O 38
TCPMbs_S 39
TCPMbs_t 39
TrapInt 43
TrapStr 43
US_PING 45
Word_R 27
Word_W 28
WordS_R 27