

# **Programming with LadderWORK**

Copyright 1999 - 2000 MicroSHADOW Research (uS)  
All rights reserved

English Version  
Document Update : 26 January 2000



## COPYRIGHT

*Copyright 1999 MicroSHADOW Research, all rights reserved.*

The program and its related documentation are copyrighted. The user may not copy the program or its documentation except for your own back-up purposes and load the program into the computer as part of executing the program. All other copies of the program and its documentation are in violation of this agreement. No part of the manual may be photocopied or reproduced in any form or electronic medias without prior written authorization from MicroSHADOW Research.

## SINGLE USER LICENSE AGREEMENT

**License :** The user has the non-exclusive right to use the enclosed program on a single CPU or personal computer. The user may physically transfer the program from a PC to another provided that the program is used on only CPU at time. The user may not electronically transfer the program from one PC to another over a network. The user may not distribute copies of the program or related documentation to others. The user may not modify or translate the program or related documentation without the prior written consent of MicroSHADOW Research. The user may not use, copy, modify or transfer the program or documentation, or any copy thereof, or permit anyone else to do so, except as expressly provided in this agreement.

**Back-up and transfer :** The user may make one ( 1 ) copy of the program solely for own back-up purposes. The user must reproduce and include the copyright notice on the back-up copy. Transfer of program and license to another party may only be made after written approval from MicroSHADOW Research, provided the other party agrees to the terms and conditions of this agreement and completes and returns a product registration form to MicroSHADOW Research. If the user transfer the program, at the same time he must transfer the documentation and back-up copies or transfer the documentation and destroy the back-up copies.

**Upgrades :** LadderWORK software is sold without warranty. Furthermore MicroSHADOW Research reserve the right to make changes to any products herein to improve reliability, functionality and performance. The user can download software upgrades directly from the web according to its hardware key programmed version. Normally the hardware key, sold with the software, can accept major software number equal to the purchased issue. Free demo version available from the web has to be intended for evaluation only and gives no warranty of any kind.

**Media defects Warranty :** MicroSHADOW Research warrants to the original licensee that the media ( diskettes, cd-rom or others ) which the program is recorded be free from defects in materials and workmanship under normal use and a free substituting service is available for a period of 90 days from the date of delivery, accompanied by a copy of the purchase invoice.

## DISCLOSURE

The informations contained in the manuals ( traditional or electronic medias ) are subject to change without prior notice. MicroSHADOW Research assumes no responsibility or liability for any errors or inaccuracies ( technical or editorial ). The program is supplied "as is" without warranty of any kind. The entire risk as to quality and performance of the program is charged to the user.

MicroSHADOW Research assumes no liability for any damages resulting from defects, software bugs or design's solutions of LadderWORK software. It makes no warranty of any kind ( express, implied or statutory ) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third part rights.

In no event shall MicroSHADOW Research, its directors, officers, engineers, employees or agents be liable for any indirect, special, incidental or consequential damages ( including damages for loss of profits, loss of business, loss of use or data, interruption of business an the like ), even if MicroSHADOW Research has been advised of the possibility of such damages arising from any defects or error in the documentation or software.

**LadderWORK software can't be used for medical instrumentation, life-support equipment or military systems.**

Any referement to Corporations, names and data used in the screen's reproduction are purely casuals and it has to be intended as tutorial purpose only. Product and corporate names appearing in the documentation may or may not be registered trademarks or copyrights of their respective companies, and are used only for identification or explanation and to the owner's benefit, without intent to infringe.

**Miscellaneous :** This license agreement shall be governed by the laws of ITALY and shall insure to the benefit of MicroSHADOW Research.

## Index

<b>SECTION 1 - Introduction to PLC.....</b>	<b>10</b>
1.1 - PLC Overview.....	10
1.2 - Run-time enviroment.....	10
1.3 - Input read and output write scan.....	10
1.4 - Housekeeping.....	10
1.5 - Program Scan.....	10
1.6 - CEI / IEC 1131-3 Programming languages.....	11
1.7 - Ladder Logic.....	12
1.8 - Ladder logic's elements.....	13
<b>SECTION 2 - The Integrated Dev. Enviroment.....</b>	<b>15</b>
2.1 - System Requirements.....	16
2.2 - Installing the software.....	17
2.3 - Integrated enviroment overview.....	18
2.4 - Menu .....	20
<b>2.5 - File Menu .....</b>	<b>21</b>
2.6 - Edit Menu.....	23
2.7 Build Menu.....	24
2.8 - View Menu.....	25
2.9 - Zoom Menu.....	26
<b>2.10 - Options Menu .....</b>	<b>27</b>
2.11 - Upgrade Dialog.....	30
2.12 - Directories.....	31
2.13 - Colors Dialog.....	32
2.14 - Fonts Dialog.....	33
2.15 - The reference grid.....	34
2.16 - The components bar.....	35
2.17 - Drawing schematics.....	36
2.17.1 - Placing components.....	36
2.17.2 - Moving components.....	36
2.17.3 - Deleting components.....	36
2.17.4 - Connecting components.....	36
2.17.5 - Set components property.....	37
2.18 - Saving & loading projects.....	38
2.18.1 - Saving projects.....	38
2.18.2 - Loading projects.....	38
2.19 - Starting a new project.....	39
2.20 - Building the code.....	39
2.21 - Uploading the code.....	41
2.22 - Running the PLC.....	41
2.23 - Profiles.....	42

2.24 - Printing schematics.....	43
<b>SECTION 3 - Tutorial.....</b>	<b>47</b>
<b>3 - Tutorial .....</b>	<b>48</b>
3.1 - About Tutorial.....	48
3.1.1 - Organisation of this chapter.....	48
3.1.2 - Conventions used in this section.....	49
3.1.3 - Related documentation.....	49
3.2 - Introduction to LadderWORK.....	50
3.2.1 Paragraph Information.....	50
3.2.2 - What Is LadderWORK?.....	50
3.2.3 - How does LadderWORK work?.....	50
3.2.4 - Installing LadderWORK.....	51
3.2.5 - Virtual Circuits (VCs).....	52
3.2.6 - IDE toolbar.....	54
3.2.7 - Principal tricks and tips in connecting VC devices.....	55
3.3 - Working with VCs – elementary components.....	59
3.3.1 - VC Clock Generator device.....	59
3.3.2 - VC Delay device.....	60
3.3.3 - AND, OR, NOT Ports.....	62
3.3.4 - VC Flip Flop D Device.....	64
3.3.5 - VC Debounce Device.....	65
3.3.6 - VC Counter Device.....	67
3.3.7 - VC Relay Device.....	68
3.3.8 - VC Threshold Device.....	71
<b>SECTION 4 - LIBRARY.....</b>	<b>74</b>
ADD.....	75
AD_CONV.....	76
AND.....	77
ASSIGN.....	78
BIT.....	79
CLOCK.....	80
CONST.....	81
COUNTER.....	82
CTD.....	84
CTU.....	85
CTUD.....	86
DEBOUNCE.....	87
DEC1-8.....	88
DELAY.....	89
DISPLAY.....	91
DIV.....	93
EINPUT.....	94
ENCINPUT.....	95
EOUTPUT.....	96
FFD.....	97
FIELD.....	98

FIFO.....	99
F_TRIG.....	101
IDENT.....	102
INPUT.....	105
IPIN.....	106
KEYBCTRL.....	107
KEYBOARD.....	108
LIFO.....	109
LIMIT .....	110
MAX.....	111
MBCONF.....	112
MBIN.....	113
MBOUT.....	114
MBSLAVE.....	115
MIN .....	116
MOD .....	117
MUL.....	118
MUX.....	119
NCINPUT.....	120
NOT .....	121
OPIN.....	122
OR .....	123
OUTPUT.....	124
PWMOUT.....	125
QTP_DSPY.....	126
QTP_KEYB.....	127
READVAR.....	128
RELAY.....	129
ROL .....	130
ROR.....	131
RS.....	132
R_TRIG.....	133
SEL.....	134
SEMA.....	135
SEVENSEG .....	136
SHL.....	137
SHR .....	138
SR.....	139
SUB .....	140
THRESHLD.....	141

TMI .....	142
TOF .....	143
TON .....	144
TP .....	145
TSQ .....	146
USER1.....	147
USER2.....	148
USER3.....	149
<b>SECTION 5 - MATHEMATICAL EXPRESSIONS.....</b>	<b>151</b>
5.1 - Entering formulas using function block format.....	152
<b>SECTION 6 - Interfacing with assembler.....</b>	<b>153</b>
6.1 - Interfacing with assembler using user functions.....	154
6.1.1 - User function calling conventions.....	154
6.2 - Generic embedded board adapting.....	157
6.2.1 - Generic LADDER standard I/O functions.....	158
6.2.2 - Custom I/O software example.....	159
6.2.3 - Serial I/O hook functions.....	161
6.2.4 - Panel & Keyboard handling functions.....	162
6.3 - USASM51 Assembler language reference.....	165
6.3.1 - Assembler directives summary.....	165
6.3.2 - Assembler operators summary.....	165
6.3.3 - Literals.....	165
6.3.4 - 8051 microprocessor instruction set.....	166
<b>SECTION 7 - Technical notes .....</b>	<b>167</b>
7.1 - MODBUS® PROTOCOL.....	168
7.1.1 - Read Boolean ( Function Code 01 ).....	169
7.1.2 - Read Numeric ( Function Code 03 ).....	170
7.1.3 - Set Single Boolean ( Function Code 05 ).....	171
7.1.4 - Set Single Numeric ( Function Code 06 ).....	172
7.1.5 - Remote terminal unit (RTU) framing.....	173
7.2 - Timing resolution.....	174
7.3 - Memory models.....	174
7.4 - Flow process.....	175
7.5 - Logical links.....	177
<b>SECTION 8 - Error messages.....</b>	<b>178</b>
SFD0200.....	179
SFD0201.....	179
SFD0202.....	179
SFD0203.....	180
SFD0204.....	180
SFD0205.....	180
SFD0206.....	181
SFD0207.....	181
<b>APPENDIX.....</b>	<b>182</b>

**Appendix A - Function block cross reference.....184**

## List of figures

Figure 1 - Integrated enviroment overview	18
Figure 2 - The Menu	20
Figure 3 - File Menu	21
Figure 6 - The View Menu	25
Figure 7 - The toolbar view sub-menu	25
Figure 8 - The Zoon Menu	26
Figure 9 - The Options Menu	27
Figure 10 - Compiler Options , Code generator Folder	27
Figure 11 - Compiler Options , Linker Folder	28
Figure 12 - Compiler Options , Files Folder	29
Figure 13 - Upgrade Dialog	30
Figure 14 - Directories Dialog	31
Figure 15- The Color Dialog	32
Figure 16 - Fonts Dialog	33
Figure 17 - The components bar	35
Figure 18 - Configuring property	37
Figure 19 - A sample property dialog	37
Figure 20 - Save project dialog	38
Figure 21 - Load project dialog	38
Figure 22 - The device select dialog	39
Figure 23 - The upload dialog	41
Figure 24 - Profiles	42
Figure 25 - Page setup : Pagination	43
Figure 26 - Page Setup : Margin	43
Figure 27 - Page Setup : Project summary	44
Figure 28 - Hardware configuration	51
Figure 29 - IDE representation	52
Figure 30 - Rampdemo project	53
Figure 31 - IDE Toolbar	54
Figure 32 - Rampdemo.pjn compiling result	54
Figure 33 - Error Warning Box	55
Figure 34 - Not connected reset input	56
Figure 35 - Always check the correct label assignment of input devices	57
Figure 36 - Always check the correct label assignment of OUTPUT devices	58
Figure 37 - tutor1.pjn VC layout	59
Figure 38 - tutor2.pjn VC layout	60
Figure 39 - Time representation of Input and Output signals in tutor2.pjn	61
Figure 40 - tutor3.pjn schematic layout	62
Figure 41 - Time representation of Input and Output signal in tutor3.pjn	63
Figure 42 - Use of Flip Flop D Device	64
Figure 43 - tutor5.pjn Elevator Controller	65
Figure 44 - Time representation of input and output signal of the Debounce device	66
Figure 45 - tutor6.pjn Water Distributor	67
Figure 46 - Flashing light Control Block	68
Figure 47 - Crossing Sensor Block	69
Figure 48 - Opening Logical Block	69
Figure 49 - Closing Logical Block	70
Figure 50 - Timer Block	70
Figure 51 - Fuel Distributor Input Block	71
Figure 52 - Output and Compare Block of Fuel Distributor	72
Figure 53 - COUNTER Timing Diagram	83
Figure 54 - Delay Timing	90
Figure 55 - Expression sample	152
Figure 56 - Flow process diagram	176
Figure 57 - Logical links	177





## **SECTION 1 - Introduction to PLC**

### **1.1 - PLC Overview**

A Programmable Logic Controller (PLC) is an industrial computer specialized for real time applications. PLCs are integrated systems containing a processor, power supply, input modules, output modules and special purpose modules. Input modules interface with plant equipment and convert the field signals to logic levels for the processor to read. The processor uses these inputs to perform control functions based on application software. Output modules transmit the signals via an interface with the plant equipment. In addition there are special modules for communication with other computers, specialized dedicated functions, and conventional high-level language co- processors. Ladder logic will be used in the examples for the purpose of exposition.

### **1.2 - Run-time enviroment**

The PLC runtime environment is firmware which provides the operating systemservices and library functions associated with the PLC. In the RUN mode,the PLC firmware runs as real-time executive which processes the (LadderLogic) instructions that have been loaded into the program RAM area. Theprogram runs in a continuous loop which consists of the following major phases:

- Input read and output write scan
- Housekeeping
- Program scan (logic solve).

### **1.3 - Input Read and Output Write Scan**

During the input/output (I/O) scan, the processor updates its internal input and output buffers with data being read from or written to I/O devices. Local I/O devices are the input and output cardsresiding in the same physical chassis as the PLC processor. Remote I/Odevices reside external to this chassis and are communicated with the processor's peer communications interface port.

I/O data for input and output cards used in the application are maintained in input and output image tables. Typically the PLC will organize the I/O image tables. This means that the inputs which are present will read into an area in memory. The program will write into another area of memory which is used to represent the outputs. It can be said that the input image table is representative of 'how the inputs are perceived', and the output image table is 'the desired state' of the outputs. These tables are accessible to the Ladder Logic program as data files. During the I/O scan, data read from input cards are placed in appropriate locations in the input image table. At the same time, output data written to the output image table by the Ladder Logic are transferred to the appropriate output cards.

### **1.4 - Housekeeping**

Following the I/O scan, the PLC performs what is referred to as "housekeeping."This portion of the program cycle is used by the real-time executive to maintain and update its own internal state.

### **1.5 - Program Scan**

The program scan is the portion of the overall cycle where Ladder Logic instructions of the user's application software are executed. Here, the embedded firmware program operates on the portions of memory (RAM) that have been loaded previously with the application software from the binary file.

Program files contain the actual instructions to be executed. Data files are used to maintain program variables and other data structures required by the logic. It is the responsibility of the firmware program to properly decode and execute instructions in the program files. The program must also properly update the contents of the data files based on these instructions

## 1.6 - CEI / IEC 1131-3 Programming Languages

CEI / IEC 1131, Part 3, specifies the semantics and syntax of a unified suite of five programming languages for PLCs. These languages can be grouped into two categories: textual and graphical. Graphical languages are based upon graphical representation, that is, lines, boxes and text to represent specific relations among inputs and outputs. Appropriate quantities flow along lines between elements according to well defined rules. There are three graphical programming languages: *Ladder Logic*, *Sequential Function Charts*, and *Functional Block Diagrams*. Ladder logic is the most common of PLC languages and is discussed in the following section of this appendix. Sequential function charts can be used as a simple language, but their most important function is to integrate modules written in other languages into a single higher level program. Function Block Diagrams uses block diagrams to interconnect the function.

Textual languages consist of a defined set of characters, rules for combining characters with one another to form words or other expressions, and the assignment of meaning to some of the words or expressions. There are two textual languages defined in the standard: *Instruction List (IL)* and *Structured Text (ST)*. IL is a very low-level language, and may be considered as a standard Assembly Language for PLCs. Structured text is a textual programming language using assignment, sub-program control, and selection and iteration statements to represent the application program for a PLC. ST, as distinguished from IL, is the high-level text-based language for PLCs. Much of its syntax is derived from Pascal.

The models of execution, program organization, and variable handling of all CEI / IEC 1131-3 languages are based on a common hierarchical architecture consisting of *Configurations*, *Resources*, *Tasks*, and *Programs*.

Configurations are the highest level at which Global variables and Directly Represented Variables may be shared and accessed. A Configuration may often correspond to a single PLC unit, but certain types of PLC Network Architectures as well as multi-processor PLCs also meet this definition. A Configuration is composed of one or more Resources. Each Resource corresponds to a signal processing function, its associated man-machine interface functions, and sensor-actuator interface functions. A single-processor stand-alone PLC Configuration would have but a single Resource. A Configuration composed of a dozen processors capable of sharing the defined global variables and directly represented variables, on the other hand, would have 12 Resources associated with it.

Each Resource may have Global Variables (which are limited in scope to that Resource), zero or more defined Tasks, and Programs associated with those Tasks. Tasks may be defined as periodic, in which case they are defined with a specified periodicity, or as non-periodic, in which case they are executed upon the detection of the rising edge of a boolean variable. Tasks may also be assigned an execution priority. Tasks may also be scheduled pre-emptively, or non-preemptively. A Program not assigned to a Task will execute repetitively at the lowest priority level.

Programs in the IEC 1131-3 architecture begin with a variable declaration section, followed by the program statements themselves. Programs may contain calls to Functions, which return a single value, or Function Blocks, which return one or more values. Each Program, Function, or Function Block is written in one of the five IEC 1131-3 defined languages. Multi-language programming is accomplished by calling a Function or Function Block written in one language from a Program, Function, or Function Block written in another.

Variables in IEC 1131-3 languages may be either Symbolic Variables, or Directly Represented Variables. Directly Represented Variables provide a standard nomenclature for direct access to specific addresses of the I/O and internal memory map of the PLC. All Directly Represented Variables begin with a '%' character, followed by a location prefix, a size prefix, and then a sequence of numbers to indicate the actual location. Some examples of these and their meanings:

- %QX75 Output (Q) Bit (X) number 75
- %IW215 Input (I) Word (W) number 215
- %MW48 Internal (M) Word (W) number 48

The precise meaning of the hierarchy of location numbers is not defined, so it is possible to have constructs like the following, taken from an actual PLC architecture:

%MW3.23.8.12.2.4,

which corresponds in this particular case to, from right to left, the 4th Internal 16 Bit Integer Word located in Module subsection 2 of module 12 of Rack 8, of the unit at drop 23 of MODBUS® Network 3. Directly Represented Variables do not have to be declared. Their use is legal only in Programs and Configurations.

Symbolic Variables do need to be declared. In the case where Symbolic Variables refer to actual input and output points, the declaration assigns them to these points by associating them with the appropriate Directly Represented Variables. Symbolic Variables that do not refer to I/O points need not be assigned a Directly Represented Variable - the IEC 1131-3 language system will assign these an address at compile time.

## **1.7 - Ladder Logic**

Ladder Logic is an instruction set to provide services of real time, I/O, user interface, and similar services. These services are associated with the special requirements of the PLC applications domain. Because Ladder Logic is targeted toward special applications, it provides features that are compatible with real-time control application requirements. These features, when used correctly and appropriately can contribute to the safe operation of the program.

The origin of Ladder Logic is the Relay Ladder Logic notation which was first introduced to represent combinations of contacts and coils of relays using specific notation. These combinations implemented logical functions (e.g., AND or OR). The introduction of PLCs transformed Ladder Logic from a hardware design notation to a high level language, specialized for process and logic control. The Ladder Logic language, in the case of the PLC, is not the traditional limited Ladder Logic implemented with relays, but an advanced language supported by the numerical capabilities of the processor, while the Ladder Logic notation serves only a graphical user interface. Ladder Logic supports all types of programming structures from advanced subroutines, parameter passing, loops, mathematical functions, proportional plus integral plus derivative (PID) controllers, I/O calls, timers, and any other features of a high-level language. Although much changed from their original purpose and implementation, current forms of Ladder Logic are still similar to relay logic, allowing electrical engineering personnel who have traditionally have been in charge of factory automation to review and understand the code. This is an important advantage throughout the development process.

Ladder Logic is not a formally defined programming language. Each manufacturer has its own variation of Ladder Logic. In addition, many of the features associated with programming the PLC are not features of Ladder Logic itself, but the programming environment, the "shell," and the firmware mentioned above. The variety of ladder logic implementations is due to the strong coupling between software and hardware dictated by the requirements of the industrial control applications domain.

## 1.8 - Ladder logic's elements

Ladder Logic programs consist of the following types of elements

- *Power rails:* Ladder Logic networks are delimited on the left and right by vertical lines known as left and right power rails, respectively. The right power rail may be explicit or implied.
- *Link elements and states:* Links indicate power flow in the rungs of the Ladder Logic diagram. A link element may be horizontal or vertical. A horizontal link transmits the state of the element to its immediate left to the element to its immediate right. The state of an element can be either ON or OFF. A vertical link intersects with one or more horizontal links on each side and its state is the inclusive OR of the states of the horizontal links on its left. This state is transmitted to all horizontal links attached to the vertical link on its right.
- *Contacts:* A contact is an element which imparts a state to the horizontal link on its right side equal to the AND of the state of the horizontal link on its left side with an appropriate function. A contact does not modify the value of the associated Boolean variable.
- *Coils:* A coil copies the state of the link on its left to the link on its right without modification, and stores an appropriate function of the state or transition of the left link into the associated Boolean variable.
- *Functions and function blocks:* A function is a program unit which, when executed, yields exactly one result. A function block may yield more than one result. Internal variables of a function or function block are not accessible to users of the function. In Ladder Logic, at least one Boolean input and one Boolean output is shown for each function block to allow power to flow through the block.



---

## **The Integrated Development Enviroment (IDE)**

## System Requirements

What you need to install **LadderWORK** software is listed below.

- Personal computer class Pentium 133 or higher
- 32 Mbyte of RAM
- 20 Mbyte of HARD DISK space
- A CENTRONICS standard parallel port for hardware key inserting
- A RS-232C standard serial port for PLC communication
- A CD-ROM drive ( For CD-ROM Version )



---

## 2.2 - Installing the software

Software installation procedure is different depending on which version you have.

### Installing from CD-ROM

Open the computer resources icon and select your CD-ROM drive. Run the program called **Install** present on the root directory of the CD-ROM and follow the instructions of the installation program.

### Installing from FLOPPY

Insert the distribution floppy disk named **Install Disk 1** in your floppy drive, open the computer resources icon from the desktop window of your computer and select your floppy drive. On the root directory of your floppy drive select and run the program called **install** and follow the instructions of the installation program.

### Installing from Self-Extracting file.

Run the program called **LADDERWORK.EXE** and follow the instructions of the installation program.

### Installing the PROTECTION-KEY

If you have a full functional version of LadderWORK software you have to install the PROTECTION-KEY on your PC. The PROTECTION-KEY must be installed on the PC parallel port. The PROTECTION-KEY is transparent so you can attach your printer connector at the opposite end of the key.

### Launch the software

LadderWORK software can be launched using the **START** menu of **Windows 95/98** operating system, selecting the entry **LadderWORK** and choosing the program **LadderWORK**. With **Windows 95/98** operating system you have the possibility to create short-cut icons on your desktop screen.

## 2.3 - Integrated enviroment overview

The picture below, represents the apperance of the program LadderWORK on your computer. LadderWORK has an integrated enviroment feature, allowing you to draw schematics, compile programs and upload code to PLC always working on the same window. The integrated enviroments are composed by several parts described below.

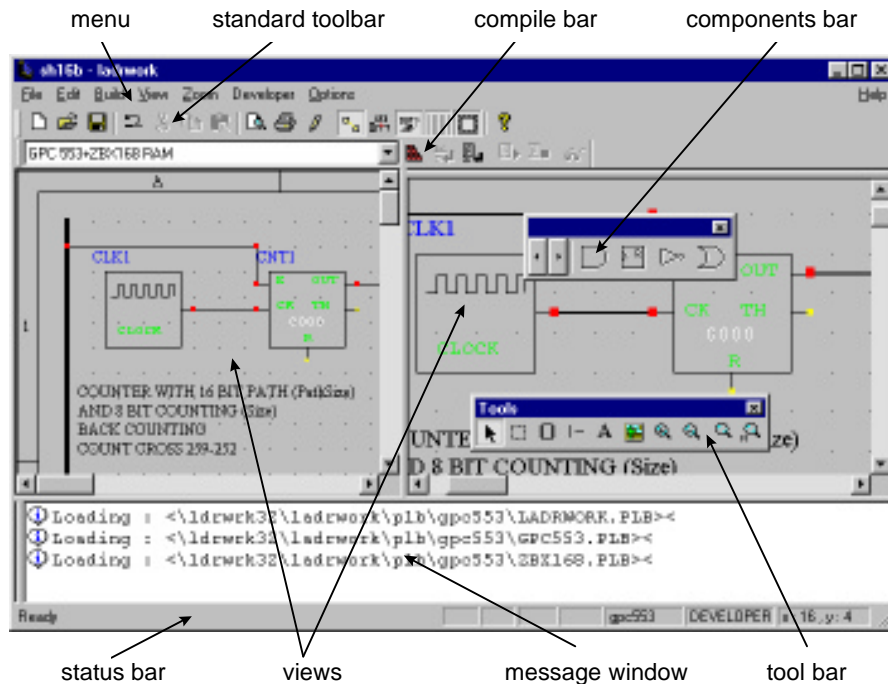


Figure 1 - Integrated enviroment overview

### menu

The menu is a Windows standard menu.

### standard toolbar

The **standard toolbar** is the bar where you can find the traditional windows commands like new, load, save and so on.

### tool bar

The **tool bar** is the window where you can find the tools for placing components, wires, text and bitmaps. In this bar also you can find the commands for change zoom factors in the current view.

### compile bar

The **compile bar** contains the command to build the code and other commands for PLC controlling like upload, run and stop. Also in the **compile bar** you can find the selection list for the profiles defined in the project. This feature allows you to change the configuration easily without accessing the configuration dialog.

### components bar

This floating window contains the components that you can place in the schematic. The components are grouped for functionality. The active group can be changed through the up and down button present at the top of the window. For more information see using components bar .

### status bar

---

This window contains information about the software status. Also on this window you can find context-relative help about commands and components

**message window**

The message window gives information about the compiling process. Many are the messages that the software can show to the user. The messages can be divided in three classes : Informations, Warnings and Errors. A message is always escorted by a icon that identify the class. To get more information about a particular message simply double click the message on the window.

## 2.4 - Menu

LadderWORK menu, is a Windows standard menu. Use the mouse to select a menu entry and click with the mouse left button to select an option. A shortcut key may be present near the menu command. Use shortcuts to entry commands using the keyboard.

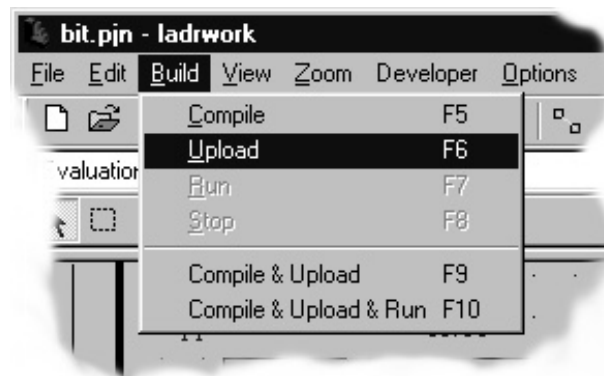


Figure2 - The Menu

---

## 2.5 - File Menu

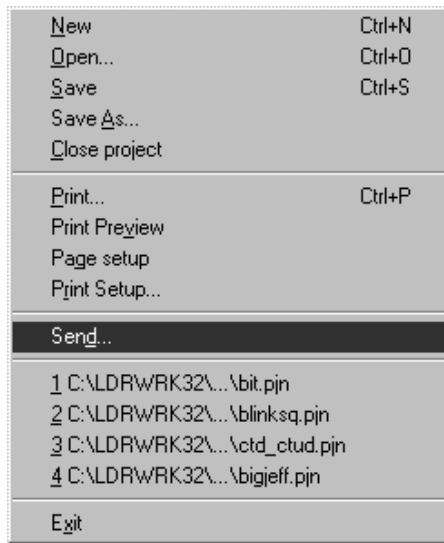


Figure3 - File Menu

The File menu gives you the possibility to operate with the Windows standard command line New, Open and Save.

On the bottom of this menu there is a list of files also called *recent list* which keep track of the file most recently used

### **New Project**

Select New Project from the File Menu to start a new project file. All the information from the previous file, if any, will be erased. If the previous project was edited but not saved, you will be warned and you will have the option to save the file before the information is erased. The default file name will be **noname.pjn**. Note that LadderWORK will not let you save a file with the name **noname.pjn** so you should use Save As and give it a name.

### **Open Project**

Select Open to read in an existing project file with **.pjn** extension.

### **Save Project**

Save will save the current project file without asking for a file name unless a file name has not yet been established. The default file name, **noname.pjn**, is not considered to be a designated file name and you will be asked to provide a file name if you select Save from the File Menu.

### **Save As**

Save As gives you the option of specifying a file name for the current help file before saving. As described in Save, above, Save As will be called if a file name has not yet been assigned to the current help file. You can also use Save As to save a project file under a different file name than the name that is currently assigned to the file. The file extension for a LadderWORK project file must be **.PJN**.

### **Close Project**

The Close Project command closes the current project.

### **Print**

With the Print command you can print your schematic on the printer. A lot of print options can be changed using the Page Setup command.

### **Print Preview**

The Print Preview command allows you to view the schematic in the same way it will be printed on the printer.

### **Page Setup**

This command allow to change a lot of print options. See

### **Print Setup**

This command opens the standard printer setup dialog.

**Exit**

Exits the program.

---

## 2.6 - Edit Menu



Figure 4 - Edit Menu

The Edit Menu gives you the standard edit capability like Copy, Cut and Paste

### **Undo (Shortcut : CTRL+Z)**

The Undo command restores the schematic to the state before the last action you performed. The Undo queue is ten operations deep.

### **Cut**

With the Cut command you can move in the private clipboard a single object or a group of objects from the schematic. To move a single object select the object pressing the left button of the mouse when you are above the object then perform a Cut command. To move a group of object first select the objects with the **select** tool then use the Cut command.

### **Copy**

The Copy command allows you to copy a single object or a group of objects in the private clipboard. To copy a single object first select the object pressing the left button of the mouse when you are above the object then perform a Copy Command. To copy a group of object first select the objects with the **select** tool then use the Copy command.

### **Paste**

The Paste command pastes a single object or a group of objects in the **worksheet** previously copied with the Copy

### **Delete**

With the Delete command you can delete a single object or a group of objects from the schematic. To delete a single object select the object pressing the left button of the mouse when you are above the object then perform a Delete command. To delete a group of objects first select the objects with the **select** tool then use the Delete command.

### **Notes**

The Notes command opens a little editor. This is useful to write information about the project you are building.

## 2.7 - Build Menu

Compile	F5
Upload	F6
Run	F7
Stop	F8
Compile & Upload	F9
Compile & Upload & Run	F10

Figure 5 - Build Menu

**NOTE : Remember that the BUILD COMMANDS are available only if you have assigned a name for your project.**

### **Compile (Shortcut : F5)**

With the Compile command you activate the Compile Process. See the section **Building the code** for further information.

### **Upload (Shortcut : F6)**

Many PLC models supported by **LadderWORK** software have the remote control feature, so you can upload the code directly from the integrated environment. If your PLC doesn't have this feature you shouldn't use this command.

### **Run (Shortcut : F7)**

If your PLC supports a remote control feature you can run the PLC simply by executing this command.

### **Stop (Shortcut : F8)**

If your PLC supports a remote control feature you can stop the PLC simply by executing this command.

### **Compile & Upload (Shortcut : F9)**

The Compile & Upload command execute in sequence the Compile and the Upload sessions.

### **Compile & Upload (Shortcut : F10)**

This command execute in sequence the Compile, Upload and Run sessions.



---

## 2.8 View Menu



Figure 6 - The View Menu

### Toolbar

This menu entry opens the following sub-menu.

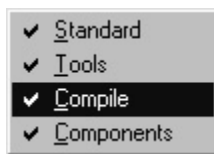


Figure 7 - The toolbar view sub-menu

### Standard

This command enables/hides the **Windows standard tool bar**. The standard toolbar is the bar where you can find the traditional windows commands like new, load, save and so on.

### Tools

This menu entry enables/hides the **tool bar** window. The **tool bar** is the window where you can find the tools for placing components, wires, text and bitmaps. In this bar you can also find the commands for change zoom factors in the current view.

### Compile

Checking/Unchecking this entry will enable/disable the **compile bar** window. The **compile bar** contains the command to build the code and other commands for PLC controlling like upload, run and stop. Also in the **compile bar** you can find the selection list for the profiles defined in the project. This feature allows you to change the configuration easily without accessing the configuration dialog.

### Components

Checking/Unchecking this entry will enable/disable the **components bar** window. This window contains the components that you can place in the schematic. The components are grouped for functionality. The active group can be changed through the up and down button present at the top of the window.

### Grid

The grid check enables/hides the grid in the **worksheet**.

### Reference Grid

Checking/Unchecking this entry will enable/disable the **reference grid** in the **worksheet**.

### Watch

If your software version handles the variable watching this menu entry enables/hides the **watch window**.

### Options

This command will open a dialog that allows you to enable/disable information in the schematic. Information includes LOGICAL\_LINKS, plugs, nodes and others.

## 2.9 - Zoom Menu

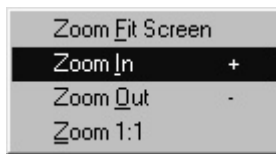


Figure8 - The Zoon Menu

### **Zoom fit screen**

This command will fit your schematic in the current view.

### **Zoom in (Shorcut : +)**

This command performs a Zoom-in in the current view.

### **Zoom out (Shortcut : -)**

This command performs a Zoom-out in the current view.

### **Zoom 1:1**

This command restores the current view to the default zoom factor .

---

## 2.10 Options Menu

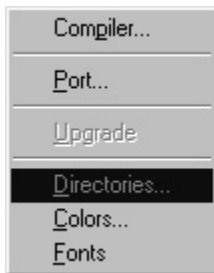


Figure9 - The Options Menu

### Compiler

The compiler dialog changes according to the software version and the target processor. In this section we discuss distinctly the dialogs for all the software version.

---

#### 8051 MICROPROCESSOR

---

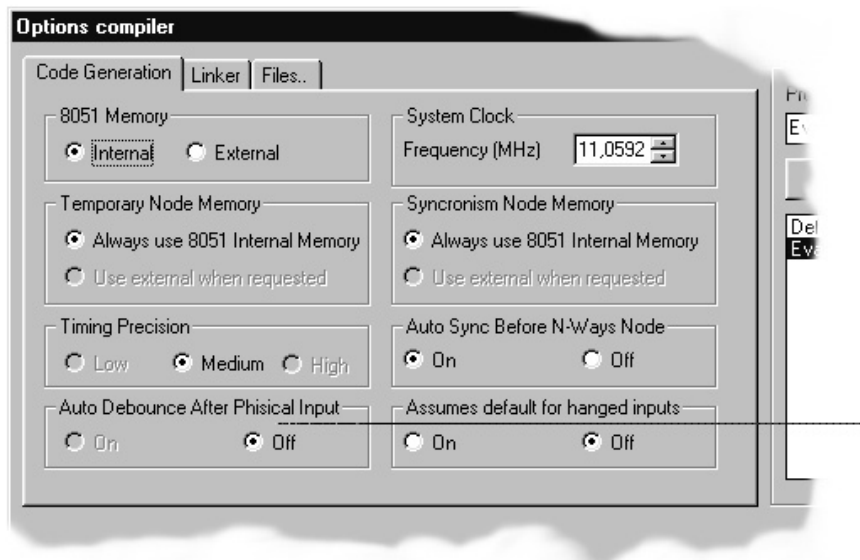


Figure10 - Compiler Options , Code generator Folder

### 8051 Memory

This option affects the use of the ram memory of a 8051 system. 8051 microprocessor can address two distinct data memory areas : internal or external. For further information about this argument see MEMORY MODELS .

### Temporary node memory

The compiler uses some ram cells to keep temporary node information. With this option you can select if temporary node information should be kept in internal or external ram.

### Synchronism node memory

When the compiler detects a n-way node condition, a temporary node is created. In this way the entire tree preceeding the node is evaluated once. With this option you can decide where this sync node will be stored.

### Timing precision

This parameter changes the resolution of the timer used as base-timer. Higher precision means a more detailed timing definition but more frequent hardware interruptions. Lower precision means a less detailed timing resolution but a lower interrupt overcharge.

#### Auto sync before n-way node

As said for the **Synchronism node memory** this flag enable/disable the sync. node optimizing.

#### Auto debounce after phisical input

When this flag is enabled, the compiler automatically places a debounce component after a physical input.

#### Assume default value for hanged inputs

When this flag is enabled, the compiler automatically places the default values for the not connected inputs. If the flag is disabled and there are no connected inputs the compiler produces error messages.



Figure 11 - Compiler Options , Linker Folder

#### Code offset

This parameter, expressed in exadecimal, changes the code start offset in the linking phase.

#### Code limit

This parameter, expressed in exadecimal, set the maximum address for the microprocessor code memory.

#### Internal data offset

This parameter, expressed in exadecimal, changes the internal data start offset. This value must be equal or greater than 10H to avoid conflits with LadderWORK kernel.

#### Internal data limit

This parameter, expressed in exadecimal, set the maximum address for the microprocessor internal ram memory.

#### External data offset

This parameter, expressed in exadecimal, changes the external data start offset.

#### External data limit

This parameter, expressed in exadecimal, set the maximum address for the microprocessor external ram memory.

#### Stack size

---

This parameter, expressed in exadecimal, changes the size of the stack area. This area always is allocated in the microprocessor internal ram. Normally for 8051 microprocessor the parameters must be placed at 10H - 24H.

### Jump optimizing

When this flag is active the linker optimize jump instructions.

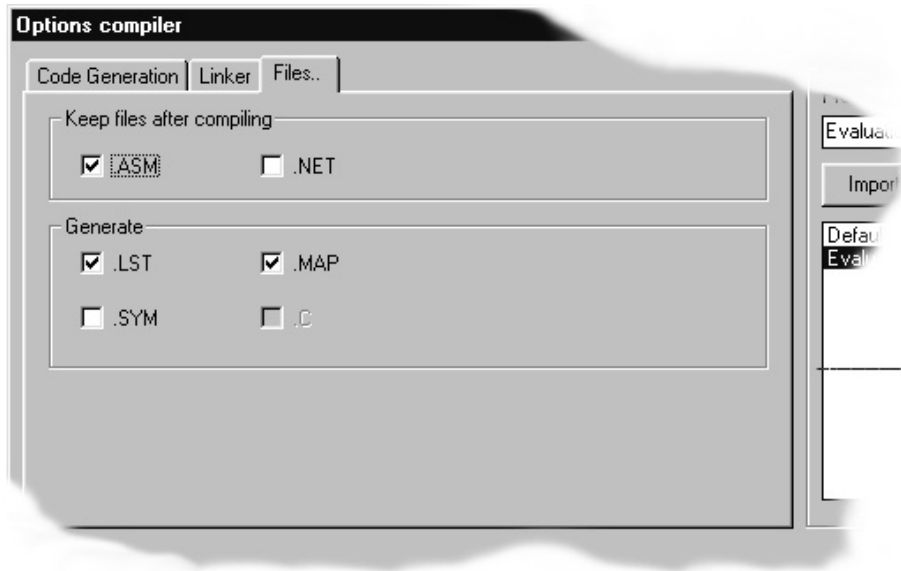


Figure12 - Compiler Options , Files Folder

### Keep files after compiling

These checkboxes select which files must be kept after the compiling process. If these checkboxes are unchecked at the end of the process the file will be erased.

### Generate

These checkboxes affect the generation of some files like Listing, Symbols, Map, and C-Code generation.

## 2.11 - Upgrade Dialog

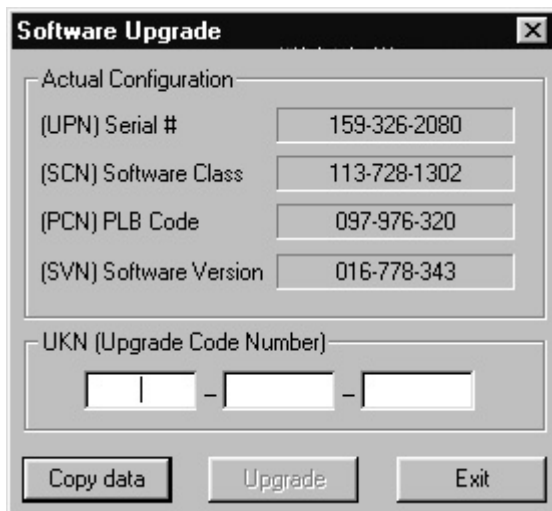
The image shows a 'Software Upgrade' dialog box with a title bar containing a close button. It is divided into two main sections. The first section, 'Actual Configuration', contains four rows of text labels and input fields: '(UPN) Serial #' with the value '159-326-2080', '(SCN) Software Class' with '113-728-1302', '(PCN) PLB Code' with '097-976-320', and '(SVN) Software Version' with '016-778-343'. The second section, 'UKN (Upgrade Code Number)', contains three empty input fields separated by hyphens. At the bottom of the dialog are three buttons: 'Copy data', 'Upgrade', and 'Exit'.

Figure 13 - Upgrade Dialog

### Upgrade

**Note :** *The upgrade menu entry can appear only if you close the current project.*

With this command you can perform a remote-upgrade session. When this command is executed the following dialog will appear.

There are four distinct codes, the Unique Product Number ( UPN ), the Software Class Number ( SCN ), the PLB compatibility number ( PCN ) and the Software Version Number ( SVN ). The User may buy a cheaper version of the software and upgrade the software at distance without having to change the protection key. The User must supply all the codes directly to MicroSHADOW Research or to a local dealer through fax or e-mail. MicroSHADOW Research will replay with the UKN code . When the UKN code is inserted in the dialog, the software upgrade will take effect. Upgrading the software allow to extend the number of components available in the components bar or using a newer version of the software.

---

## 2.12 Directories

Normally the directories are initialized during software installing. If you need to change paths you can access the directories option only starting a new project. The information about directories will be stored in the .INI file and are globals for all the projects. The unique exception is the PLB Files path. This path is modified according to the PLC model you are working on. The PLC model string is added at the end of the specified path so the compiler can access the rights PLB for the selected device in the sub-directory.

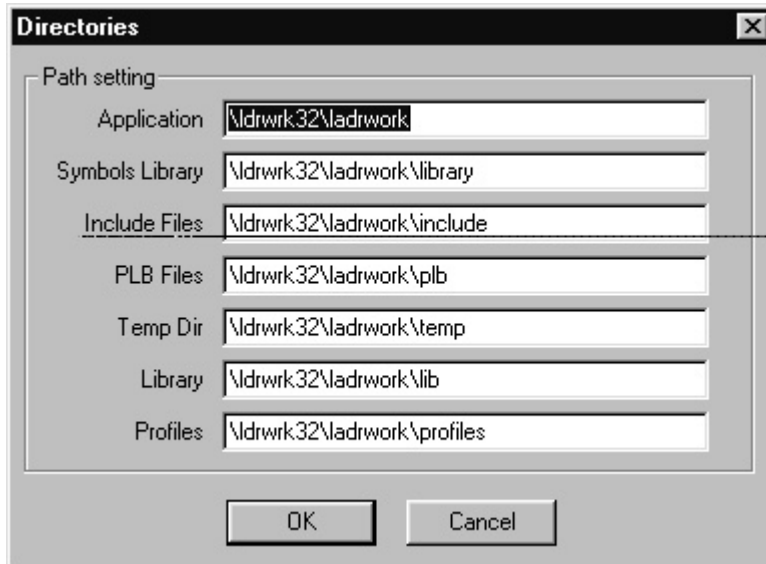


Figure 14 - Directories Dialog

### Application

This path tells the system where the system files can be found. System files includes DLL(s), INI file, help files and other.

### Symbols Library

The symbols library path tells the system where to localize the symbol's library module (.SLI files) . The .SLI files contains information about the components. Information includes vectorial representations, component's geometry, plugs and others.

### Include Files

This path tells the assembler where to localize the include files (.INC) .

### PLB Files

This path tells the compiler where the PLB files can be found. PLB files are relative to the devices you are working on, so as said above, a string is added to this path to access the correct sub-directory.

### Temp dir

Temporary files are stored in this directory.

### Library

This path tells the linker where the library files are located. Library files include the run-time kernel procedures.

### Profiles

The profiles are stored in this directory.

## 2.13 - Colors Dialog

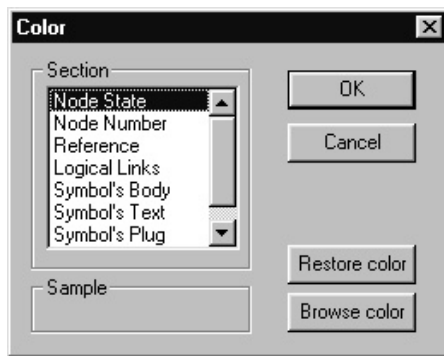


Figure 15- The Color Dialog

With the colours dialog you can change the appearance of some elements. In order you can modify the colour of the following elements

### Node State

Every time you place wires between components, the system places a particular symbol at the extremity of the link. The symbol placed by the system indicate if the wire is connected right or hanged. The colour of this symbol can be modified through this option.

### Node Number

The software automatically numbers the nodes of the schematic. With this option you can change the colour of this text.

### Reference

Every component placed in the sheet has it's own REFERENCE code. This entry allows you to modify the colour of the REFERENCE text.

### Logical Links

If a component is linked with another a straight line appears as connection between the two or more components. The colour of this line is changed using this option.

### Symbols's Body

The body of the symbol ( or component ) is formed by graphic primitives like lines and circles. This entry changes the colour of this primitives.

### Symbols's Text

The body of the symbol can contain text elements. This entry changes the colour of this text.

### Symbols's Plug

Around the bound of a component there are the net plugs. A net plug is a point where you can connect another component or a wire. The colour of the shape that indicate the plug (normally a yellow square box ) can be changed using this option.

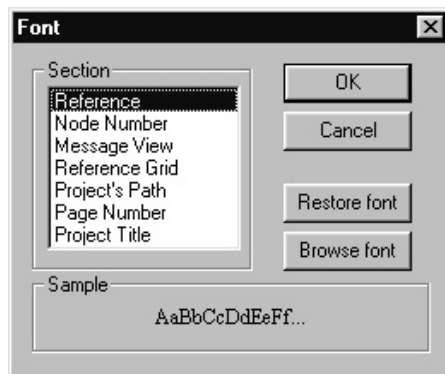
### Symbols's Show Plugs

This entry is out-of-use for now. In the future versions this feature will be used some information for the in-line simulator.



---

## 2.14 - Fonts Dialog



With the fonts dialog you can change the appearance of some elements. In order you can modify the font typeface and the size of it's text of the following elements.

Figure 16 - Fonts Dialog

### Reference

Every component placed in the sheet has an own REFERENCE code. This entry modify the font of the REFERENCE text.

### Node Number

The software automatically numbers the nodes of the schematic. With this option you can change the font of this text.

### Message view

The font of the text displayed in the Message window can be changed through this option.

### Reference grid

All around the sheet there is a Reference grid. With this option you can change the font of the letters and numbers present in that grid.

### Project's Path

During printing, this option modify the font of the file name present at the bottom of the sheet.

### Page Number

During printing, this option modify the font of the page number text present at the bottom of the sheet.

### Project Title

During printing, this option modify the font of project summary box present in the bottom-right corner of the sheet.

## **2.15 - The reference grid**

All around the sheet, a particular element called REFERENCE GRID can be activated. The reference grid is useful to locate a particular component or element inside the sheet. With this method you can easily locate a particular component. For example, if someone asks you to modify the count range of the counter CNT5, this could be difficult to find CNT5 in a large sheet. But if someone asks you to coordinate B-4 you can easily reach the position of that component by simply scrolling the view.

---

## 2.16 - The components bar

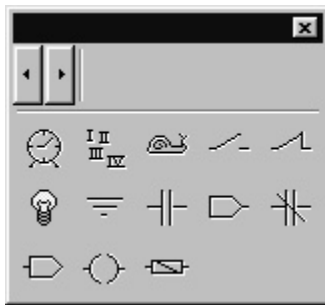


Figure 17 - The components bar

LadderWORK's components bar may change according to the software version. The components bar allows you to select components to be placed in the schematic. In the bar you can see a standard set of Ladder symbology, a set of electrical symbology and other more complex components like clock generators and counters.

The components are grouped for their functionality. For example, the third group contains the pure-logical symbols and the second group contains the analog devices. In the first group you can find the standard set of components. To change group use the spin buttons present at the top of the window. Pressing the left-arrow button will decrease the group number, pressing the right-arrow button will increase the group number. Group numbers greater than five contain PLC-dedicated devices and the functionality of these objects is discussed elsewhere.

LadderWORK loads the symbols information from a particular file called *Symbol Library* or .SLI file . This file is present in the library sub-directory

## 2.17 - Drawing schematics

### 2.17.1 - Placing components

To place a component, select a object in the component bar. Automatically the **place tool** will be selected. The shape of the selected object is shown if you move the mouse in the worksheet area. Placement is made by clicking with the left button on the mouse at the selected point. There are regions in the worksheet where the component can't be placed. For example you can't overlap the new component with an existing component. You can understand if the current position is right for placement observing the cursor. If an **NO-ACCESS** symbol appear near the cursor then the components can't be placed there.

### 2.17.2 - Moving Components

In order, to move a component the following operations should be executed. First choose the **pointer** in the **tools bar**. Now select a particular object in the schematic clicking with the left button on the mouse onto the component. Always keeping the left button pressed now you can move the components anywhere in the sheet. The wrong positions are always indicated with the **NO-ACCESS** symbol. The component may be placed in the new position simply releasing the mouse button.

Groups of components can be moved executing the following operations. Select the **pointer** or the **area tool** in the **tools bar**. Now, keeping the left button pressed on the mouse select a region including the desired components. Releasing the button will select all the devices included in the selected area. Now you can move the block performing a operation analogue to the single component moving. First selecting with the mouse one of the components included in the area and keeping the left button on the mouse pressed move the block anywhere in the sheet. The **NO-ACCESS** symbol will automatically appear if you enter into wrong regions. Effective placements will be made after releasing the button on the mouse.

### 2.17.3 - Deleting components

Components can be deleted individually or in group. To delete a single component select with the mouse the component. The component must be selected by clicking with the left button on the mouse on the component. Once selected the components will appear. Now you can use the delete command ( pressing the **CANCEL KEY** ) or the cut command to remove the object from the schematic. Using the cut command gives you the undo feature .

Group of components can be deleted choosing the **pointer** in the **tool bar** and selecting a region in the schematic. All the components included in that region will appear. Now you can definitely remove the group of objects pressing the **CANCEL KEY** or cutting the block in the clipboard with the cut command. Using the cut allows you to restore the deleting with the undo feature.

### 2.17.4 - Connecting components

Once the components are placed in the schematic you have to connect the device's pins with wires. To draw a wire first select the **wire tool** from the **tools bar**. With the mouse click on a object pin and keep the left button on the mouse pressed, drag the wire to the destination position. When the destination pin is reached the wire will be effectively placed realing the mouse button. Wires extremes are snapped by a fixed grid and errors in placement are impossible. Remember that wires are always orthogonals so you have to split your wire in more parts if the destination pin is obstructed by other objects in the schematic. The effective good connection between components can be checked by looking at the wires extremity. If the wire terminal is marked with a coloured box then the connection is right. Wrong connections are indicated by **across** .

Wires can also be connected to the power bars. Connecting a wire to the **left power bar** will give a logical one to the net ( **VCC** ), connecting a wire to the **right power bar** gives a logical zero to the net ( **GND** ) . Sometimes it is more practical to use **GND** devices instead of the **right power bar** . Wires can be moved using the same procedures used for components. Also the wires length can be modified performing the following procedure. Select with the mouse a wire extremity, a double arrow symbol will indicate the resizing, and keeping the left button on the mouse pressed move the extremity in the desired

point. Wire length will be modified by releasing the mouse button. It can happen that a wire terminal is connected to other wires. To select a particular wire in a node click once the mouse button on the node until the desired wire is selected then proceed with the resizing.

### 2.17.5 - Set components property

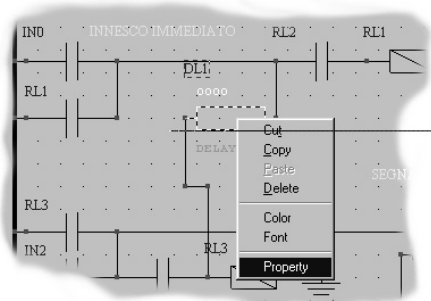


Figure 18 - Configuring property

After placing components, you have to set the components property. Generally, properties are variables local to the components that affects the behavior of the object. For example if you have placed a counter, the system should know the count range and the direction of step.

All these parameters can be configured performing a double click on the component or selecting the *Property* entry on the list opened with the right button on the mouse.

The property command will open the dialog associated with the component. The parameters included in the dialogs are relative to the placed object. See the LIBRARY section to find individual information about the parameters in the dialogs.

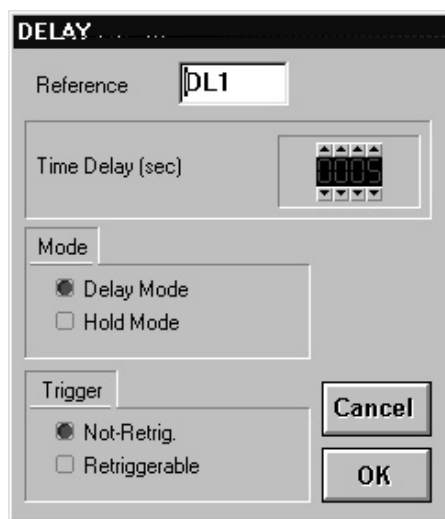


Figure 19 - A sample property dialog

For example, if we request the property command for a DELAY object we will be prompted for the following dialog. Once the parameters are configured you simply have to push the *OK* button to save the changes or press the *CANCEL* button to discharge the changing.

## 2.18 - Saving & loading projects

### 2.18.1 - Saving projects

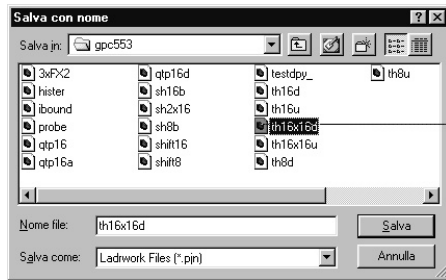


Figure20 - Save project dialog

Projects can be saved using the menu entries Save and Save As in the file menu. Also a shortcut key CTRL+S is present for the Save command. Save should be used every time you want to save the project you currently editing. If you attempt to save a new project the system performs the Save As task.

The Save As command should be used to assign a new name to the project so another project file will be created. If another file with the same name already exists, the system asks for the file overwriting.

### 2.18.2 - Loading projects



Figure21 - Load project dialog

Project can be loaded using the Load command in the file menu. The User will be prompted for a file selection in the current directory. If the User attempts to load a project when another project is already in memory and the project wasn't saved, the system will inform the User that the last modification will be trashed.

---

## 2.19 - Starting a new project

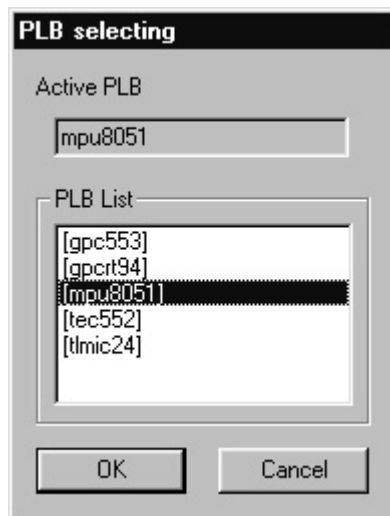


Figure22 - The device select dialog

In order, to begin a new project you have to set some parameters to inform the system about the PLC you are working on. Every time you begin a new project, using the **New** command, the program will prompt you for the PLC model. No other operations are possible until you don't specify a PLC model. According to this the here illustrated dialog will appear ..

Remember that selected PLC model can't be changed in your project. Once selected the choice will be stored in the project file.

Now, to work properly, the system must know other information like compiler options and memory mapping. All these parameters must be configured using the **Options** menu. To facilitate the configuration for a particular PLC, the **PROFILES** feature should be used. With profiles you can import dedicated configurations for a particular PLC model or create a new profile that can be used in other projects.

For example, if your working with the GRIFO GPC553 PLC, you can import the appropriate configuration for that PLC simply pressing the **IMPORT** button in the **Options** dialog and selecting the right profiles. At this point pressing the **OK** button will store the imported profile into the project and further modification to this configuration will affect locally only the project.

## 2.20 - Building the code

Code could be generated simply using the shortcut key **F5**. The software perform five or six main steps. For further information see the section **FLOW PROCESS**.

**Remember that the BUILD COMMANDS are available only if you have assigned a name for your project.**

- |                 |  |
|-----------------|--|
| <b>1st step</b> | In this phase the software checks the circuit for logical errors. Unconnected wires and components are reported and LOGICAL_LINKS rules are controlled. If all these tests are passed the system pass to phase two.  |
| <b>2nd step</b> | In this phase the system generates the netlist for the project.  |
| <b>3rd step</b> | In the third step the system will pass the control to the solver module. This process resolves the logical sequence for the function calling and generates the assembler file for the main project. In this phase logical errors like illegal loop and others are intercepted. |
| <b>4th step</b> | In the fourth step the control is passed to the assembler module. Normally in this phase no other errors should be present.  |
| <b>5th step</b> | In this compiler last phase the system calls the linker that proceed with the connection of the main module with the run-time library. In this phase the executable file ( HEX FILE ) is generated.  |
| <b>6th step</b> | If the PLC you are working on has the direct-upload feature, the system calls the communication module and the executable file is directly uploaded and your PLC will begin to run.  |





---

## 2.21 - Uploading the code

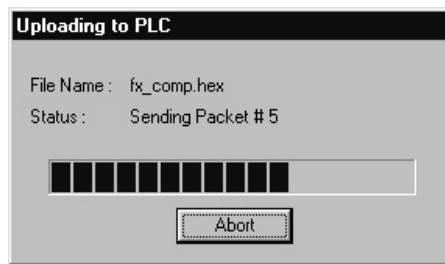


Figure23 - The upload dialog

Some PLC models have a direct upload capability. Using this feature you can upload the generated code directly to PLC using a standard RS232C port of your PC. To upload the code onto the PLC simply press the **F6** button on the keyboard or select *Upload* from the Build menu. Also uploading can be activated pressing the *Upload* button in the **Compile bar**.

Sometimes, to automatize the *Compile & Upload & Run* processes it is advisable to use the **F10** button. The three processes are executed sequentially simply pressing this button

## 2.22 - Running the PLC

PLC is normally started after the upload session automatically by the system if you use the **F10** shortcut key. If your PLC supports the direct-upload feature than other operations like **PLAY** and **STOP** could be performed. Use the **PLAY** button to run the PLC and use the **STOP** button to stop the PLC.

## 2.23 - Profiles

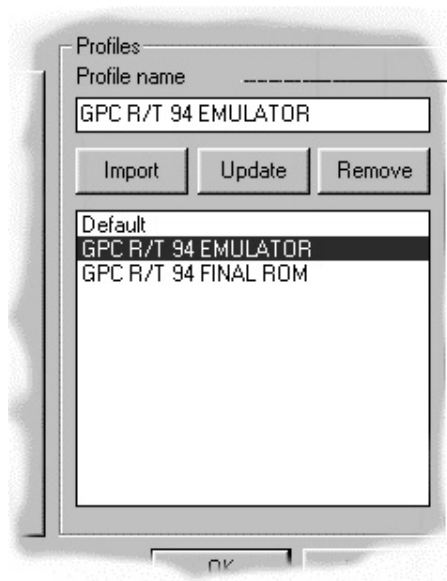


Figure24 - Profiles

Profiles are particular archives that store particular compiler configurations. The information affected by profiles are all the compiler options. With profiles you can freeze a particular configuration and store it in your project and into a private archive global for all the other projects. This is very useful for PLC with different memory mappings. For example, some PLC have different types of memory, normally RAM or E2PROM. Mapping must be changed if you want to upload the code in RAM or E2PROM. Uploading the code in RAM preserve E2PROM duration and is faster. According to this, you need to create two particular mappings, for RAM and E2PROM.

Once the configurations are created you can switch between settings simply by selecting a profile in the options dialog or changing the selection in the **compile bar**.

When you start a new project a particular **runmodifiable** default profile is created. This profile is normally out of use so you have to create your own configuration.

To create a new profile you simply have to modify the name in the Profile name text box and press the *Add* button. At this point you can change all the configuration parameters for your needs. Pressing the *Update* button will update the information in the profile. In this way you can create all the profile you need.

Remember that profiles are local to you project so the configuration you are changing will not affect any other project.

However you can use the import feature to copy a existing profile into your project. Pressing the *Import* button you will be prompted for a list of existing profiles. These profiles are stored in a private archive. Selecting the profile automatically imports all the information contained in this archive.

The *Remove* button allow you to remove any particular profiles from your project.

## 2.24 - Printing schematics

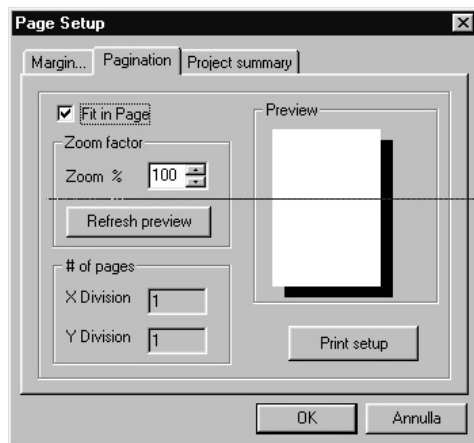


Figure25 - Page setup : Pagination

LadderWORK's worksheet is a virtual logic sheet without fixed dimensions. According to this, you can fit your schematic on your printer sheet operating on the **Pagination scheme**. After opening the **Page Setup** dialog, select the Pagination folder. The following dialog will appear

Checking the *Fit in Page* checkbox will force your drawing to be fitted on the selected printer sheet. In this case no other adjustment can be made to the layout.

Removing the *Fit in Page* option will enable the zoom factor spin button. You can reduce or expand the drawing changing the percentage factor. Expanding the drawing size, a out-of-bound situation can occur. In this case the software automatically splits the sheet in two or more pages. The total generated pages can be seen in the section *# of pages* of the dialog.

Pressing the *Refresh preview* button will show the width of the drawing respect to the sheet.

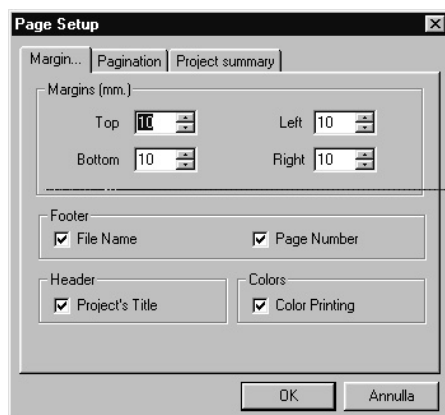


Figure26 - Page Setup : Margin

Margin parameters and other setting can be adjusted selecting the *Margin* folder. In the Margin section of this folder you can set the sheet margin and configure the information that will appear in the sheet.

### File Name

Checking this box enables the printing of the file name at the bottom of the sheet.

### Page Number

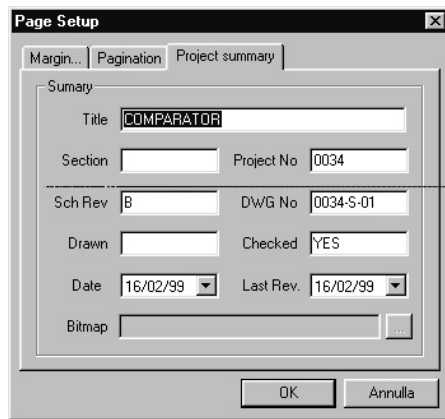
If active includes the page number in the sheets.

### Project's Title

The activation of this check box enables the printing of the title at the top of the sheet,

### Color Printing

This check box enable/disable the color printing.



The screenshot shows the 'Page Setup' dialog box with the 'Project summary' tab selected. The dialog has three tabs: 'Margin...', 'Pagination', and 'Project summary'. The 'Project summary' tab contains a 'Summary' section with the following fields:

Field	Value
Title	COMPARATOR
Section	
Project No	0034
Sch Rev	B
DWG No	0034-S-01
Drawn	
Checked	YES
Date	16/02/99
Last Rev.	16/02/99
Bitmap	

At the bottom of the dialog are 'OK' and 'Annulla' buttons.

The last folder of the Page Setup allows you to enter information about the project. This information will be printed in the bottom-right corner of the sheet.

Figure27 - Page Setup : Project summary



Page intentionally left blank

## **SECTION 3 - TUTORIAL**

### **3 - Tutorial**

#### **3.1 - About Tutorial**

The LadderWORK Tutorial contains the information you need to get started with this new graphical LADDER language programming software. LadderWORK simplifies the work of LADDER language programmer with an user-friendly graphical interface which with 'drag and drop' approach allows you to realise, load and test anything you can image in PLC programming World.

This manual gives you an overview of the fundamental concepts of LadderWORK, and include lessons to teach you what you need to know to program your PLC.

This manual presumes that you know how to operate your computer and that you are familiar with its operating system.

##### **3.1.1 - Organisation of this chapter**

Each paragraph discusses a different LadderWORK concept, although you can design your virtual circuit which may incorporate several of these basic concepts. Therefore, we suggest you to work through the entire tutorial before you begin building your application.

This chapter is organised as follows:

- Paragraph 3.2, Introduction to LadderWORK, describes what LadderWORK is, what a Virtual Circuit (VC) is, how to use the LadderWORK environment, how to check VCs, how to edit VCs, and how to create VCs.
- Paragraph 3.3, Working with VCs – elementary components, describes, using practical examples, the main properties of the elements common to the various families of PLC supported by LadderWORK and their mutual interactions.
- The Glossary contains an alphabetical list of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics and symbols.
- The Index contains an alphabetical list of key terms and topics in this tutorial, including the page where you can find each one.



### 3.1.2 - Conventions used in this section

The following conventions are used in this section :

<b>bold</b>	Bold text denotes menus, menu items, or dialog box buttons or options. In addition bold text denotes VC input and output parameters.
<i>Italic</i>	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
<b>Bold italic</b>	Bold italic text denotes a note, caution or warning.
Courier	Courier font denotes text or characters that you enter using keyboard. Selections of code, programming examples, syntax examples, and messages and responses that the computer automatically prints to the screen also appear in this font.
➡	<p>The ➡ symbol leads you through nested menu items and dialog box options to a final action. The sequence</p> <p><b>Options ➡ Colors ➡ Reference ➡ Browse color</b></p> <p>Directs you to pull down the Option menu, select sub-menu, select Reference item and finally select the Browse color option from the last dialog box.</p>

### 3.1.3 - Related documentation

The following documents contain information that you may find helpful as you read this manual:

### 3.2 - Introduction to LadderWORK

This Paragraph describes what LadderWORK is, what a Virtual Circuit (VC) is, how to use LadderWORK environment (windows, menus, and tools), how to operate VCs, how to edit VCs and how to create VCs.

Because the great variety of the possible realisation , this tutorial cannot practically show how to solve every possible programming problem. Instead, this tutorial explains the theory of LadderWORK, contains exercises to teach you to use the LadderWORK programming tools and guides you through practical uses of LadderWORK features as applied to actual programming tasks.

#### 3.2.1 Paragraph Information

Each paragraph begins with a section like the one that follows, listing the learning objectives for that paragraph.

##### **You Will Learn:**

- What LadderWORK is.
- What a Virtual Circuit (VC) is.
- How to use the LadderWORK environment (windows and tools).
- How to operate VCs.
- How to edit VCs.
- How to create VCs.

#### 3.2.2 - What Is LadderWORK?

LadderWORK is a program development application, much like various commercial C or BASIC development system. However, LadderWORK is different from those applications in one respect. Other programming system use *text-based* languages to create the appropriate lines of code for each particular family of PLC, while LadderWORK uses *graphical* programming language, compliant to CEI symbolism, to create programs.

You can use LadderWORK with little programming experience. LadderWORK uses terminology, icons and ideas familiar to engineers and electronic technicians and relies on graphical symbols rather than textual language to describe programming actions.

LadderWORK has an extensive library of Ladder language idioms so it can match easily to different automation systems. LadderWORK includes conventional program development tools, so you can see how data passes through the program and make debugging and program development easier.

#### 3.2.3 - How does LadderWORK work?

LadderWORK programs are called *Virtual Circuits* (VCs) because their appearance and operation imitate actual circuits. However, they are analogous to functions from conventional language programs. Each of the possible elements of the component library represents a module of the program. When the user selects and places a new component on the actual Project layout connecting it by wires to the circuit automatically he links the correlated program module to the program under construction. During the compiling phase the Virtual Circuit is then translated in a Ladder program which can be stored or sent to the PLC.

### 3.2.4 - Installing LadderWORK

For instructions on how to install LadderWORK, see the paragraph 2.2 of Reference Manual. In any case the hardware configuration must resemble the picture below.

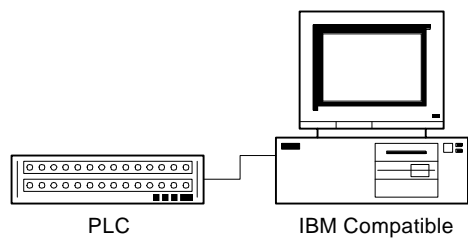


Figure28 - Hardware configuration

### 3.2.5 - Virtual Circuits (VCs)

LadderWORK programs are called virtual circuit (VCs). They are described by a collection of elementary component connected together by wires representing the interaction between the PLC and the its working environment.

#### Objective

To open, examine, and operate a VC, and familiarise yourself with the basic concepts of a virtual circuit.

Open LadderWORK by double-clicking with the mouse button on the LadderWORK icon in the LadderWORK group. After a few moments, a blank, untitled Project Layout appears.

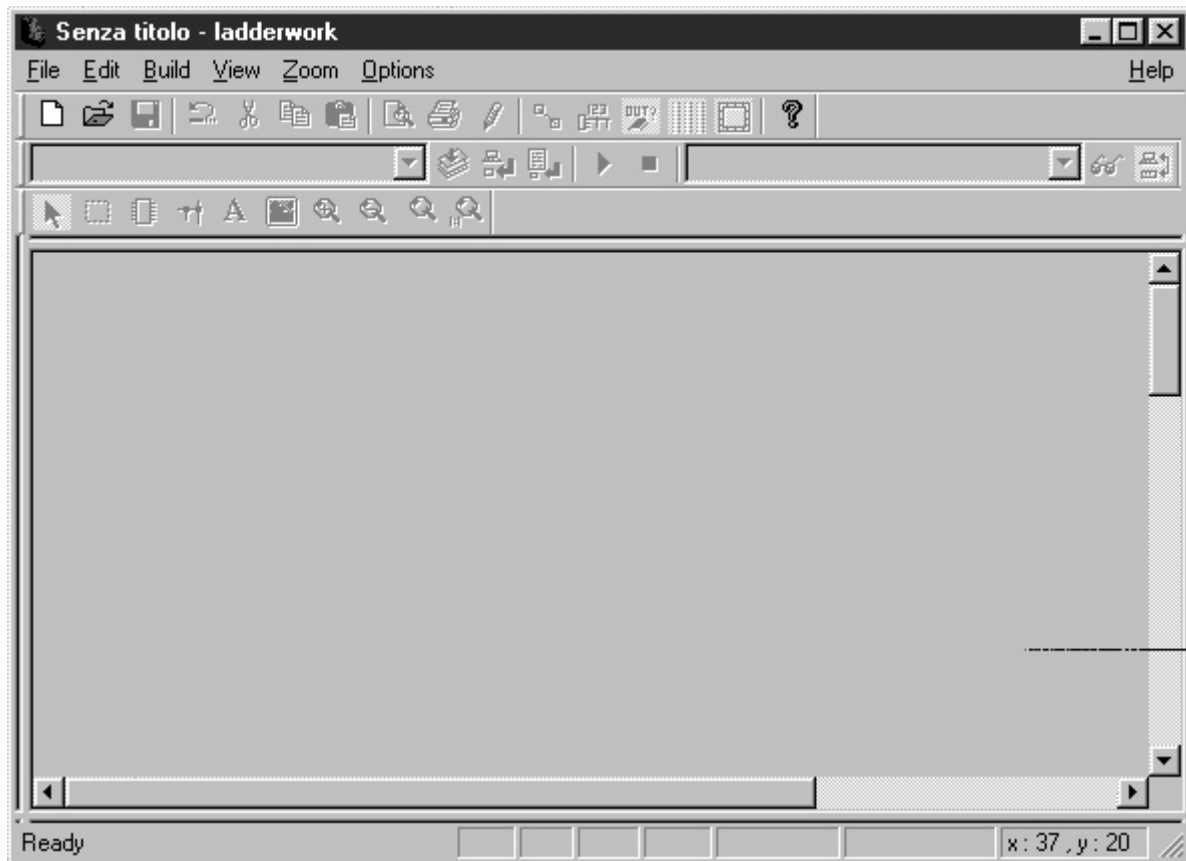


Figure29 - IDE representation

Select **File**➔**Open** option.

Select **project/samples/gpc553/tutor** directory. If the PLC in use is different from GPC553® PLCs choose the relative subdirectory present in project/samples. Remember *from this time the default subdirectory is the subdirectory selected for the first time*. In any case It is always possible to change in the File➔Open dialog box the PLC subdirectory.

Load the Rampdemo VC by double-clicking on **Rampdemo.pjn** icon.

After a few moments, the Rampdemo VC appears on the Project layout. As described in the Reference Manual the VC contains several elementary components connected together by wires, a Reference Voltage bar on the right border and a ground bar on the left one. *To expand or contract the horizontal/vertical dimension of the layout put the cursor on one of the two bars and drag the bar until the desired vertical/horizontal dimension of the layout is reached ( Valid only in version 1.00.x ).*

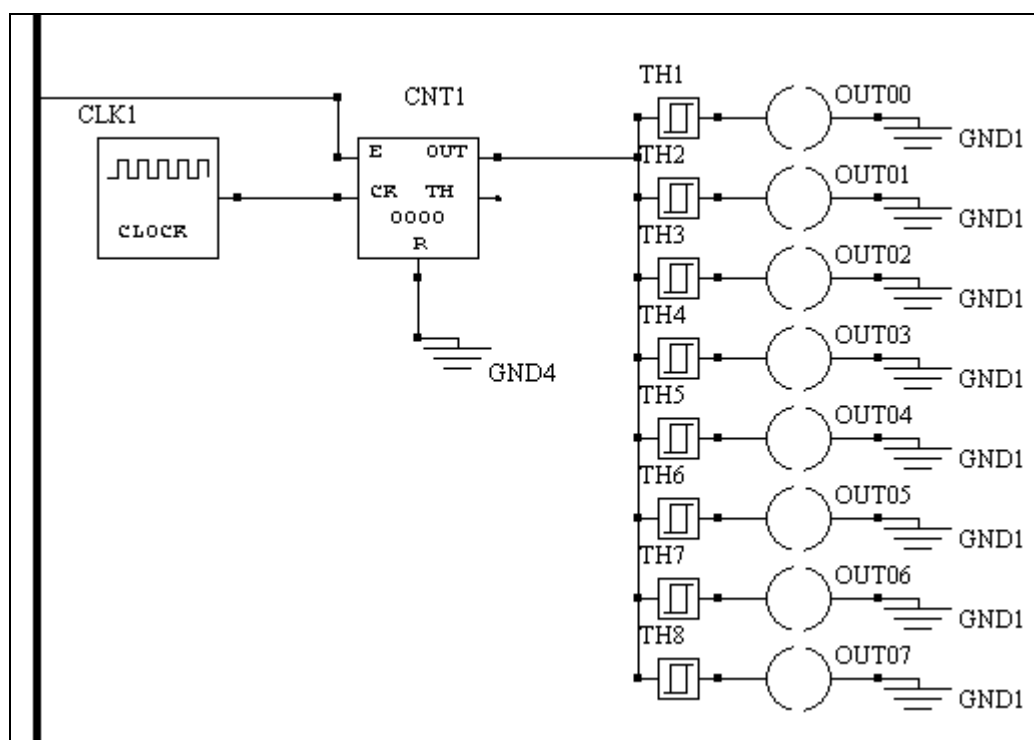


Figure30 - Rampdemo project

### 3.2.6 - IDE toolbar

As described in the Reference Manual The IDE contains a toolbar of command buttons and status indicators to compile, run and debug VCs. IDE also contains font and color options for editing VCs.



Figure31 - IDE Toolbar





Compile the VC by selecting **Build⇒Compile** option or clicking on  pushbutton, after few second the following dialog box appears.



Figure32 - Rampdemo.pjn compiling result

Note in the lower part of the IDE the Message Window resuming the results of the compile and link actions. In this case no error has been detected and the Message Window reports the dimension of the main modules of the compiled program. It is now possible to download the program to the PLC by the **Build⇒Upload** option or clicking on  pushbutton. If no doubt exists on formal and logical correctness of the implemented VC it is also possible to execute together the compile and download actions using the appropriate **Build⇒Compile & Upload** option or clicking on  pushbutton. To run the PLC program use the **Build⇒Run** option or  pushbutton. The result of run action of the Rampdemo VC will be the sequenced turn on of the output LED of the PLC.

### 3.2.7 - Principal tricks and tips in connecting VC devices

In VCs realisation, to compile correctly the VC, some principal rules must be followed during the connecting and label assignment operations of the VC devices. Any inattention generates an error message in the Message Window to help the user in debugging operation.

#### Objectives

To examine and focus the principal connecting and label assignment errors. To get the user aware of the more common IDE tricks and tips.

#### Example n°1


Following the steps described before open the **demoerr1.pjn**. This project was just created to produce errors during compiling.

Chose the **Build** ➔ **Compile** option and the following dialog box appears.



Figure33 - Error Warning Box

In the Message Window are resumed the errors detected by the compiler and the linker, in this case it is reported the message:

 **ERROR : (SLV0018)** Can't find a valid evaluation path through the network.

Before compiling always check the correct wiring of the VC, in particular verify the correct positioning of the terminal nodes of the wires.

Remember: *two wires crossing each other without any common node are electrical disjoint*

## Example n°2

Open **demoerr2.pjn** VC using the previous described procedure. This project was created to evidence a particular error message that appear when user leave unconnected pin in the schematic.

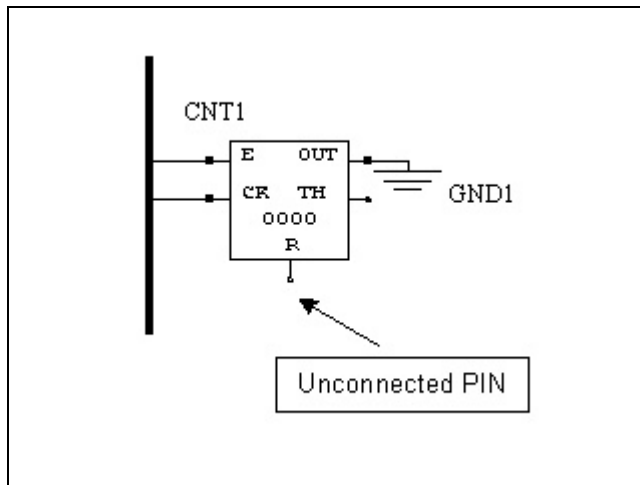


Figure34 - Not connected reset input

Compile VC, a dialog box appears with an error message. In the Message Window the reported message is:

 **ERROR :** (SLV0024) Component <COUNTER> with REF=<CNT1> has an hanged plug <R>

Before compiling always check the presence of expected Inputs of each component.

Remember: *It is possible to assign default values to the Input terminals of the VC devices by setting ON the **Options**→**Compiler**→**Assume default for hanged inputs** Switch.* In this case no error message appears and the VC is correctly compiled. For default input values of each VC device refer to the Component Library.



### Example n°3

Open **demoerr3.pjn** VC using the previous described procedure. The project was created to evidence a typical sourceless-input condition.

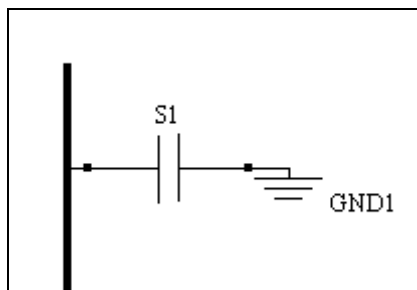



Figure35 - Always check the correct label assignment of input devices

Compile VC, a dialog box appears with the error message. In the Message Window the reported messages are:

 ERROR : (SFD0207) INPUT : SOURCELESS INPUT REF = <S1> INPUT/DELETE

Before compiling always control the label assignment of each VC Input device to the appropriate 'real' (one of the PLC hardware Input) or 'logical' input (the software link of an Input device to a Relay device). For more information on 'logical' input refer to Relay in the Component Library.

**Example n°4**

Open **demoerr4.pjn** VC using the previous described procedure. The project was created to evidence a typical output-conflict error.

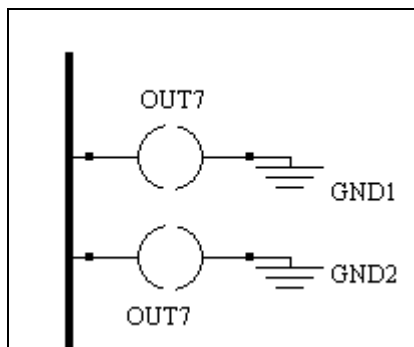



Figure36 - Always check the correct label assignment of OUTPUT devices

Compile VC, a dialog box appears with an error message. In the Message Window the reported messages is:

 ERROR : (SLV0031) OUTPUT <OUT7> CLASHES WITH OUTPUT <OUT7>

Before compiling *always verify the assignments of output labels to the VC output devices*

### 3.3 - Working with VCs – elementary components

This Chapter describes, using simple but practical examples, the main characteristics of the “elementary components” common to the different PLC families supported by LadderWORK.

#### You Will Learn:

- How to operate VC Clock Generator devices.
- How to operate VC delay devices.
- How to operate VC AND, OR, NOT Ports.
- How to operate VC Counter devices.
- How to operate VC Input and Output devices.
- What a logical link is.
- How to operate VC Debounce devices.
- How to operate VC Threshold devices.

#### 3.3.1 - VC Clock Generator device

Clock Generator device is a software square wave generator. The properties of the output signal can be changed, as described in the Reference Manual, by selecting the device and double clicking on it. A dialog box appears to modify the Reference Label of the device and the frequency of the output signal. The range of the possible values of the frequency changes with PLC but in any case it must be greater than zero. Let's suppose now to want to realise, using a PLC, a control system of the lights of the Christmas tree of our firm. The simplest solution might be the following VC.

Open **tutor1.pjn** VC. A simple VC appears with two Clock Generator devices connected to the Output devices, representing the Christmas tree lights, through two Input devices.

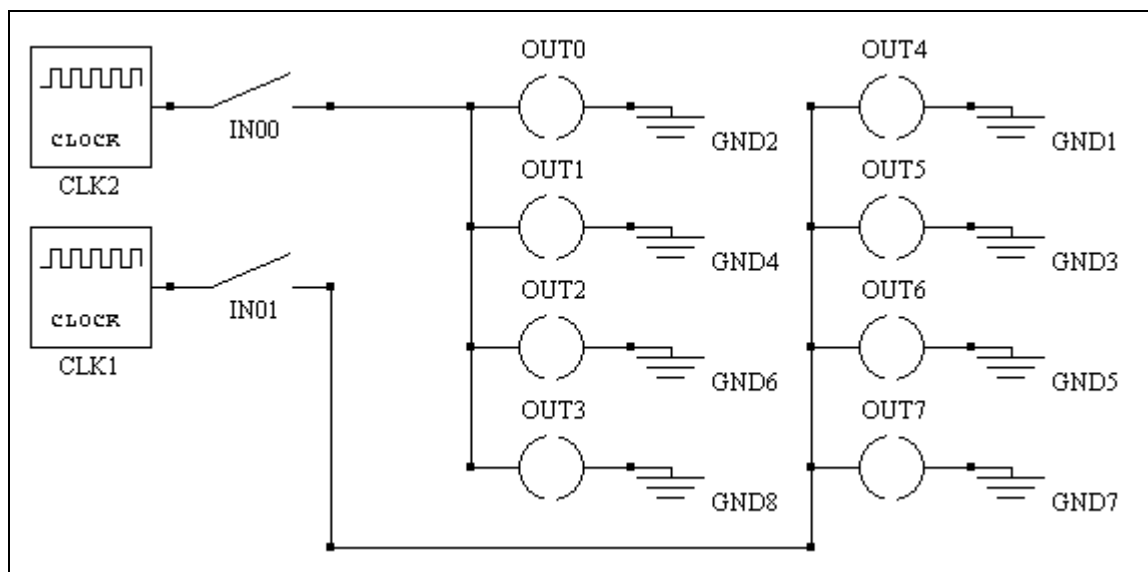


Figure37 - tutor1.pjn VC layout

Select **Build►Compile & upload & Run** Option. The Output LEDs 4-7 flash at the frequency of 0,5 Hz when INPUT 00 is closed while the Output LEDs 0-3 flash at the frequency of 1 Hz when the INPUT 01 is closed.

### 3.3.2 - VC Delay device

The VC Delay device can act as a delay line or as a monostable. The device behaviour can be changed acting on the **Delay/Hold** mode switch. Moreover the device reaction to the input signal can be modified acting on **Not retrigg./Retriggerable** mode switch. In **Not retriggerable** mode any signal at the Input terminal of the device is ignored during the delay/hold time of the device. In **Retriggerable** mode the pulses occurred during the delay/hold time are stored and processed by the device at the end of the delay/hold time. Come back to our Christmas tree lights control system and suppose to want to realise a VC where odd and even Output LEDs flash alternatively. One possible solution might be the following.

Open **tutor2.pjn** VC. The VC is composed by a clock Generator with an Output frequency of 0,5 Hz (1 impulse per 2 seconds) and two VC Delay devices. The VC Output section is similar to the previous. Note the first Delay device is set in delay mode while the second Delay device is set in hold mode.

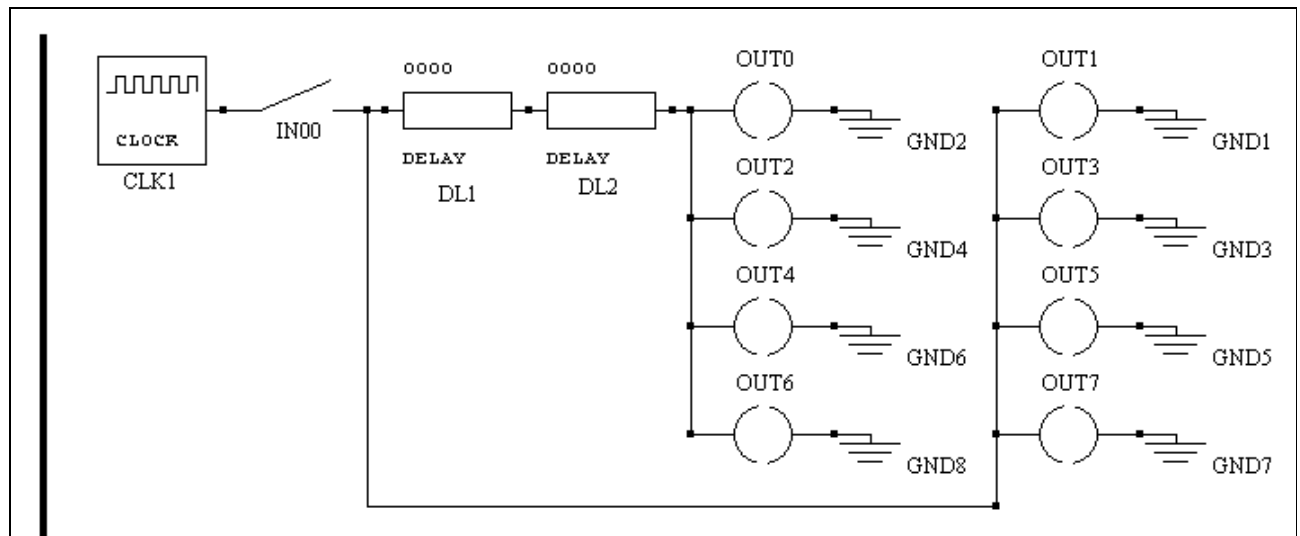


Figure38 - tutor2.pjn VC layout

Select Build → Compile & upload & Run Option.

Close the switch associated to INPUT 00. The odd and even Output LEDs flash at the same frequency but with a phase difference of half a period.

The behaviour of the signals is the following:

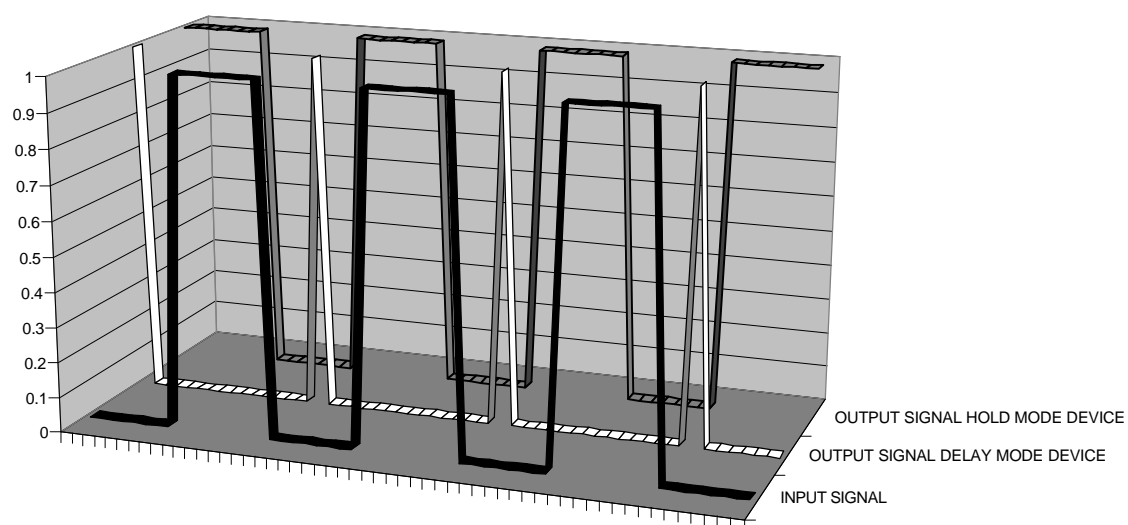


Figure39 - Time representation of Input and Output signals in tutor2.pjn

### 3.3.3 - AND, OR, NOT Ports

The behaviour of LadderWORK logical devices is similar to the theoretical behaviour. The only exception occurs when an Input terminal is connected to an Input device. In this case the open circuit status is considered as a logical “zero”. To analyse this class of components consider the following VC relative to an another version of Christmas tree lights control system. In this release the VC is characterised by two functional modes: a flashing mode which turns on and off the output LEDs at a frequency of 1Hz and switching off mode which switches off in a pre-programmed sequence the Output LEDs. To examine the VC execute the following steps:

Open **tutor3.pjn** VC. The VC is divided in three functional blocks: a clock generator block near the left border of the VC layout responsible of the control and output signal generation, an output block near the right border with a set of AND ports used both for the automatic mode selection and for the sequenced switching off mode realisation and a delay time block responsible of the sequenced switching off of the Output LEDs.

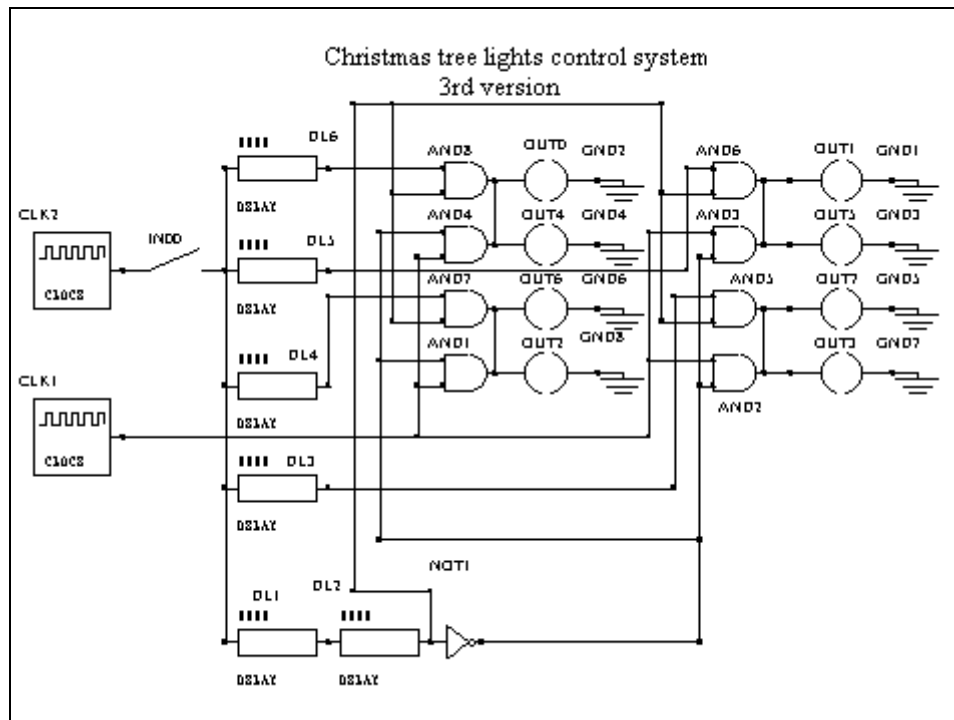


Figure40 - tutor3.pjn schematic layout

Select Build → Compile & upload & Run Option.

Close INPUT 00 Input device the Output LEDs start to the switching on and off cycle as previously described. N.B. The command signal for mode selection is obtained using a 0.2 Hz signal conditioned by two Delay devices set in **Not Retriggerable** mode. In this way a square wave generator with pulse repetition frequency of  $\frac{1}{6}$  Hz is realised.

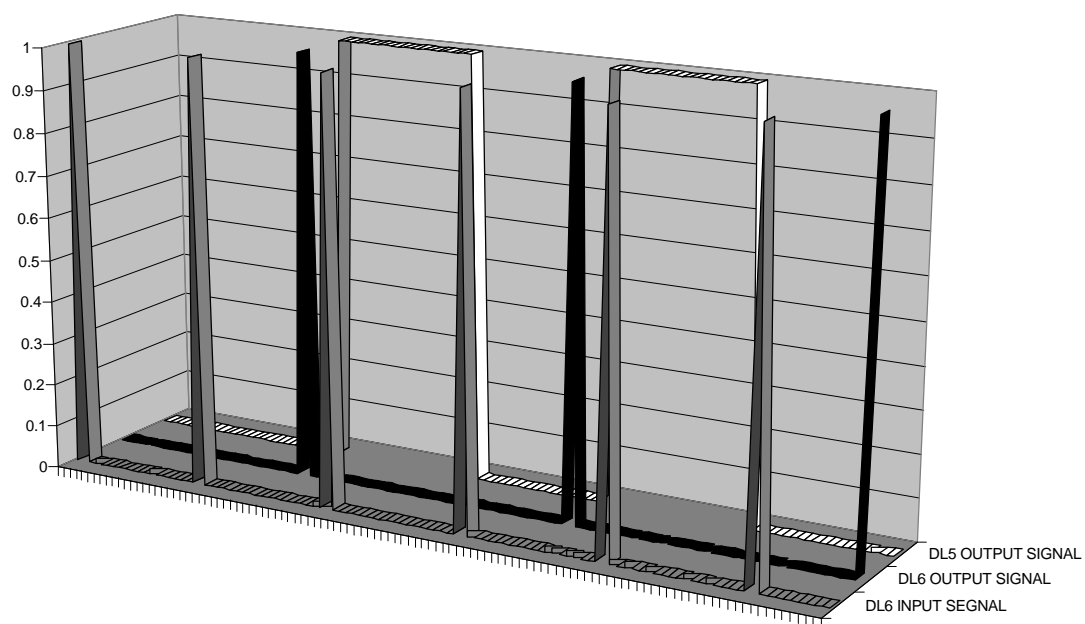


Figure41 - Time representation of Input and Output signal in tutor3.pjn

### 3.3.4 - VC Flip Flop D Device

Flip Flop D in real circuit is usually implemented using four AND ports properly connected. To help the user in VC drawing LadderWORK presents the Flip Flop D among the implemented VC devices. In the previously considered VC the command signal was obtained conditioning an 0.2 Hz signal. An another strategy should be the following:

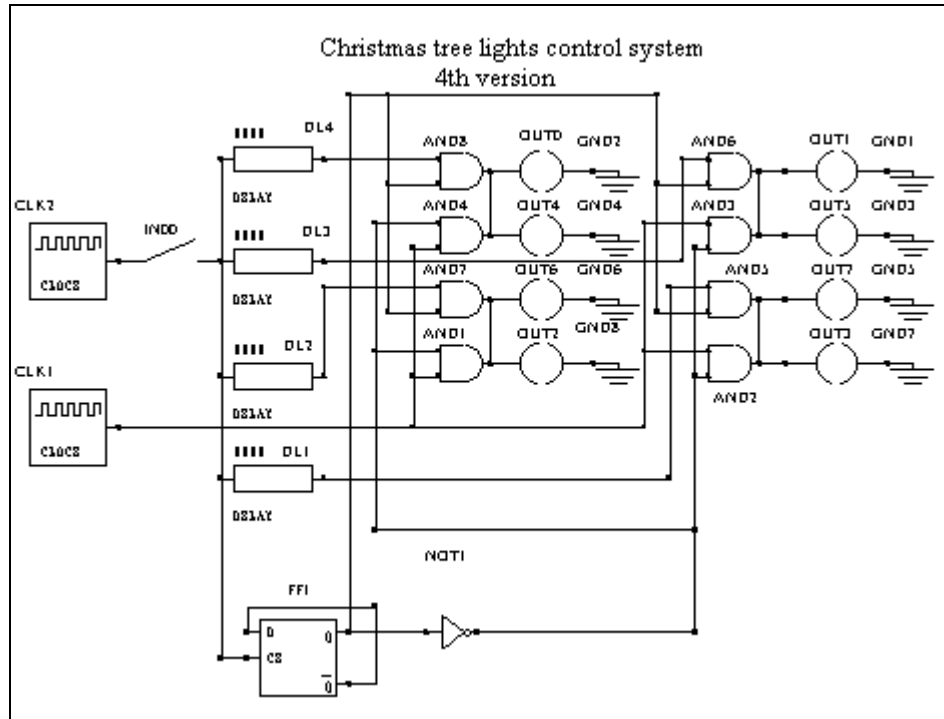


Figure42 - Use of Flip Flop D Device

Open **tutor4.pjn** VC. The VC is similar to the previous version, the only variation is the use of a feed back Flip Flop D to double the period of the 0.2 Hz input signal. In fact the Flip Flop Output status, in this configuration, changes only when the clock input signal presents a rising wave front.

Select Build → Compile & upload & Run Option.

Close INPUT 00. The Output LEDs flash as previously described. The only difference consists that using Flip Flop D the flashing and switching off periods are exactly equal.



### 3.3.5 - VC Debounce Device

The use of DEBOUNCE devices is particularly important when the Input devices connected to PLC are electromechanical relays. In fact this type of devices are characterised by a “bouncing behaviour” during the transaction from the open state to the closed one and vice versa with the generation of a lot of rising wave fronts which can random modify the circuit response. For this reason it is sometime necessary to introduce a low pass filter to eliminate any spurious fluctuation of the signal during state variations. This is the main function of a Debounce device.

Another possible use of the Device is the introduction of delay of few milliseconds in the propagation of the state variations. In fact the “Debounce” property is obtained implementing an average operation on the input signal calculated on a user pre-programmable period. Therefore the output signal of a DEBOUNCE device is a delayed and averaged version of the input signal.

A practical example of the device use is the realisation of a control system for an elevator where the actual position of the elevator car is provided by some relays. To simplify the controller the VC manages only two elevator stations.

Open tutor5.pjn VC. The VC is composed by three functional blocks: an Input block composed by INPUT 00 and INPUT 01, an enable block which allows to run the motor only in the right direction and an output block which provides the command signals for the motor.

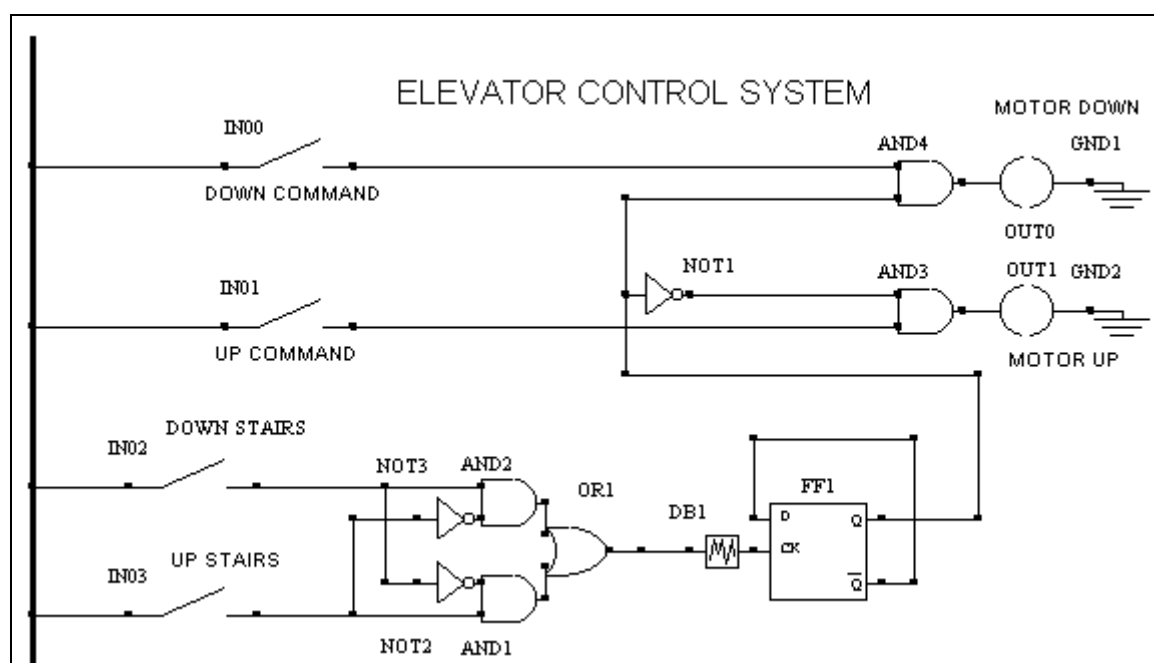


Figure43 - tutor5.pjn Elevator Controller

Select **Build►Compile & upload & Run** Option. To run correctly at the beginning the system status must be 'elevator car downstairs'.

Try the system, no problem should be revealed.

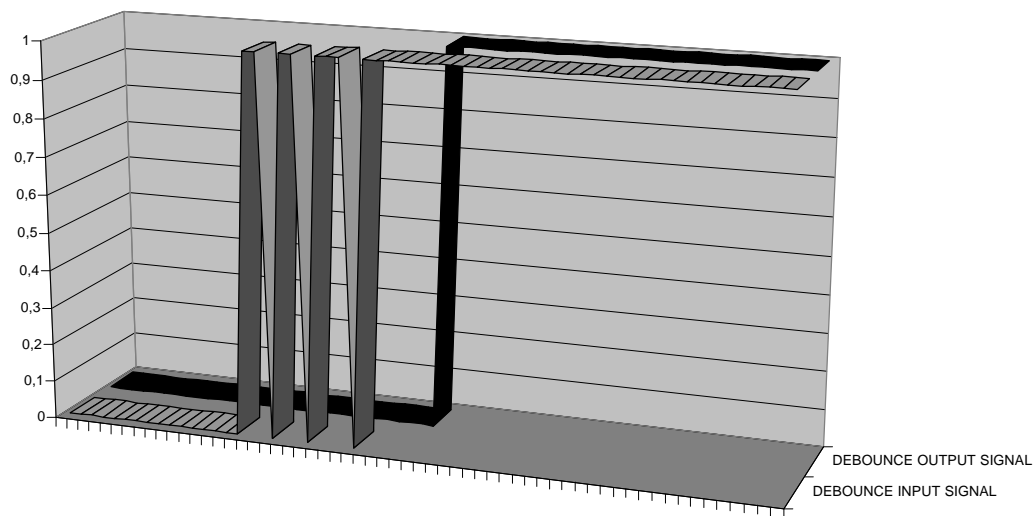


Figure44 - Time representation of input and output signal of the Debounce device

Remove the Debounce device and extend the wire to connect together OR1 with FF1.

Select **Build** ➔ **Compile & upload & Run** Option again. Now The controller doesn't work properly. In fact the relay bouncing random modify the status of FF1 (each rising front of the OR1 output signal changes the status of FF1) and doesn't allow to enable correctly the elevator rising/lowering movements.

### 3.3.6 - VC Counter Device

VC Counter Device main function is to count the number of input pulses (properly the number of rising fronts), to provide the digital value of them on OUT terminal, to set high TH output when the threshold value is reached. The threshold can be modified by the user. Moreover through the “E” and “R” inputs it is possible to enable or reset the device. A practical example of the use of this device is the Water distributor described bellow. The distribution logic is: to fill a vessel with a pre defined amount of liquid allowing the user to stop the filling up by pressing the push button a second time.

Open **tutor6.pjn** VC. The VC is illustrated in the following picture.

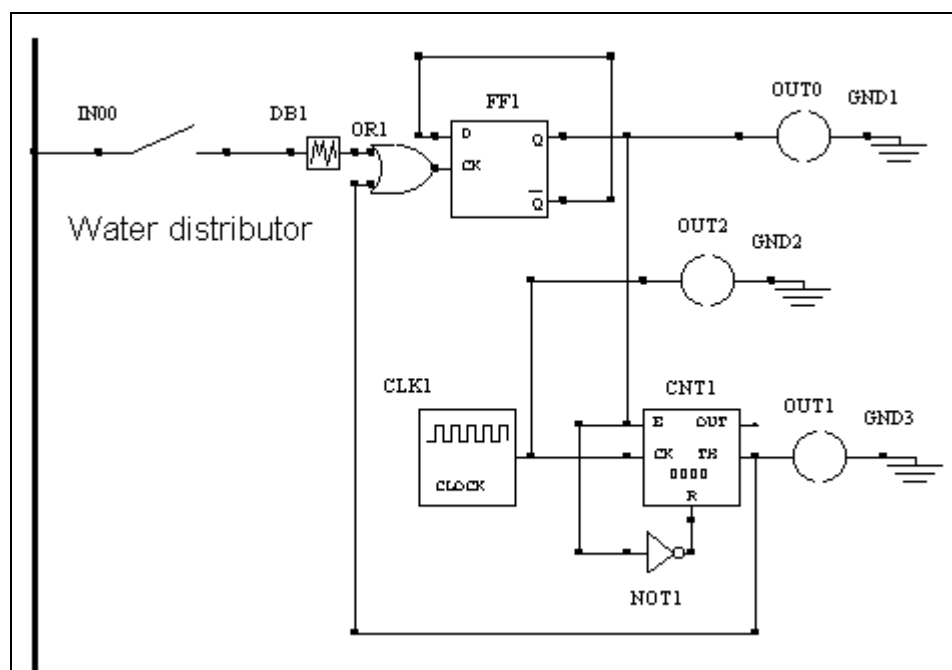


Figure45 - tutor6.pjn Water Distributor

Select **Build➔Compile & upload & Run** Option. When the user presses and releases the input Terminal INPUT 00 the water valve OUT00 opens for a pre programmed time interval. If the time interval is over or the user presses and releases again INPUT 00 the water valve closes. The pre programmed time interval is obtained using a Counter Device CNT1 setting properly the threshold value of the device and using the “TH” Output to stop the water flow. Moreover The Output signal of the Flip Flop D is used to reset the counter.

### 3.3.7 - VC Relay Device

The Relay Device allows to manage several output devices at the same time avoiding the employ of logical devices (AND OR ports) and simplifying VC layout. This class of devices implements the “Logical Links” between the input device (the Relay) and the assigned input devices (for more information about them refer to the Reference Manual). Assignment operation is realised selecting the input devices and giving them the same Label of the Relay. An example of the use of Relay Device is represented by an “Automatic gate control system”. Usually this kind of system must ensure the following characteristics:

- To open the Gate on user Input Command.
- To close the Gate after a pre programmed time interval.
- To stop the closure procedure if an object crosses the gate aperture.

A possible practical solution is the following VC. From a functional point of view the VC is divided in:

a flashing light control block which manages alarm lamp during the opening and closing procedures,

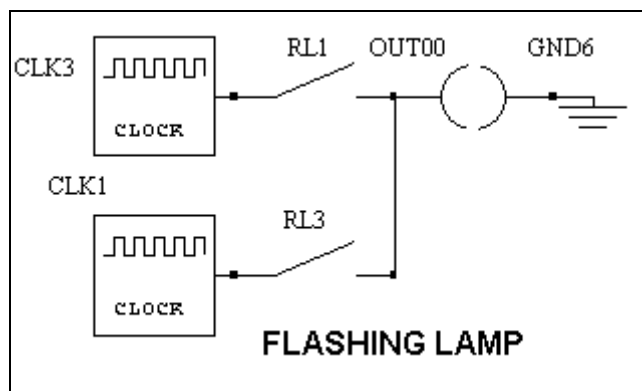


Figure46 - Flashing light Control Block

A Crossing Sensor Block, which interrupts the closing Procedure and immediately starts the re-opening of the gate.

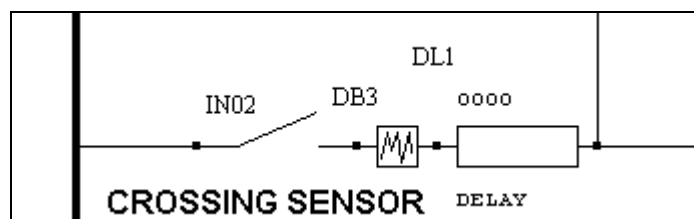


Figure47 - Crossing Sensor Block

The Opening Logical Block,

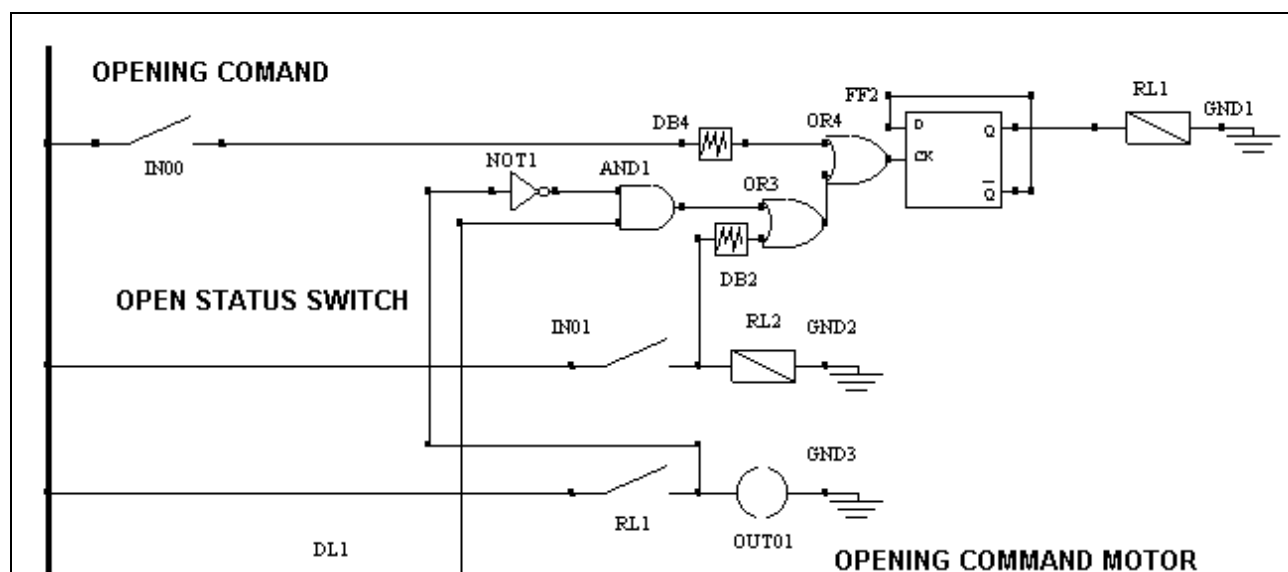


Figure48 - Opening Logical Block

### The Closing Logical Block

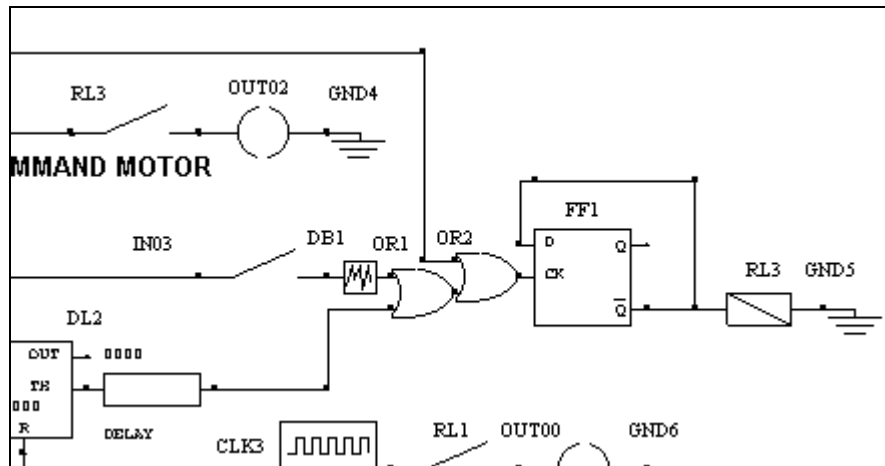


Figure49 - Closing Logical Block

The Timer Block which defines the aperture time interval.

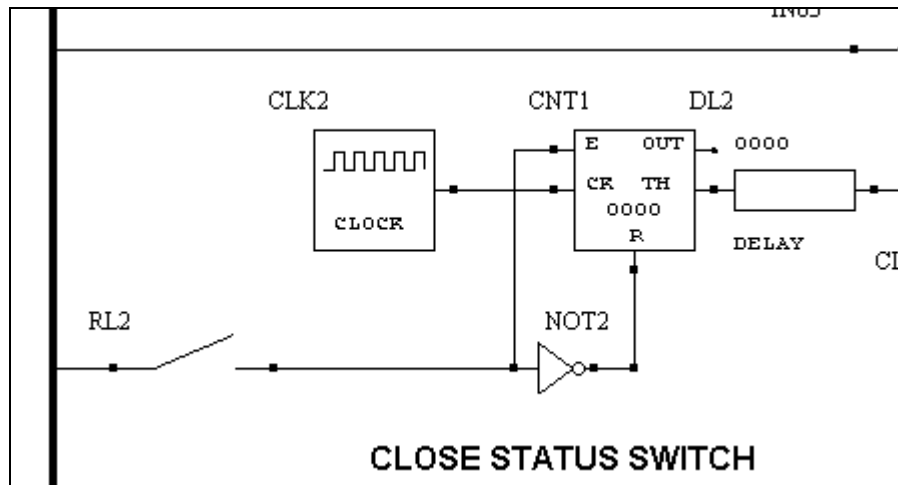


Figure50 - Timer Block

Open **tutor7.pjn** VC.

Select **Build►Compile & upload & Run** Option. The VC functional logic is to open the gate when the user presses INPUT 00. When the Open Status is reached the closing procedure starts after a pre programmed time interval when the 'TH' Output of CNT1 is high. This procedure stops if either the gate is completely close (this status is reached when the Close Status Switch INPUT 03 is closed) or the Crossing Sensor INPUT 02 is closed. To avoid false Signal from the Crossing Sensor a Delay Device set in Hold/Not Retrigger Mode is used.

### 3.3.8 - VC Threshold Device

The VC Threshold device is used to compare the Counter device output to a pre programmed value. The compare condition can be selected by the user. When it is reached the Threshold Output device is set 'True'. A practical example of a VC containing Threshold Device is a simplified version of a Fuel Distributor. Usually this kind of system must ensure the following characteristics:

To allow the user to select the needed amount of fuel.

To fill the tank decreasing the selected amount of fuel still its remaining value is zero.

A possible solution is the following. The VC is divided in three functional block:

An Input Block which, in this simplified case, allows the user to select a maximum amount of three units of a pre programmed quantum of fuel.

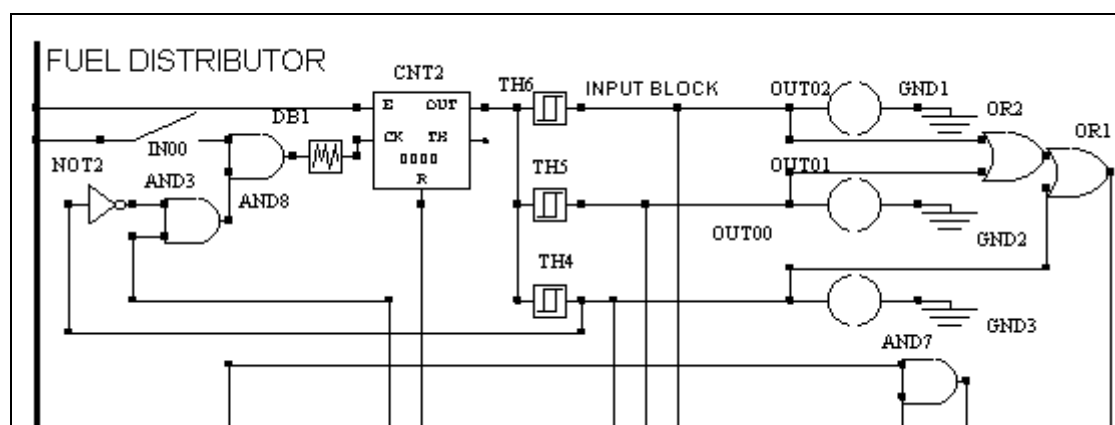


Figure51 - Fuel Distributor Input Block

A compare Block which controls the fuel distribution and stops it when the selected amount of fuel is reached.

An Output Block which controls the distribution valve.

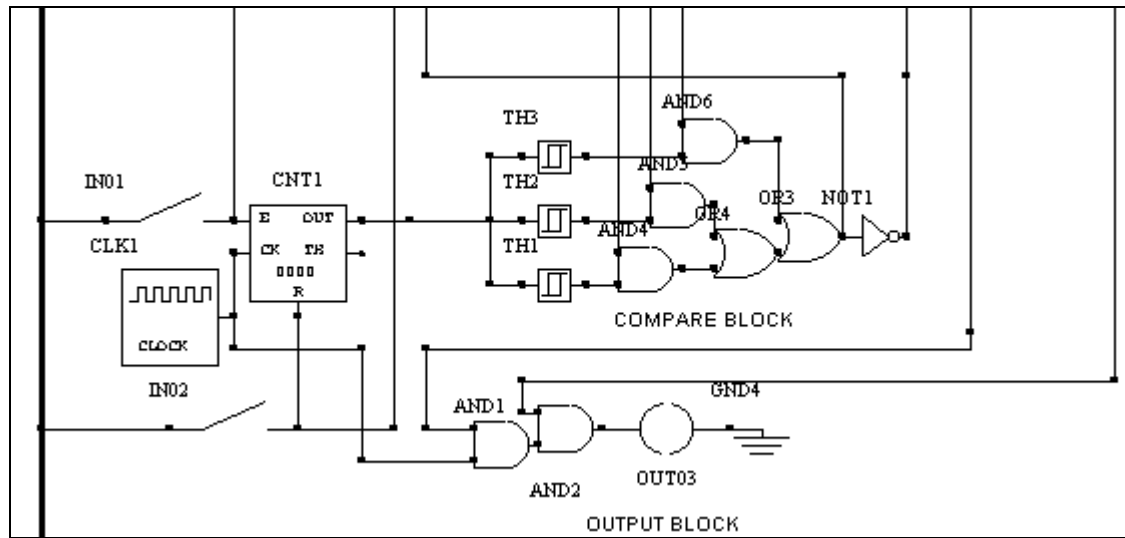


Figure52 - Output and Compare Block of Fuel Distributor

Open **tutor8.pjn** VC.

Select **Build** ➔ **Compile & upload & Run** Option

The rising front of INPUT 00 signal increases the Fuel Selection Counter CNT1. After three rising fronts the AND1 Port setting prevents any further CNT1 increment. The user can control the current amount of fuel selected checking the Output Device OUTPUT 0, OUTPUT 2 and OUTPUT 3. The INPUT 02 Input Device is the switch relative to the extraction/insertion fuel distributor pump. When the fuel distributor pump is extracted INPUT 02 is open. In this case any further increment of CNT1 is prevented and at the same time the Fuel Distribution Counter CNT2 is no more in Reset Status. When this condition occurs and the user closes the Command Distribution Switch INPUT 01 CNT2 is enabled to increase its current value and the Distribution valve is opened. If CNT2 Output gets equal to the selected value the Compare Block stops Fuel distribution.



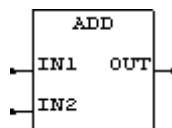


## SECTION 4 - LIBRARY

## ADD

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



This device performs the sum of the two values present at inputs IN1 and IN2 giving the result on the OUT pin.

### Net Plugs

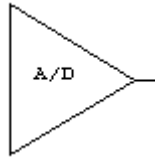
Pin	Description
IN1	First operand
IN2	Second operand
OUT	Result of the sum

**See also** : Mathematical expressions

## AD\_CONV

Software version : ADVANCED

---



Some PLC models has one or more analog to digital converters. This device allows you to aquire analogs value and convert it in a numerical value. The value supplied by the converter is normalized in the range 0-65535 indipendently by the converter resolution. Two parameters, called OFFSET and SPAN, are available to change the converter dynamic.

The OFFSET parameter adds a base value to the converted value and the SPAN parameter changes the converter gain. With values OFFSET=0 and SPAN=1.0 no alteration will be applied to the converted value. SPAN values greater than 1.0 produce a gain, SPAN values less than 1.0 produce value attenuation.

### **Dialog Settings**

Parameter	Description
OFFSET	This parameter adds a OFFSET value to the converted value
SPAN	This parameter changes the gain of the converter.

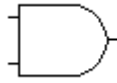
### **Net Plugs**

Pin	Description
OUT	Data output (normalized 0-65536).

## AND

Software version : STANDARD, ADVANCED

---



The AND device performs a logical AND between two boolean signals. The output of this device is true when both the inputs are true.

## ASSIGN

Software version : STANDARD, ADVANCED

---



ASSIGN creates a variable in memory and unconditionally assigns to it the value present at component input pin.

The associated variable name is specified in the REFERENCE parameter.

LadderWORK software handles integer unsigned 16 bits variables.

As written in IEC / CEI 1131-3 specification, we suggest, for this kind of variable, the use of the standard %MW prefix which means a memory word variable.

When a variable is created using ASSIGN it is public to the entire net.

It is possible to read the value assigned by an ASSIGN object using the READVAR device.

ASSIGN transfer without changes the value present on its input pin to the output pin.

### **Dialog Settings**

---

Parameter	Description
-----------	-------------

---

REFERENCE	This parameter specify the name of the variable.
-----------	--

### **Net Plugs**

---

Pin	Description
-----	-------------

---

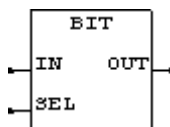
IN	Value that will be assigned to the variable
----	---

OUT	This pin gives the same value of the input pin
-----	--

**See also :** READVAR

## BIT

Software version : ADVANCED



This function block allow to extract a single bit from a word. The bit number is specified by the value applied to the SEL input and the word must be placed at input IN. The resulting boolean value will be available on the output named OUT.

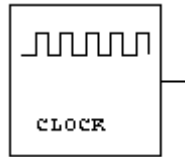
### **Net Plugs**

Pin	Description
IN	The word where the bit must be extracted
SEL	The bit selector
OUT	The resulting extracted bit

## CLOCK

Software version : STANDARD, ADVANCED

---



The CLOCK devices generates fixed frequency pulses. The frequency of the pulses can be programmed through the FREQUENCY parameter.

### **Dialog settings**

Parameter	Description
FREQUENCY	This parameter defines the frequency of the pulses. The minimum and maximum frequency depends by the used PLC. For further information about the timing resolution of the devices CLOCK and DELAY see TIMING RESOLUTION.

### **Net Plugs**

Pin	Description
OUT	Generated pulses are available on this output.



# CONST

Software version : STANDARD, ADVANCED



This device gives you the possibility to enter constant word values as input for many library devices available in the software. For example you can use CONST to define the count range for a CTU device applying this component to the PV ( programmed value ) input. The constant value is entered double-clicking on the object and configuring the VALUE parameter on the property dialog

## Dialog Settings

Parameter	Description
VALUE	The constant must be entered using this parameter.

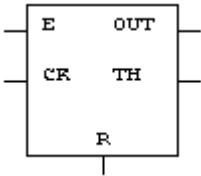
## Net Plugs

Pin	Description
OUT	The constant programmed word value will be available on this output.

**See also :** IDENT

# COUNTER

Software version : STANDARD, ADVANCED



The COUNTER device counts pulses applied to its CLOCK input. The device can be programmed to be UP counter or DOWN counter. The counter's counting is incremented or decremented on the raising edge of the CLOCK pulse when the enable pin E is asserted. At startup the counter is initialized to the value programmed on the BASE parameter. The counter proceeds with its counting until the THRESHOLD value is reached. When this is reached the threshold signal TH will become true. The next cycle will load the counter with the BASE value again.

The counter can be initialized by asserting a true signal into the R pin. When the R signal is asserted, the counter is loaded with the BASE value.

The maximum counting value for this device is 65535.

### Dialog Settings

Parameter	Description
BASE	The base value for the counting.
THRESHOLD	The threshold value for the counting.
UPDOWN	This parameter changes the direction of the counting.

### Net Plugs

Pin	Description
E	Counting enable pin.
CK	CLOCK input.
R	RESET input.
TH	THRESHOLD output
OUT	The counter value output

### Timing Diagrams

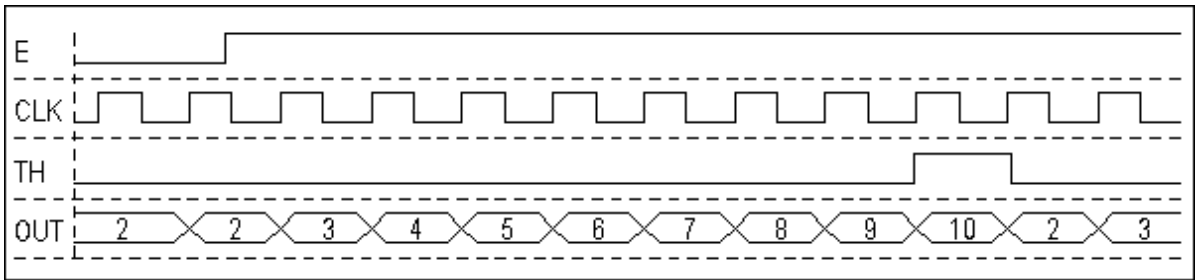


Figure53 - COUNTER Timing Diagram

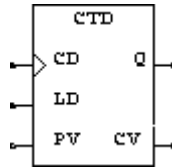
Diagrams refers to a COUNTER programmed with BASE=2 and THRESHOLD=10.

**See also :** CTU , CTD , CTUD

## CTD

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



The CTD object represents a DOWN COUNTER. A rising-edge on CD input will decrement the counting by one. The Q output become TRUE when the current counting value is equal or less than zero. Applying a TRUE signal on LD (LOAD) input will load the counter with the value present at input PV ( Asynchronous load ).

The CV output pin reports the current counting value.

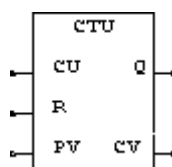
### Net Plugs

Pin	Description
CD	A rising-edge on this input will decrement the counter by one.
LD	Applying a TRUE signal on this input will load the counter with the value present at input LD.
PV	When the LD pin is asserted, the value applied to this pin will be <del>used</del> as current count value. User should use a CONST or IDENT object to enter numerical constant.
Q	This output become TRUE when the counting is equal or greater than zero.
CV	This output reports the current counting value.

**See also :** COUNTER, CTU, CTUD, IDENT, CONST

## CTU

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



The CTU object represents an UP COUNTER. A rising-edge on CU input will increment the counting by one. When the programmed value, applied to the input PV, is reached, the Q output become TRUE.

Applying a TRUE signal on R input will reset the counter to zero ( Asynchronous reset ).

The CV output pin reports the current counting value.

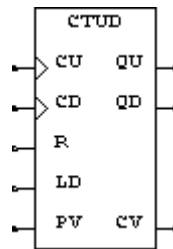
### Net Plugs

Pin	Description
CU	A rising-edge on this input will increment the counter by one.
R	Applying a TRUE signal on this input will reset the counter.
PV	The Q output become TRUE when the current counting value reaches the value applied to this input. User should use a CONST or IDENT object to enter numerical constant.
Q	This output become TRUE when the counting is equal or greater than the programmed value.
CV	This output reports the current counting value.

**See also :** COUNTER, CTD, CTUD, CONST, IDENT

## CTUD

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



This device represents an UP/DOWN programmable counter. A rising-edge on the CU ( COUNT-UP) input will increment the counter by one while a rising-edge on the CD ( COUNT-DOWN ) decreases the current value.

Applying a TRUE signal on R input will reset the counter to zero. A TRUE condition on the LD signal will load the counter with the value applied to the input PV ( PROGRAMMED VALUE ) .

QU output becomes active when the current counting value is greater or equal to the programmed value. The QD output becomes active when the current value is less or equal to zero.

The CV output reports the current counter value.

As specified in the IEC / CEI 1131-3 standard this kind of counter has a counting range expressed by an integer 16 bit variable. This means that this counter can span from -32768 to +32767 .

### Net Plugs

Pin	Description
CD	A rising-edge on this input will decrement the counter by one.
CU	A rising-edge on this input will increment the counter by one.
R	Applying a TRUE signal on this input will reset the counter.
LD	Applying a TRUE signal on this input will load the counter with the value present at input PV.
PV	When the LD pin is asserted, the value applied to this pin will be loaded as current count value. User should use a CONST or IDENT object to enter numerical constant.
QU	This output become TRUE when the counting is equal or greater than the programmed value.
QD	This output become TRUE when the counting is less or equal than zero.
CV	This output reports the current counting value.

**See also :** COUNTER, CTD, CTU, CONST, IDENT

# DEBOUNCE

Software version : BASE, STANDARD, ADVANCED



The DEBOUNCE device can be used in conjunction with the standard input device like INPUT, EINPUT, NCINPUT, ENCINPUT. The main function of this device is to eliminate the typical spikes/noises generated by a hardware switch or button.

The DEBOUNCE device computes a unitary integration in the programmed time. When the integration time is elapsed the output will become true if the computed value is greater than a pre/programmed threshold.

Typically, the main function of this device is to eliminate spikes and noises on PLC digital physical inputs.

## Dialog Settings

Parameter	Description
INTEGRATION TIME	This parameter ( expressed in milliseconds ) changes the filter integration time. For signals generated by switches a filtering time of 100 ms is sufficient.

## Net Plugs

Pin	Description
INPUT	Filter input
OUT	Filter output

# DEC1-8

Software version : ADVANCED

---



This function block represents a one to eight decoder. The boolean value applied to the input E is transferred to the output selected by the value applied to the input S.

**Net Plugs**

Pin	Description
S	The output selector
E	The boolean value that will be transferred to the selected output
0..7	The outputs



## DELAY

Software version : STANDARD, ADVANCED



The DELAY device generates delayed signals respect to the input signal. Two kinds of functionality are available : DELAY MODE or HOLD MODE. In DELAY MODE a pulse applied on its input generates a single pulse after the programmed time. In HOLD MODE a pulse on its input activates the output for all the programmed time.

There is the possibility to condition the behavior of the device for the pulses following the first trigger pulse. Using the NO RETTRIGERABLE option the pulse following the first trigger pulse will be ignored. With the RETTRIGERABLE option set, the elapsed time will be cleared every time a rising-edge is recognized on the input so the time-past event will take place starting from the last pulse.

DELAY device, like CLOCK and DEBOUNCE , are SYSTEM TIMER dependant.

For further information see TIMING RESOLUTION .

### Dialog Settings

Parameter	Description
DELAY TIME	Delay time expressed in seconds.
MODE	Selects HOLD MODE or DELAY MODE.
TRIGGER MODE	The response of the DELAY device to the pulses following the first pulse can be changed through this parameter.

### Net Plugs

Pin	Description
INPUT	Delay input. This pin is raising-edge sensitive.
OUT	The delay output.

### Timing Diagrams

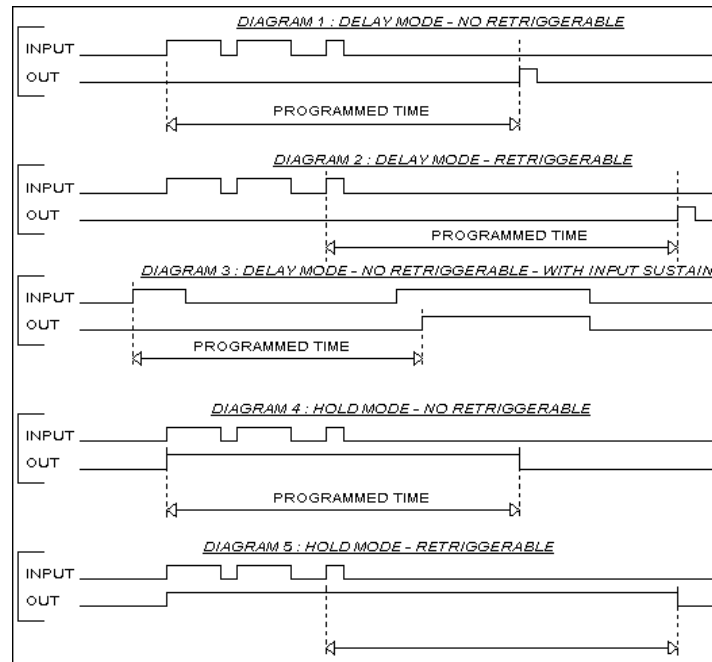
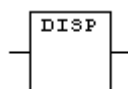


Figure54 - Delay Timing

**See also :** TP, TON, TOF, TMI, TSQ

# DISPLAY

Software version : ADVANCED



Display support is automatically activated placing a DISPLAY component. There are two main displaying modes called TWO-STATE MODE and PROBE MODE. In TWO-STATE MODE the software check for the value applied to the input signal of DISPLAY block and shows the *ASSERT MESSAGE* if the value is one (TRUE) or shows the *NOT ASSERT MESSAGE* if this value is zero (FALSE). In this way you can redirect particular message to panel on state changing. Message is displayed at the position configured in the parameters *X Coord* and *Y Coords*

Terminals with messages storing capability, can display a previous stored message through the fields *NOT ASSERT MESSAGE CODE* and *ASSERT MESSAGE CODE*. If its values are not zero the software uses its values to select a message inside the terminal memory. In this case the string placed on the *[NOT]* *ASSERT MESSAGE* fields are ignored.

PROBE MODE should be used to display numerical values. In this operating mode, first the system displays the message contained in the *ASSERT MESSAGE* field, then the software writes the numerical value supplied on its input. The numerical format may be one of the following : BOOLEAN, UNSIGNED INT, SIGNED INT with the possibility to choose the decimal or the hexadecimal notation.

## Dialog Settings

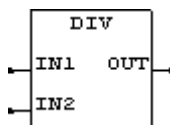
Parameter	Description
X-Coord,Y-Coord	These are the coordinate where the message will be located
MODE	Selects TWO-STATE MODE or PROBE-MODE
DISPLAY AS	Selects the decimal or hexadecimal notation
DATA TYPE	Selects the numerical format
FIELD LENGTH	Select the length of display field
ASSERT MESSAGE	The message entered in this field will be displayed in the ASSERT condition or as prefix when using PROBE MODE
NOT ASSERT MESSAGE	The message entered in this field will be displayed in the NOT-ASSERT condition
BELL	In same conditions, display kernel can generate bell signals
TERMINAL ID	The terminal identifier. ladderWORK software can handle up to four terminals at time
TERMINAL TYPE	This parameter should be used to select a terminal model
PARM#1	General purpose parameter # 1
PARM#2	General purpose parameter # 2

## Net Plugs

Pin	Description
IN	Data input
OUT	This pin reports the same value of the IN pin.

## DIV

Software version : ADVANCED



This function block divide the value applied to the input IN1 by the value applied to the input IN2 and the result of the division is available on the OUT pin.

### **Net Plugs**

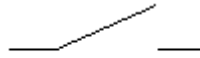
Pin	Description
IN1	Dividend
IN2	Divisor
OUT	Result of division

**See also :** Mathematical expressions

## EINPUT

Software version : BASE, STANDARD, ADVANCED

---



EINPUT represents the normally-open input device in electrical symbology. This component may represent a physical input of PLC or a logical input to associate with a RELAY device. To establish if an input is physical or logical opportune values must be programmed on the REFERENCE parameter of the property dialog. If the REFERENCE value refers to a physical resource of the PLC then a physical input will be created. If the REFERENCE field does not refer to any physical resources then the EINPUT will be configured as logical input to associate with a RELAY device. This configuration is also called LOGICAL LINKS .

There is no limit on the EINPUT devices that can be related with a RELAY device.

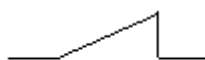
In general input devices perform a logical AND between the signal supplied as input and the switch signal. For switch signal we mean the signal that closes the switch ( logical or physical ). The output of this device will be TRUE if both the input and the switch signal are TRUE. The output will show FALSE everytime the switch is OPEN or the input is FALSE.

**See also :** INPUT , NCINPUT , ENCINPUT

## ENCINPUT

Software version : BASE, STANDARD, ADVANCED

---



ENCINPUT represents the normally-closed input device in electrical symbology. This component may represent a physical input of PLC or a logical input to associate with a RELAY device. To establish if an input is physical or logical opportune values must be programmed on the REFERENCE parameter of the property dialog. If the REFERENCE value refers to a physical resource

of the PLC then a physical input will be created. If the REFERENCE field does not refer to any physical resources then the EINPUT will be configured as logical input to associate with a RELAY device. This configuration is also called LOGICAL LINKS .

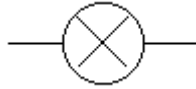
There's no limit on the EINPUT devices that can be relationed with a RELAY device.

**See also** : INPUT , EINPUT , NCINPUT

## EOUTPUT

Software version : BASE, STANDARD, ADVANCED

---



This object represents a generic output device in electrical symbology. This device can be associated to a physical output of the PLC or linked to other input in the schematic. The physical or logical property is opportunely configurable selecting a physical resource or not on the associated dialog. Through the REFERENCE field on the configuration dialog allows the user to select one of the available physical resources. If one of the these resources is selected then the device will be configured as physical output. Any other selection will produce a logical device to associate with components like INPUT , NCINPUT, EINPUT , ENCINPUT .

For further information about input-output device association see the section LOGICAL LINKS .

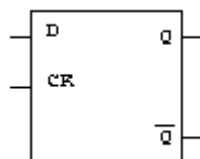
In detail, when a OUTPUT device is configured as logical object, a global variable is created and is public to all the net. The EOUTPUT device reply the signal present on its input on the output, so more EOUTPUT devices can be chained on the same rung.

**See also :** OUTPUT , RELAY



## FFD

Software version : STANDARD, ADVANCED



This component represents a D-TYPE FLIP-FLOP. These components, are not normally present on the LADDER standard symbology but are present to give power to the software.

The D-TYPE FLIP-FLOP represent the elementary memory cell on the logic circuits.

The behavior of the device is the following : The data present on the D input is frozen on the raising edge of the CK signal. The Q output reports the value of the last freeze cycle and the /Q signal reports the complement of the Q signal.

This component represents a D-TYPE FLIP-FLOP. These components, are not normally present on the LADDER standard symbology but are present to give power to the software.

The D-TYPE FLIP-FLOP represent the elementary memory cell on the logic circuits.

The behavior of the device is the following : The data present on the D input is frozen on the raising edge of the CK signal. The Q output reports the value of the last freeze cycle and the /Q signal reports the complement of the Q signal.

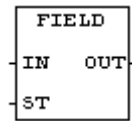
### Net Plugs

Pin	Description
D	Boolean data input
CK	CLOCK signal
Q	Direct output
/Q	Negated output

## FIELD

Software version : ADVANCED

---



The FIELD component allow user to enter data using an operator panel. When the field is selected, user is able to enter a decimal or boolean number using the keypad.

The field entry area is preceded by a prompt string that user may enter through the apposite text box.

If the ST pin is active then the value applied to the IN pin will be copied in the FIELD.

*NOTE : To enable the data entry for a particular terminal, remember to apply a logical ONE in the INPUT pin of a KEYBCTRL device programmed with the same TERMINAL IDENTIFICATOR.*

### Dialog Settings

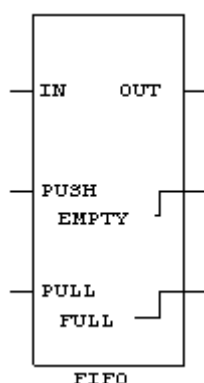
Parameter	Description
X Coord	X Coordinate where the field will be located on the panel
Y Coord	Y Coordinate where the field will be located on the panel
Data Type	This field select the numeric data type
Prompt	The string entered in this box will precede the entry area
Field Size	User can limit the entry area using this field
Terminal ID	Since up to four terminals can be handled by LadderWORK kernel, this parameter selects which terminal will handle the field

### Net Plugs

Pin	Description
IN	Data input.
ST	The data applied to the IN pin will be stored into the field when this pin become TRUE
OUT	The current numeric value is available from this pin

## FIFO

Software version : ADVANCED



This device handles a queue of data using the FIFO ( FIRST IN FIRST OUT ) method. This kind of data structure is known also as CIRCULAR QUEUE. The FIFO device allows you to store a lot of data and retrieve it in reverse order. The queue can handle words of 8 or 16 bits with user definable stack-depth. Keep in mind that in some versions of the software the maximum size of the queue is function of the current memory model. For further information about this argument refer to the section MEMORY MODELS .

The FIFO functionality is explained below :

The data present on the IN plug is inserted in the queue on the raising-edge of the PUSH signal. The OUT signal always gives the value of the first data available. Data can be sequentially retrieved in reverse order applying signals on the PULL signal. When the PULL signal is applied the OUT plug will give the value for the next data available. In the case of the queue being empty the OUT plug will supply zero. Two pins are available to check the FIFO status. The EMPTY signal is true when the queue is empty and the FULL signal is true when the queue is FULL.

PUSH and PULL signals are sampled simultaneously. If a transition is detected on both the signals, PUSH has precedence respect to PULL.

### Dialog Settings

Parameter	Description
DEPTH	The queue depth
DATA SIZE	The word size for the queue ( BYTE / WORD )

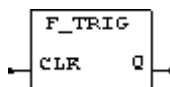
### Net Plugs

Pin	Description
IN	Data input.
OUT	Data output.
PUSH	This signal pushes data in the queue.
PULL	This signal pulls data from the queue.
FULL	This signal becomes true when the queue is full.
EMPTY	This signal becomes true when the queue is empty.

**See also :** LIFO

## F\_TRIG

Software version : STANDARD, ADVANCED  
CEI / IEC 1131-3 Compliant



This device is a rising-edge detector. The Q output become TRUE when a 0 to 1 ( or FALSE to TRUE or OFF to ON ) condition is detected on the CLK input and it sustain this state for a complete scan cycle.

### Net Plugs

Pin	Description
CLK	The rising-edge detector input
Q	When a rising-edge is detected this output become true for a single scan cycle

## IDENT

Software version : BASE, STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



IDENT allow to enter literals and identifiers using the IEC 1131-3 standard notation

### **Base 2, base 8 and base 16 notation**

Numbers in base 2,8 and 16, start respectively with the prefix 2#, 8# and 16# . The number, expressed in the specified base follow. For clarity, the underscore character '\_' could be used to separate part of the number. This is useful, for example, for binary numbers.

### **Time duration notation**

Time duration literals begin with the prefixes TIME# or time# or T# or t#. A time specification string follow. The time specification string is formed by one or more numbers followed by a time-unit suffix. For clarity, the underscore character '\_' could be used to separate part of the time string.

<b>Time unit suffix</b>	<b>Meaning</b>
ms	milliseconds
s	seconds
m	minutes
h	hours
d	days

## Examples

Syntax	Result
2#10000001	Hex 81, Decimal 129
2#1000_0001	Hex 81, Decimal 129
8#10	Octal 10, Decimal 8
16#FFFF	Hex FFFF, Decimal 65535
-276	Integer Decimal -276
65535	Integer Decimal 65535
+10000	Integer Decimal 10000
TRUE	TRUE condition, Boolean 1, Logical one
FALSE	FALSE condition, Boolean 0, Logical zero
TIME#14.73ms	Time duration specification 14.73 milliseconds
time#14.73ms	Time duration specification 14.73 milliseconds
t#14.73ms	Time duration specification 14.73 milliseconds
T#14.73ms	Time duration specification 14.73 milliseconds
t#6m_34s_15ms	Time is : 6 minutes, 34 seconds and 15 milliseconds
time#1h30m	Time is : 1 hour and half
TIME#4d_5h_34m_10s_25ms	Time is : 4 days, 5 hours, 34 minutes, 10 seconds and 25 milliseconds

## Dialog Settings

Parameter	Description
REFERENCE	The literal or the identifier is entered using this field

## Net Plugs

Pin	Description
OUT	The value of the literal/identifier is available from this pin

## INCLUDE

Software version : BASE, STANDARD, ADVANCED

---



The INCLUDE block allow user to add an assembler file to the project. This method is used to supply to the compiler particular routines for dedicated I/O and other functions.



# INPUT

Software version : BASE, STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



INPUT represents the standard normally-open input device in the LADDER symbology. This component may represent a physical input of PLC or a logical input to associate with a RELAY device. To establish if an INPUT is physical or logical opportune value must be programmed on the REFERENCE parameter of the property dialog. If the REFERENCE value refers to a physical resource of the PLC then a physical INPUT will be created. If the REFERENCE field does not refer to any physical resources then the INPUT will be configured as logical input to associate with a RELAY device. This configuration is also called LOGICAL LINKS .

There's no limit on the INPUT devices that can be related with a RELAY device.

**See also:** EINPUT , NCINPUT , ENCINPUT

## IPIN

Software version : BASE, STANDARD, ADVANCED

---



IPIN represents a microprocessor's input pin. This device gives out the value of the pin.

### **Dialog Settings**

Parameter	Description
REFERENCE	This parameter will identify a particular pin of the microprocessor.

### **Net Plugs**

Pin	Description
OUT	This output gives out the value of the desired pin.

**See also :** OPIN

## KEYBCTRL

Software version : ADVANCED



This device allow to control the flow of the data coming from the keyboard. If the INPUT pin is ONE then the system redirect the keyboard messages to the field currently under focus. If this pin is ZERO, the system redirect the keyboard messages to the KEYBOARD function block. The keyboard flow-switching is independant for each terminal. The OUT pin simply reports the value of the INPUT pin.

### Dialog Settings

Parameter	Description
MODE	Selects NORMAL or BISTABLE mode
KEY_CODE	Selects the switch on the key-pad
TERMINAL_ID	Selects one of the four terminals handled by the system
OPTION	Reserved for future use
PARM#1	Reserved for future use
PARM#2	Reserved for future use

### Net Plugs

Pin	Description
IN	This pin, if TRUE, enables the KEYBOARD component
OUT	This output gives out the value of the desired switch

## KEYBOARD

Software version : ADVANCED

---



This component, reads the press/release state of a switch on the keypad of a user terminal. The current key state is reported on the right pin (OUT) only if the left pin (IN) is TRUE.

Two behavior models can be programmed to be NORMAL or BISTABLE. In normal mode, the system just reads the value of the switch. In BISTABLE mode the OUT pin will toggle ON and OFF every time the switch is pressed.

The switch on the keyboard is identified through the KEY\_CODE parameter. Refer to the appropriate manufacturer documentation for this information.

### **Dialog Settings**

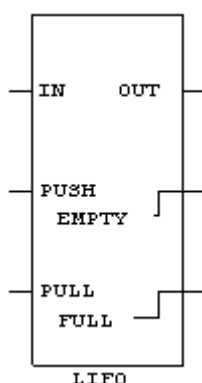
Parameter	Description
MODE	Selects NORMAL or BISTABLE mode
KEY_CODE	Selects the switch on the key-pad
TERMINAL_ID	Selects one of the four terminals handled by the system
OPTION	Reserved for future use
PARM#1	Reserved for future use
PARM#2	Reserved for future use

### **Net Plugs**

Pin	Description
IN	This pin, if TRUE, enables the KEYBOARD component
OUT	This output gives out the value of the desired switch

# LIFO

Software version : ADVANCED



This device handles a queue of data using the LIFO ( LAST IN FIRST OUT ) method. This kind of data structure is know also as STACK. The LIFO device allows you to store a lot of data and retrieve it in same sequence. The queue can handle words of 8 or 16 bits with user definable stack-depth. Keep in mind that in some version of the software the maximum size of the queue is function of the current memory model. For further information about this argument refer to the section MEMORY MODELS.

The LIFO functionality is explained below :

The data present on the IN plug is inserted in the queue on the raising-edge of the PUSH signal. The OUT signal always gives the value of the first data available. Data can be sequentially retrieved applying signals on the PULL signal. When the PULL signal is applied the OUT plug will give the value for the next data available. In case of empty-queue the OUT plug will suply zero. Two pins are available to check the LIFO status. The EMPTY signal is true when the queue is empty and the FULL signal is true when the queue is FULL.

PUSH and PULL signal are sampled simoultaineously detected on both of the signals. PUSH has the precedence respect to PULL.

## Dialog Settings

Parameter	Description
-----------	-------------

DEPTH	The queue depth
-------	-----------------

### Net Plugs

Pin	Description
-----	-------------

IN	Data input.
----	-------------

OUT	Data output.
-----	--------------

PUSH	This signal pushes data in the queue.
------	---------------------------------------

PULL	This signal pulls data from the queue.
------	--

FULL	This output becomes true when the queue is full.
------	--

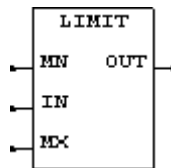
EMPTY	This output becomes true when the queue is empty.
-------	---

**See also :** FIFO

## LIMIT

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



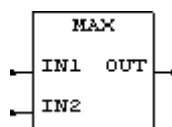
This function block compares the value applied to the input IN with the values applied to the inputs MN ( MINIMUM VALUE ) and MX ( MAXIMUM VALUE ). If the input value is less than the value applied to the MN input then the output reports the value of MN pin. If the input value is greater than the value applied to the MX input then the output gives the value applied to the MX pin. When the value applied to the input is inside the two limits the value is transferred to the output without limitations.

### **Net Plugs**

Pin	Description
MN	Minimum allowable value
MX	Maximum allowable value
IN	Value to be limited
OUT	Limited value

## MAX

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



This function block compares the magnitude of the values present at input IN1 and IN2 reporting on its output the largest value ( maximum value ).

### Net Plugs

Pin	Description
IN1	First operand
IN2	Second operand
OUT	Maximum value

## MBCONF

Software version : ADVANCED

---

*FUTURE IMPLEMENTATION*



## MBIN

Software version : ADVANCED

---

*FUTURE IMPLEMENTATION*

## ABOUT

Software version : ADVANCED

---

*FUTURE IMPLEMENTATION*

# MBSLAVE

Software version : ADVANCED

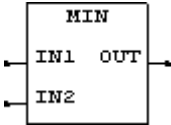
---

*FUTURE IMPLEMENTATION*

# MIN

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



This function block compares the magnitude of the values present at input IN1 and IN2 reporting on its output the smallest value ( minimum value ).

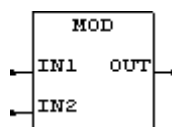
## Net Plugs

Pin	Description
IN1	First operand
IN2	Second operand
OUT	Minimum value

## MOD

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



This function block gives the remainder, result of division of the value applied to the input IN1 by the value applied to the input IN2. The computed value is available on the OUT pin.

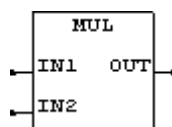
### Net Plugs

Pin	Description
IN1	The dividend
IN2	The divisor
OUT	Remainder ( MODULE ) of the division

## MUL

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



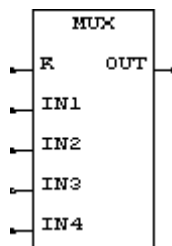
This device multiplies the values applied to the pins IN1 and IN2 giving the result on the OUT pin.

### **Net Plugs**

Pin	Description
IN1	First operand
IN2	Second operand
OUT	Result of multiplication

## MUX

Software version : ADVANCED



This function block selects one of the four possible values applied to the inputs IN1..IN4 transferring the value to the OUT pin. The selection of the input is performed through the K input. A zero on K pin selects the IN1, a value equal to three selects the input IN4.

### Net Plugs

Pin	Description
K	The selector
IN1..IN4	Multiplexer inputs
OUT	This pin gives the value of the selected word

## NCINPUT

Software version : BASE, STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



NCINPUT represents the normally-closed input device in LADDER symbology. This component may represent a physical input of PLC or a logical input to associate with a RELAY device. To establish if an input is physical or logical opportune values must be programmed on the REFERENCE parameter of the property dialog. If the REFERENCE value refers to a physical resource of the PLC then a physical input will be created. If the REFERENCE field does not refer to any physical resources then the EINPUT will be configured as logical input to associate with a RELAY device. This configuration is also called LOGICAL LINKS .

There is no limit on the NCINPUT devices that can be related with a RELAY device.

**See also** INPUT , EINPUT , ENCINPUT



# NOT

Software version : STANDARD, ADVANCED

---



The NOT device performs a data inversion of the signals input. If the input is true the output is false and viceversa.

## OPIN

Software version : BASE, STANDARD, ADVANCED

---



OPIN represents a microprocessor's output pin. The boolean value applied to the input of this device is transferred to the microprocessor physical pin.

### **Dialog Settings**

---

Parameter	Description
-----------	-------------

---

REFERENCE This parameter will identify a particular pin of the microprocessor.

### **Net Plugs**

---

Pin	Description
-----	-------------

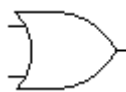
---

INPUT The boolean value applied to this signal will be transferred to the microprocessor pin.

## OR

Software version : STANDARD, ADVANCED

---



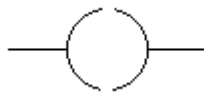
The OR device performs a logical OR between two boolean signals. The output become true when at least one input is true.

**See also :** AND, NOT

## OUTPUT

Software version : BASE, STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



This object represents a generic output device. This device can be associated to a physical output of the PLC or linked to other inputs in the schematic. The physical or logical property is opportunely configurable selecting a physical resource or not on the associated dialog. Through the REFERENCE field on the configuration dialog allows the user to select one of the available physical resources. If one of the these resources is selected then the device will be configured as physical output. All the other selections will produce a logical device to associate with components like INPUT , NCINPUT , EINPUT , ENCINPUT .

For further information about input-output device association see the section LOGICAL LINKS .

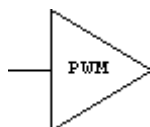
In detail, when a OUTPUT device is configured as a logical object, a global variable is created and is public to all the net. The OUTPUT device will reply the signal present on its input on the output, so more OUTPUT devices can be chained on the same rung.

**See also :** EOUTPUT , RELAY

## PWMOUT

Software version : ADVANCED

---



This device, available only for some PLC models, sets an hardware PWM converter, or a generic D/A interface, with the value applied to the input PIN.

The input value must be normalized in the range 0 +65535 which means that zero will give the minimum analog value while 65535 gives the maximum analog value. There are two parameters that control the D/A dynamic : OFFSET and SPAN.

With OFFSET you add a base constant to the value applied to the input. The SPAN parameter allow you to change the converter gain. With values equal to 1.000 no modify will be applid to the input. Values greater than one will increase the gain while valued less than one will produce attenuation.

### Dialog Settings

Parameter	Description
OFFSET	This parameter adds a OFFSET value to the input value
SPAN	This parameter changes the gain of the converter.

### Net Plugs

Pin	Description
IN	Data input (normalized 0-65535).

## QTP\_DSPY

Software version : ADVANCED

---

LadderWORK can support GRIFO's user terminal models QTP16, QTP22 and QTP24. Supporting it is activated selecting GPC553 model during project set-up. QTP16 panel uses the GPC553 CN5 connector so user projects can't use the microprocessor ports P1[0..7] P4[0..7]. QTP Kernel is automatically installed placing a QTP component into the schematic ( QTP\_DSPY or QTP\_KEYB ).

Using QTP panels model QTP 22 or QTP 24, serial communication line available on GPC 553 is connected to the panel, so the automatic upload feature and other remote commands like PLAY and STOP are lost .

### **Using QTP panels with LadderWORK**

As said above, QTP support is automatically activated placing a QTP\_DSPY or QTP\_KEYB component. The working modes are very similar for both the models. There are two main displaying modes called NORMAL MODE and PROBE MODE. In NORMAL MODE the software check for the value applied to the input signal of QTP\_DSPY block and shows the *ASSERT MESSAGE* if the value is one (TRUE) or shows the *NOT ASSERT MESSAGE* if this value is zero (FALSE). In this way you can redirect particular message to panel on state changing. Message is displayed at the position configured in the parameters *X Coord* and *Y Coord*s

In QTP 22 and QTP 24 user can display previous stored messages through the fields *NOT ASSERT MESSAGE CODE* and *ASSERT MESSAGE CODE* . If its values are not zero the software uses its values to select a message inside the terminal memory. In this case the string placed on the *[NOT] ASSERT MESSAGE* fields are ignored.

PROBE MODE should be used to display numerical values. In this operating mode, first the system displays the message contained in the *ASSERT MESSAGE* field, then the software writes the numerical value supplied on its input. The numerical format may be one of the following : BOOLEAN, HEXADECIMAL, DECIMAL.

## QTP\_KEYB

Software version : ADVANCED

---

### **Using QTP Keyboards with LadderWORK**

QTP Keyboards may be used simply by placing a QTP\_KEYB object in the schematic. This device on its output gives the state of the programmed key. For the *Key Code* user should refer to the appropriate documentation. There are two main operating modes. In normal mode the component has different behaviour due to the selected model. For QTP 16 device, the output is asserted for all the time the key is pressed. For QTP 22/24 devices the output is asserted at pressing just for one pulse. In bistable mode, all the QTP models have the same behavior. In this operating mode the output is asserted or de-asserted alternatively pressing the configured key (Toggle mode).

## READVAR

Software version : STANDARD, ADVANCED

---



This device reads the numerical value relative to the associated variable that was created using the ASSIGN object.

The name of the variable, that must be supplied through the REFERENCE parameter must be a variable name that exists in the net.

With the using of the pair ASSIGN / READVAR it is possible to transfer numerical values from one point to another in the net.

### **Dialog Settings**

---

Parameter	Description
-----------	-------------

---

REFERENCE	This parameter specify the name of the associated variable
-----------	--

### **Net Plugs**

---

Pin	Description
-----	-------------

---

OUT	This pin gives the value of the associated variable
-----	---

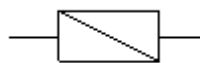
**See also :** ASSIGN



## RELAY

Software version : BASE, STANDARD, ADVANCED

---



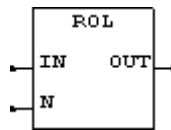
This object always represents a logical device. The RELAY device has a behavior analogue to electromechanical RELAYS of electrical plants. The RELAY device must be used in conjunction with INPUT, NCINPUT, EINPUT, ENCINPUT devices. For further information about this argument, see the section LOGICAL LINKS. In details, the RELAY device creates a global boolean variable that can be used by the entire net. The RELAY device transports the value of the input to its output so more than a RELAY can be cascaded together.

**See also :** EOUTPUT, OUTPUT

## ROL

Software version : ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



This function rotates left the word applied to the input IN giving the result on the OUT pin. The number of rotates is specified by the value applied to the pin N. During every single rotation the most significant bit of the word is transferred to the less significant bit.

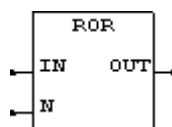
### **Net Plugs**

Pin	Description
IN	The word to be rotated
N	The number of rotation to be performed
OUT	The resulting word

**See also :** ROR, SHL, SHR

## ROR

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



This function rotates right the word applied to the input IN giving the result on the OUT pin. The number of rotations is specified by the value applied to the pin N. During every single rotation the less significant bit of the word is transferred to the most significant bit.

### Net Plugs

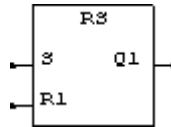
Pin	Description
IN	The word to be rotated
N	The number of rotations to be performed
OUT	The resulting word

**See also :** ROL, SHL, SHR

## RS

Software version : STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



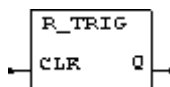
This function represents a standard reset-dominant set/reset flip flop. The Q1 output become TRUE when the input S is TRUE and the R1 input is FALSE. In the same way, the Q1 output become FALSE when the input S is FALSE and the R1 input is TRUE. After one of these transitions, when both the S and R1 signals return to FALSE, the Q1 output keeps the previous state until a new condition occurs. If you apply a TRUE condition for both the signals, the Q1 output is forced to FALSE ( reset-dominant ).

### **Net Plugs**

Pin	Description
S	The SET input
R1	The RESET-DOMINANT input
Q1	The FLIP-FLOP output

## R\_TRIG

Software version : STANDARD, ADVANCED  
CEI / IEC 1131-3 Compliant



This device is a rising-edge detector. The Q output become TRUE when a 0 to 1 ( or FALSE to TRUE or OFF to ON ) condition is detected on the CLK input and it sustain this state for a complete scan cycle.

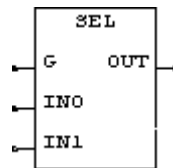
### **Net Plugs**

Pin	Description
CLK	The rising-edge detector input
Q	When a rising-edge is detected this output become true for a single scan cycle

## SEL

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



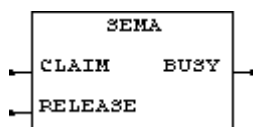
This function block selects one of the two possible values applied on the inputs IN0 and IN1 transferring the value to the OUT pin. The selection of the input is performed through the G input. Placing a zero on the G input will select the value applied to IN0 else the selected value will be IN1.

### Net Plugs

Pin	Description
IN0	Input # 0
IN1	Input # 1
G	The selector pin
OUT	This pin reports the value of the selected pin

## SEMA

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



This function block implements a semaphore function. Normally this function is used to synchronize events. The BUSY output is activated by a TRUE condition on the CLAIM input and it is de-asserted by a TRUE condition on the RELEASE input.

### **Net Plugs**

Pin	Description
CLAIM	The CLAIM input
RELEASE	The RELEASE input
BUSY	The BUSY output

## SEVENSEG

Software version : BASE, STANDARD, ADVANCED

---

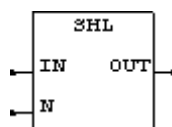
The SEVENSEG device can be applied only in the TECLAB PLC552 logic controller. This component will show the numerical value applied on its input in the seven segment display present on the board.

The system can display all the digits from 0 to 9 and the hexadecimal digits A to F.



## SHL

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant



This function perform a logical left shift of the value applied to the input IN and the resulting word is available on the OUT pin. The number of shifts is specified by the value applied to the pin N. The most significant bit of the word is filled with zero.

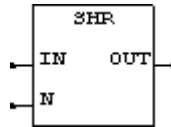
### **Net Plugs**

Pin	Description
IN	The word to be shifted
N	The number of shifts to be performed
OUT	The resulting word

## SHR

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



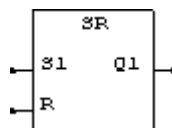
This function perform a logical right shift of the value applied to the input IN and the resulting word is available on the OUT pin. The number of shifts is specified by the value applied to the pin N. The less significant bit of the word is filled with zero.

### **Net Plugs**

Pin	Description
IN	The word to be shifted
N	The number of shifts to be performed
OUT	The resulting word

## SR

Software version : STANDARD, ADVANCED  
CEI / IEC 1131-3 Compliant



This function represents a standard set-dominant set/reset flip flop. The Q1 output become TRUE when the input S1 is TRUE and the R input is FALSE. In the same way, the Q1 output become FALSE when the input S1 is FALSE and the R input is TRUE. After one of these transitions, when both the S1 and R signals return to FALSE, the Q1 output keeps the previous state until a new condition occurs. If you apply a TRUE condition for both the signals, the Q1 output is forced to TRUE ( set-dominant ).

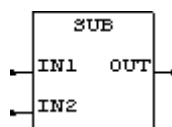
### Net Plugs

Pin	Description
S1	The SET DOMINANT input
R	The RESET input
Q1	the FLIP-FLOP output

## SUB

Software version : ADVANCED  
CEI / IEC 1131-3 Compliant

---



This device subtract the value present at IN1 from the value present at input IN2 giving the result on the OUT pin.

### **Net Plugs**

Pin	Description
IN1	First operand
IN2	Second operand
OUT	Result of subtraction

**See also :** Mathematical expressions

## THRESHLD

Software version : STANDARD, ADVANCED



The threshold device compares the magnitude of the value present in its input with a pre-programmed value. The value of the output (TRUE/FALSE) is conditioned by the result of this compare according to the QUALIFIER parameter.

### Meaning of QUALIFIER parameter

=	The output becomes true when the input value is equal to the programmed value
<>	The output becomes true when the input value is not equal to the programmed value
>=	The output becomes true when the input value is greater or equal to the programmed value
<=	The output becomes true when the input value is less or equal to the programmed value
>	The output becomes true when the input value is greater than the programmed value
<	The output becomes true when the input value is less than the programmed value

### Dialog Settings

Parameter	Description
COMPARE VALUE	The value to be compared with the input value
QUALIFIER	This parameter will establish the comparing type

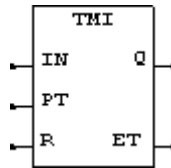
### Net Plugs

Pin	Description
INPUT	The comparator input
OUT	The comparator output

## TMI

Software version : ADVANCED

---



This device, also called INTEGRAL TIMER, accumulates the sum of the time for the periods where the IN input is in assert state. This means that the computed time is the sum of the ON times of the IN input.

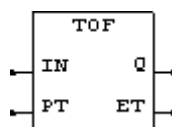
If the computed sum reaches the programmed time, applied to the PT input, the Q output become TRUE. At this point, the only way to reset the integral timer can be performed asserting the R ( RESET ) input. The ET output pin reports the current elapsed time.

### Net Plugs

Pin	Description
IN	The timer accumulates the time when this input is TRUE.
R	The integral timer must be resetted using this input
PT	To this input user must apply the programmed time. User should use an IDENT object to enter a time constant.
Q	After that the pre-programmed time threshold is reached, this output become TRUE.
ET	This output reports the current elapsed time.

## TOF

Software version : STANDARD, ADVANCED  
CEI / IEC 1131-3 Compliant



Asserting the input signal IN of this device immediately activates the Q output. At this point, releasing the input IN will start the time elapsing. When the programmed time, applied to the input PT, is elapsed and the input IN is still de-asserted, the Q output become FALSE. This condition will be kept until the input IN is remains de-asserted.

If the IN input is asserted again before time elapsing, the time counting will be cleared and the Q output remains ON.

The ET output pin reports the current elapsed time.

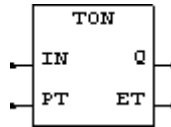
### **Net Plugs**

Pin	Description
IN	A TRUE condition on this input starts the TON timer.
Q	If the input IN is asserted and the programmed time is elapsed this output become TRUE.
PT	To this input user must apply the programmed time. User should use an IDENT object to enter a time constant.
ET	This output reports the current elapsed time.

## TON

Software version : STANDARD, ADVANCED  
*CEI / IEC 1131-3 Compliant*

---



Asserting the input signal IN of this device starts the time elapsing of timer. When the programmed time, applied to the input PT, is elapsed and the input IN is still asserted, the Q output become TRUE. This condition will continue until the input IN is deasserted.

If the IN input is released before time elapsing, the timer will be cleared.

The ET output pin reports the current elapsed time.

### **Net Plugs**

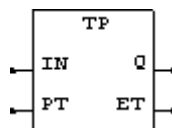
Pin	Description
IN	A TRUE condition on this input starts the TON timer.
Q	If the input IN is asserted and the programmed time is elapsed this output become TRUE.
PT	To this input user must apply the programmed time. User should use an IDENT object to enter a time constant.
ET	This output reports the current elapsed time.



## TP

Software version : STANDARD, ADVANCED  
CEI / IEC 1131-3 Compliant

---



This kind of timer has the same behavior of a single-shot timer or a monostable timer.

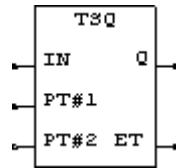
When a rising-edge ( OFF to ON or FALSE to TRUE ) transition is detected on the IN input, the Q output become immediately TRUE .This condition continue until the programmed time PT, applied to the relative pin, is elapsed. After that the programmed time is elapsed, the Q output keeps the ON state if the input IN is still asserted else the Q output returns to the OFF state. This timer is not re-triggerable. This means that after the timer started it can't be stopped until the complete session ends. The ET output pin reports the current elapsed time.

### **Net Plugs**

Pin	Description
IN	A rising-edge on this pin will trigger the pulse timer.
Q	When the pulse timer starts, and before that the programmed time is elapsed, this output become TRUE.
PT	To this input user must apply the programmed time. User should use an IDENT object to enter a time constant.
ET	This output reports the current elapsed time.

# TSQ

Software version : ADVANCED



This device, also called SQuare wave timer, allow to generate square waves with variable duty-cycle. When the input is asserted, the time elapsing proceeds. When the elapsed time reaches the pre-programmed threshold, applied to the input PT#1, the Q output become TRUE. Time elapsing continue until the second pre-programmed threshold, applied to the input PT#2 is reached. When this condition become true, the Q output is putted to OFF state and the elapsed time is cleared so a new cycle could begin.

De-asserting the IN input will freeze the timer in the last condition. Applying a ON state again on the IN input will start the timer at the same point where it was left.

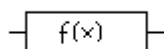
The ET output pin reports the current elapsed time.

## Net Plugs

Pin	Description
IN	The timer accumulates the time when this input is TRUE.
PT#1	The Q output become TRUE when the elapsed time reaches the value applied to this input. User should use an IDENT object to enter a time constant.
PT#2	The Q output become FALSE when the elapsed time reaches the value applied to this input. User should use an IDENT object to enter a time constant.
Q	This output is function of the pre-programmed time PT#1 and PT#2.
ET	This output reports the current elapsed time.

## USER1

Software version : ADVANCED



USER1 is an user definable function. The function can accept one input parameter, a constant programmable value that always gives a single output. User functions must be written in assembly language and must respect the system conventions. For further information about user functions see the section INTERFACING WITH ASSEMBLER .

### Dialog Settings

Parameter	Description
Source file	The assembler source file that contains the user function.
Function name	The name of the function. The assembler source code must contain a function with the same name.
Programmable value	This field contains a constant value that is transfered to the function every time the function is called.
INPUT Size	This parameter will establish the size for the input INPUT

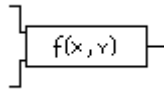
### Net Plugs

Pin	Description
INPUT	Data input . This pin can be programmed to be BYTE or WORD .

## USER2

Software version : ADVANCED

---



USER2 is a user definable function. The function can accept two input parameters, a constant programmable value that always gives a single output. User functions must be written in assembly language and must respect the system conventions. For further information about user functions see the section INTERFACING WITH ASSEMBLER .

### **Dialog Settings**

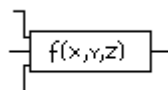
Parameter	Description
Source file	The assembler source file that contains the user function.
Function name	The name of the function. The assembler source code must contain a function with the same name.
Programmable value	This field contains a constant value that is transferred to the function every time the function is called.
IN1 Size	This parameter will establish the size for the input IN1
IN2 Size	This parameter will establish the size for the input IN2

### **Net Plugs**

Pin	Description
IN1	Data input # 1 . This pin can be programmed to be BYTE or WORD .
IN2	Data input # 2 . This pin can be programmed to be BYTE or WORD .

## USER3

Software version : ADVANCED



USER3 is a user definable function. The function can accept three input parameters, a constant programmable value that always gives a single output. User functions must be written in assembly language and must respect the system conventions. For further information about user functions see the section INTERFACING WITH ASSEMBLER .

### Dialog Settings

Parameter	Description
Source file	The assembler source file that contains the user function.
Function name	The name of the function. The assembler source code must contain a function with the same name.
Programmable value	This field contains a constant value that is transferred to the function every time the function is called.
IN1 Size	This parameter will establish the size for the input IN1
IN2 Size	This parameter will establish the size for the input IN2
IN2 Size	This parameter will establish the size for the input IN3

### Net Plugs

Pin	Description
IN1	Data input # 1 . This pin can be programmed to be BYTE or WORD .
IN2	Data input # 2 . This pin can be programmed to be BYTE or WORD .
IN3	Data input # 3 . This pin can be programmed to be BYTE or WORD .



## **SECTION 5 - MATHEMATICAL EXPRESSIONS**

### 5.1 - Entering formulas using function block format

Formulas can be entered in the schematic using the FUNCTION BLOCK notation. The figure below, for example, represents the way to compute the formula  $(5 + 3) * (9 + 2) - 7$ .

LadderWORK software supplies the classical arithmetic functions like ADD, SUB, DIV, MUL and MOD. Moreover the software gives you the possibility to operate with the logical operators like SHR, SHL, ROL, ROR and BIT.

Constant values can be entered using the CONST and IDENT components

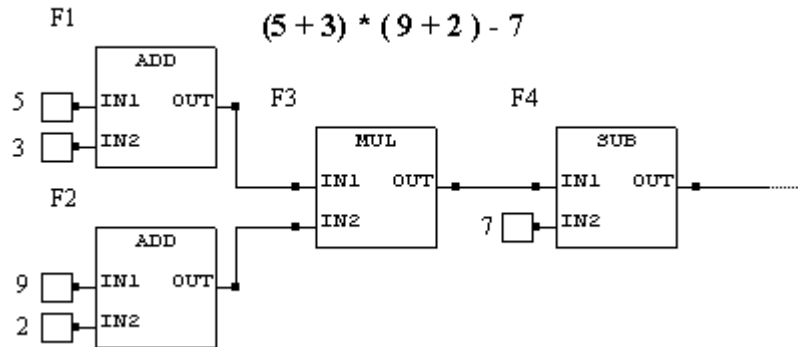


Figure55 - Expression sample



## **SECTION 6 - Interfacing with assembler**

## 6.1 - Interfacing with assembler using user functions

Sometimes, in a project, you can have the need to create custom components. LadderWORK gives the possibility to create single output functions with one up to three inputs. These functions are called USER1, USER2 and USER3. There is not limit about the number of the user functions that you can define in a project.

User functions can be programmed to accept BYTE or WORD data on its inputs.

In order to use user functions, you simply place a USER1, USER2 or USER3 component in the schematic. After placement you have to configure some element like the assembler file associated with the component and the name of the function called by the system. Another parameter is available to the function, this is called PROGRAMMABLE VALUE and gives the possibility to have variable values.

Writing user function implicate the knowing of microprocessor assembly language. Other precautions must be taken to avoid conflits with LadderWORK kernel. LadderWORK generated code is similar to a loop where input signals, functions and output signals are evaluated sequentially. The typical requirement of these systems is to make a complete cycle inside a 2/3 ms period which means that the code must be fast as possible. According to this user functions can't brake the flow. User functions must be short and fast. Moreover, user must avoid to use function's name like DELAY, FIFO etc because these names conflict with the kernel.

Remember, that in LadderWORK you haven't debug capability, so we advice to check first the routine with other tools and finally link it with the project.

### User function calling conventions for the 8051 family

Signals applied to a user function are passed through registers.

#### Format for the register data storage

Size	Registers
BYTE	R7, R5, R3, R1
WORD	R6R7, R4R5, R2R3, R0R1

We assume that registers R7, R5, R3, R1 always contain the complete value if a variable type is BYTE or the low part of the variable if this is WORD. Registers R6, R4, R2, R0 always contain the high byte of a WORD variable.

#### Output variable

The output value, or return value, always is WORD-sized and is placed in the register pair BA. The B register contains the high byte of the word and the A register ( ACCUMULATOR ) the low word.

#### Input-Registers association

Signal	Registers	
	BYTE	WORD
INPUT1	R7	R6R7
INPUT2	R5	R4R5
INPUT3	R3	R2R3

The programmable value will always receive the value in the register pair R0R1.

#### Segment's names

For the user function writing, user must respect the segment name conventions.

Nome	Description
------	-------------

CSEG	Code segment (ROM)
CONST	Constant segment (ROM)
IDATA	Internal data area (RAM)
XDATA	External data area (RAM)
IZDATA	Internal data area zeroed at startup (RAM)
IXDATA	External data area zeroed at startup (RAM)

**Predefined constants**

Name	Description
DATA8051	0 for SMALL models with internal data addressing
DATA8051	1 for LARGE models with external data addressing

**8051 user function example**

This example represents how to create a two word equality comparator using the two inputs user function. This object should be used to compare two word signals that gives a boolean value ( inside a WORD ) on its output. Refer to **fx\_comp.pjn** project for using template. The USER2 function must be configured to accept words on both the inputs. On the configuration dialog of the user function type “comprtor.s01” for source file and “compare” for function name. The source file must be in the same directory of your .pjn file.

```
; ~~~~~~  
; Module : comprtor.s01  
; Subject : Magnitude Comparator  
; Create : 09.02.99  
; Update : 09.02.99  
; Company : MicroSHADOW Research (uS)  
; Author : [GF-Jack]  
; ~~~~~~  
  
INCLUDE "sfr8051.inc"  
INCLUDE "kernel.inc"  
  
publiccompare  
  
extern __cmpw_eq  
  
; ~~~~~~  
; compare  
; ON ENTRY :  
; R6R7 = INPUT1  
; R5R4 = INPUT2  
;ON EXIT :  
; (WORD) BA = Compare Result  
; TRUE = EQUAL  
; FALSE = NOT_EQUAL  
;  
; ~~~~~~  
compare:  
;  
;  
; Uses kernel function __cmpw_eq (Compare Word EQUAL)  
; This function requires values in BA & R6R7 and returns TRUE if  
; the values are equals.  
;  
;  
    mov a,r5  
    mov B,r4  
; Call functions
```

```
    lcall __cmpw_eq
; Clear MSB
    mov B,#0
    ret
; -----
; End of comprtor.s01
; -----
```

## 6.2 - Generic embedded board adapting

If you intend to use LadderWORK on a custom embedded board we suggest to design the board using a flat memory mapping model. Normally the 8051 addressing space is divided in two areas RAM AREA and ROM AREA. RAM area is used to keep user data and nodes data, ROM AREA is used to keep LadderWORK's generated code.

### Hook and interface functions

After the board design is finished user must write own custom input / output routines. We organized the input / output routines in three main groups :

- Generic LADDER standard I/O hook functions
- Serial I/O hook functions
- Panel & Keyboard hook functions

Generic LADDER standard I/O functions allow to adapt boolean I/O ports of your hardware. With these function we handle typical ladder input and output blocks.

Serial I/O hook function handles the basic commands that are recognized by the kernel and allow you to perform probing and watching

Panel & keyboard hook functions interfaces user terminal ( Display & Keyboard )

.

### 6.2.1 Generic LADDER standard I/O functions

Library components like INPUT, OUTPUT, NCINPUT, EINPUT, ENCINPUT, EOUTPUT have a configurable parameter called CHANNEL. If you configure this parameter to zero the system will drive the pre-programmed hardware resource defined in the parameter REFERENCE.

When this parameter is in the range 128-255, the system generates code to call the functions named `__io_write` ( for writing ) and `__io_read` ( for reading ).

These functions should be attached to the project using the INCLUDE function block. The typed file name must contain user routines for custom hardware support.

*NOTE : The extension for an include assembler file must be compliant with the used processor type. For 8051 always use .s01 extension*

---

<b><code>__io_write</code></b>	This function is called during the execution of a output block. This function is called with the following convention :
--------------------------------	---

*Function is called with :*

**R7** = channel parameter ( 128..255 )

**A** = value of input pin

*Function must exits with :*

**A** = same value of input pin

---

<b><code>__io_read</code></b>	This function is called during the execution of an input standard block . The function should return the value passed as input, if the associated resource is ON ( logical 1 ) else the function must returns zero. The tipology of the input, normally open or normally closed, it's passed as parameter in a register.
-------------------------------	--

User may respect the following calling conventions

*Function is called with :*

**R7** = channel parameter ( 128..255 )

**R5** = Normally open ( 0 ) or normally closed ( 1 ) switch flag

**A** = value of input pin

*Function must exits with :*

**A** = result value

## 6.2.2 Custom I/O software example

```

;-----
; Module      : io.s01
; Subject     : Generic custom I/O hardware support
; Create      : 22.10.99
; Update      : 22.10.99
; Company     : MicroSHADOW Research (uS)
; Author      : [GF]
;
; This is an example of how o use the ladder standard output block -( )-
; and input block -||- to drive a custom hardware on your board.
;
; For the output block we assume :
; We assume to have a byte-port externally addressable at address E000H
; in a custom system. The output routine copy the input boolean value to
; the bit specified by the parameter channel using this table :
;
;           Channel      Bit
;           128          0 of port 0E000H
;           129          1 of port 0E000H
;           130          2 of port 0E000H
;
; We also assume that the port hasn't read-back feature so we keep
; a mirror of its state to correct update the entire port
;
; For the input block we assume :
; We assume to have a byte port mapped at address 0E800H in a custom
; system. The input routine will report the state of the bit#0 of this
; port. The pin will respond to channel 160
;
;-----

INCLUDE "sfr8051.inc"
INCLUDE "kernel.inc"

public __io_write
public __io_read

        izdata

MyMirror:    ds        1          ; Reserve space for port mirror ( zeroed at startup )

        cseg
;-----
; __io_write
; Hook function for custom output blocks -( )-
;
;           ON ENTRY :
;           R7 = Channel (128..255)
;           A      = input signal from left pin
;           ON EXIT
;           A      = returns the left-input signal
;-----

__io_write:    mov DPTR,#0E000H      ; DPTR point to the port
               mov B,a                ; Saves input value
               jnz WriteOne           ; Jump for ON state

;-----
; Place bit to OFF
;-----

               mov a,r7                ; Get channel value
               cjne a,#128,__test129_0
               anl MyMirror,#BINNOT(01H)
               sjmp WriteNow

__test129_0:   cjne a,#129,__test130_0
               anl MyMirror,#BINNOT(02H)
               sjmp WriteNow

```

```

__test130_0:
    cjne a,#130,__test255_0
    anl MyMirror,#BINNOT(04H)
    sjmp WriteNow

__test255_0:
    sjmp __exit

;-----
; Place bit to ON
;-----

WriteOne:
    mov a,r7                                ; Get channel value
    cjne a,#128,__test129_1
    orl MyMirror,#01H
    sjmp WriteNow

__test129_1:
    cjne a,#129,__test130_1
    orl MyMirror,#02H
    sjmp WriteNow

__test130_1:
    cjne a,#130,__test255_1
    orl MyMirror,#04H
    sjmp WriteNow

__test255_1:
    sjmp __exit

WriteNow:
    mov a,MyMirror    ; Get the port mirror
    movx @dptr,a      ; Update port

__exit:
    mov a,B            ; Pass value to output pin
    ret

;-----
; __io_read
; Hook function for custom input blocks -||-
;
;      ON ENTRY :
;      R7 = Channel (128..255)
;      R5      = 0 for normally open switches, 1 for normally closed switches
;      A       = input signal from left pin
;      ON EXIT
;      A       = returns true if the associated switch is ON and
;               the left-input signal is ON else returns zero.
;-----

__io_read:
    cjne r7,#160,__RetZero    ; Returns zero for undesired channels
    mov B,a                   ; Save left input signal
    mov dptr,#0E800H          ; Input port addressing
    movx a,@dptr              ; Get data
    cjne r5,#1,__NO_SWITCH    ; Checks for NO/NC switches
    cpl a                      ; Invert the switch control signal

__NO_SWITCH:
    anl a,B                   ; Logical AND with the left pin
    anl a,#1                  ; Force result to be boolean
    ret                       ; Returns

__RetZero:
    clr a
    ret

;-----
; End of io.s01
;-----

```



### 6.2.3 - Serial I/O hook functions

User can customize the serial I/O kernel functions. Serial kernel handles some basic functions like probing and watching. LadderWORK's run time libraries implement a sub-set of MODBUS® protocol allowing you to watch variables during running.

Function templates are present in the file **comm\_io.s01**, present under the PLB directory of the selected microcontroller. Normally this file drives the standard UART of the 8051 microprocessor but ser can customize this routines for his purposes.

<b>__comm_tx</b>	<p>This function send a single byte to the serial port.</p> <p><i>Function is called with :</i></p> <p><i>A = Character to send</i></p> <p><i>User function must preserve the value of all the used registers</i></p>
<b>__comm_check</b>	<p>This function checks if there is available characters in the receiving queue. The function must return the number of available characters. In detail, we suggest to create an interrupt-driven environment where incoming characters are queued.</p> <p><i>Function must exits with :</i></p> <p><b>A</b> = Number of available characters</p>
<b>__comm_get</b>	<p>This function retrieve the first available character from the receive queue. Before use this function, the system always checks for characters availability using the <b>__comm_check</b> function. However if the function is called without available characters the procedure must return zero.</p> <p><i>Function must exits with :</i></p> <p><b>A</b> = First available character or zero if there is not characters in the queue</p>
<b>__comm_init</b>	<p>This function is called when the serial port is initialized. This function has no entry/exit arguments.</p>
<b>__comm_timer_ISR</b>	<p>Normally this vector is called by the system timer interrupt service routine, attached to timer zero of 8051 microprocessor. If user need to customize this vector he have to call this label inside a hardware timer driven interrupt service routine. User have just to perform a long jump ( LJMP ) to this vector. When the system interrupt routine is performed, it jumps to the vector <b>__comm_timer_RETI</b> where user have to place the code to clear the pending interrupt and perform a RETI instruction. Note that the communication kernel ( MODBUS® ) requires an interrupt with an interval of about 1 ms or less.</p> <p><i>In case of register use, all registers must be preserved</i></p>
<b>__comm_timer_RETI</b>	<p>This vector is reached after that the system has branched to the vector <b>__comm_timer_ISR</b> . This vector is used as end-of-interrupt and user must place code to reset the timer's pending interrupt and perform a RETI instruction.</p> <p><i>In case of register use, all registers must be preserved</i></p>

### 6.2.4 Panel & Keyboard handling functions

There are a lot of particular components named DISPLAY, KEYBOARD and FIELD that allow user to handle user terminals. The DISPLAY component allow user to drive display panels, the KEYBOARD function allow user to handle a keyboard attached to the board while the FIELD component handles the data entry.

The system communicates with the terminal using few functions. User must write and supply this function in assembly language.

Since the system can handle more than a user terminal, a particular identification code is always passed to handling functions so if you have more than one display or keyboard you can manage the appropriate device inside your code.

**NOTE :**

■ **The Panel / Keyboard kernel requires 8051 external memory into your system.**

■ **All the panel function must preserve the value of all the used registers**

---

<b>__terminal_init</b>	<p>This function is called during startup. User must perform all the hardware initializations for the own terminal.</p> <p><i>Function is called with :</i></p> <p><b>R5</b> = Terminal ID</p>
<b>__locate</b>	<p>This function is called whenever the software requires to update a user panel locating a particular text on display. The function should respect the following conventions.</p> <p><i>Function is called with :</i></p> <p><b>R7</b> = Value of display X-Coordinate <b>R6</b> = Value of display Y-Coordinate <b>R5</b> = Terminal ID</p> <p>User function must preserve the value of all the used registers</p>
<b>__putchar</b>	<p>This is the standard character output function.</p> <p><i>Function is called with :</i></p> <p><b>A</b> = Character to print <b>R5</b> = Terminal ID</p> <p>User function must preserve the value of all the used registers</p>
<b>__put_asciiz_code</b>	<p>This function is the print string function.. The string is terminated by a zero ( C-like strings ). Since the 8051 microprocessor has distinct addressing modes for CODE and DATA there are two separated print functions</p> <p><i>Function is called with :</i></p> <p><b>DPTR</b> = Points to the ASCIIZ string. Since strings passed to this function are stored in code segment user must use the <b>movc a, @a+dptr</b> instruction to get a single character. <b>R5</b> = Terminal ID</p> <p>User function must preserve the value of all the used registers</p>
<b>__put_asciiz_data</b>	<p>This function is the print string function. The string is terminated by a zero ( C-like strings ). Since the 8051 microprocessor has distinct addressing modes for CODE and DATA there are two separated print functions</p>

---

*Function is called with :*

**DPTR** = Points to the ASCIIZ string. Since strings passed to this function are stored in ram segment user must use the **movx a, @dptr** instruction to get a single character.

**R5** = Terminal ID

User function must preserve the value of all the used registers

#### **\_\_getchar**

This function is called whenever the software must read a character from the user keyboard. Since the software can't wait for a user hit this function must return immediately returning zero if no key data is available. The kernel uses the function **\_\_charcheck** to test if there are character available from the panel and if this is true call the **\_\_getchar** function.

*Function must exits with :*

**A** = First available key or zero if no key are available

**R5** = Terminal ID

User function must preserve the value of all the used registers

#### **\_\_charcheck**

The keyboard kernel is event-driven. Every time a key is pressed or released, a particular event is generated and must be kepted in a circular queue. The software uses the function **\_\_charcheck** and the function **\_\_getchar** to extract events from the keyboard queue. In detail, this function is called to test if there are characters coming from the user keyboard. This function must return the number of available characters. If the system handles just a character at time the function simply returns 1, if there is a character, else returns zero.

*Function must exits with :*

**A** = number of available characters

**R5** = Terminal ID

User function must preserve the value of all the used registers

#### **\_\_keyboard\_scan**

This function is called every PLC scan cycle. In this function user must write own code to handle the keyboard. Normally the routine attached to this hook performs the matrix scan of the keyboard and updates the associated data.

*Function is called with :*

**R5** = Terminal ID

User function must preserve the value of all the used registers

#### **\_\_keyboard\_init**

This function is called once during software start-up. In this function user must provide code to initialize the keyboard.

*Function is called with :*

**R5** = Terminal ID

---

**\_\_update** Particular systems requires to update the display data using a software background technique. This function is called every scan cycle. User can use this function to update the state of the display without break the flow of the main scan cycle. This is true for slow display system and serial displays.

*Function is called with :*

**R5** = Terminal ID

User function must preserve the value of all the used registers

---

**\_\_keyb\_translate** Since the FIELD component requires standard ASCII codes for data entry, user must supply a translation routine to convert the keyboard scan-code in the respective ASCII code.

*Function is called with :*

**R5** = Terminal ID

**ACC** = Scan code

*Function must exits with :*

**ACC** = ASCII Code

User function must preserve the value of all the used registers

## 6-3 - USASM51 - Assembler language reference

### 6.3.1 - Assembler directives summary

ASCII	BIT	BSEG	CONST	CSEG	DB
DS	DSEG	DW	ELSE	ENDFUNCTI	ENDIF
				ON	
EQU	EXTERN	FUNCTION	INCLUDE	IF	IZDATA
LITERALLY	LOCALS	ORG	PAGE	PUBLIC	PURGE
RADIX	XSEG	XZDATA			

### 6.3.2 - Assembler operators summary

Operator	Priority & Association
()	→ 1
! ~ + - BINNOT HIGH LOW NOT	← 2
* / %	→ 3
+ -	→ 4
<< >> SHL SHR	→ 5
< <= > >= == != GT LT EQ NE	→ 6
GE LE BINAND BINOR BINXOR	
&	→ 7
^	→ 8
	→ 9
&&	→ 10
	→ 11

### 6.3.3 - Literals

Suffix	Radix
D d	10
O o Q q	8
B b	2
H X h x	16

#### For Example

10d or 10D	Decimal 10
0FH or 0Fh or 0FX or 0Fx	Decimal 15
10O or 10o or 10Q or 10q	Decimal 8
01100100B or 01100100B	Decimal 100

**6.3.4 - 8051 microprocessor instruction set**

ACALL	ADD	ADDC	AJMP	ANL	CJNE
CLR	CPL	DA A	DEC	DIV AB	DJNZ
INC	JB	JBC	JC	JNB	JNZ
JZ	LCALL	LJMP	MOV	MOVC	MOVX
MUL AB	NOP	ORL	POP	PUSH	RET
RETI	RLA	RLC A	RR A	RRC A	SETB
SJMP	SUBB	SWAP A	XCH	XCHD	XRL

## **SECTION 7 - Technical notes**

## 7.1 MODBUS® PROTOCOL

The MODBUS protocol specifies one master and up to 247 slaves on a common communication line, each slave is assigned a fixed unique device address in the range 1 to 247. The master always initiates a transaction. Transactions are either a query/response type ( only a slave is accessed at a time ) or a broadcast/no response type ( all slaves are accessed at the same time). A transaction comprises a single query an single response frame or a single broadcast frame.

### RTU MESSAGE FORMAT

ADDRESS	FUNCTION	DATA	CHECK
8-BITS	8-BITS	N x 8-BITS	16-BITS



### 7.1.1 - Read Boolean ( Function Code 01 )

Boolean points are numbered as from 1001 ( Boolean number 1 = 1001 ). The data is packed one bit for each Boolean flag variable. The response includes the slave address, function code, quantity of data characters, the data characters and error checking. Data will be packed with one bit for each boolean flag ( 1=ON, 0=OFF ). The low order bit of the first character contains the addressed flag, and the reminder follow. For Boolean quantities that are not even multiples of eight, the last characters will be filled in with zeroes at high order end.

#### Master to Slave

ADDRES S	FUNCTION	DATA START LOW	DATA START HIGH	NUMB ER OF POINTS LOW	NUMB ER OF POINTS HIGH	CRC CHECK 16 BIT
01	01	04	60	00	0C	XXXX

#### Slave to Master

ADDRESS	FUNCTION	BYTE COUNT	DATA BYTE # 1	DATA BYTE # 2	CRC CHECK 16 BIT
01	01	02	XX	XX	XXXX

### 7.1.2 - Read Numeric ( Function Code 03 )

Function code 03 allows the MASTER to obtain the binary contents of holding registers in the addressed slave. The protocol allows for a maximum of 125 16 bit registers to be obtained at each request. **Broadcast mode is not allowed for function 03.**

These 16 bit registers are also grouped in sets of registers and accessed as one variable. The numeric range of the point number defines the variable type and indicates how many 16 bit registers make up that variable.

POINT# RANGE	VARIABLE TYPE	16 BIT REGISTER/POINT	# OF BYTES/POINT	MAX POINTS
3XXX or 13XXX	SHORT INTEGER	1 REGISTER	2 BYTES	125
4XXX	8CH ASCII STRING	4 REGISTERS	8 BYTES	31
5XXX or 15XXX	LONG INTEGER	2 REGISTERS	4 BYTES	62
7XXX or 17XXX	IEEE FLOATING POINT	2 REGISTERS	4 BYTES	62
14XXX	16CH ASCII STRING	8 REGISTERS	16 BYTES	15

*Example : Read short integer 3012 through 3013 from slave # 2*

ADDRES S	FUNCTIO N	STARTIN G HIGH	STARTIN G LOW	POINT# LOW	POINT# HIGH	CRC CHECK 16 BIT
02	03	0B	C4	00	02	XXXX

### 7.1.3 - Set Single Boolean ( Function Code 05 )

This message forces a single boolean variable either ON or OFF. Boolean variables are points numbered 1XXX or 11XXX. Writing the 16 bit value 65,280 ( FF00 HEX ) will set the Boolean ON, writing the value zero will turn it OFF. All other values are illegal and will not affect the Boolean. Using a slave address 00 ( Broadcast Address Mode ) will force all slaves to modify the desired Boolean.

#### Example : Turn Single Boolean Point 1711 on Slave # 2

##### Master to Slave ( RTU MODE )

ADDRES S	FUNCTIO N	BOOLEA N POINT HIGH	BOOLEA N POINT LOW	DATA HIGH	DATA LOW	CRC CHECK 16 BIT
02	05	06	AF	FF	00	XXXX

##### Slave to Master ( RTU MODE )

ADDRES S	FUNCTIO N	BOOLEA N POINT HIGH	BOOLEA N POINT LOW	DATA HIGH	DATA LOW	CRC CHECK 16 BIT
02	05	06	AF	FF	00	XXXX

**7.1.4. - Set Single Numeric ( Function Code 06 )**

Any numeric variable that has been defined on the 16 bit integer index table can have its contents changed by this message. The 16 bit integer points are numbered from 3XXX or 13XXX . When used with slave address zero ( Broadcast Mode ) all slaves will load the specified points with the contents specified. The following example sets 1 16 bit integer at address 3106 of slave number # 2.

**Master to Slave ( RTU MODE )**

<b>ADDRES S</b>	<b>FUNCTIO N</b>	<b>POINT# LOW</b>	<b>POINT# HIGH</b>	<b>DATA HIGH</b>	<b>DATA LOW</b>	<b>CRC CHECK 16 BIT</b>
02	06	0C	22	00	03	XXXX

**Slave to Master (RTU MODE )**

<b>ADDRES S</b>	<b>FUNCTIO N</b>	<b>POINT# LOW</b>	<b>POINT# HIGH</b>	<b>DATA HIGH</b>	<b>DATA LOW</b>	<b>CRC CHECK 16 BIT</b>
02	06	0C	22	00	03	XXXX

### 7.1.5 - Remote terminal unit (RTU) framing

Frame synchronization can be maintained in RTU transmission mode only by simulating a synchronous message. The LadderWORK kernel monitors the elapsed time between receipt of characters. If 3.5 character times elapse without a new character or completion of the frame, then the frame is reset and the next bytes will be processed looking for a valid address.

For 8051 systems LadderWORK kernel user the timer 0 for the MODBUS® timing. MODBUS® define the maximum elapse time between two consecutive bytes of the same frame equal to 3.5 character time. This gives the following timing :

Baud Rate	Byte Time Considering 10 bits (Start+8+Stop)	Byte Time * 3.5
300	33ms	0.12s
600	17ms	58ms
1200	8ms	29ms
2400	4ms	15ms
4800	2ms	7ms
9600	1ms	4ms
19200	0.5ms	2ms

## **7.2 - Timing resolution**

LadderWORK's run-time kernel uses only a hardware timer ( called SYSTEM TIMER ) for generating timing. The SYSTEM TIMER is used by components like CLOCK, DELAY and DEBOUNCE . Normally the resolution of the SYSTEM TIMER is fixed to 20Hz which means a period of 50ms.

## **7.3 - Memory models**

Some processor families have different addressing modes. In this section you can find useful information about how LadderWORK uses these features to optimize the generated code. The following discussion will be divided by processor family.

### **8051 Family**

The 8051 microcontroller has an amount of RAM inside the chip. Normally this RAM array is 64 to 256 bytes. The 8051 can address an external RAM bank up to 64KBytes. LadderWORK can be configured to use the internal memory, the external memory or both. Using internal RAM will generate shorter and fastest code but you can easily break-through the internal data limit. External memory is useful for large schematics and big quantity of data. This is true above all when you use FIFO / LIFO queues with large queue depth. But use of external memory means greater code and lowest speed.

## 7.4 - Flow process

In this section we discuss in detail how LadderWORK process the schematic files to generate binary code files.

### File extensions

Extensions	Description
.PJN	Project file. This binary file must be used with the LadderWORK integrated enviroment.
.NET	Netlist file. This is an ASCII file that contains all the information about the connections between components and all the settings for every component.
.Snn / .ASM	Assembler source file. The solver processor gives as output an assembly file. The extension <Snn> could change according to the used processor.
.Unn	Binary object file. These files are generated as output by the assembler process. The <Unn> extension could change according to the used processor.

Microprocessor	Extension
8051 Family	U01
.INC	Assembler INCLUDE files. The INCLUDE files are ASCII archives that are used by the ASSEMBLER module.
.PLB	PACKED LIBRARY File. This binary file is used by the SOLVER module to transform the intermediate P-CODE in the processor assembly language.
.SLI	SYMBOL LIBRARY File. This binary file contains the information about the component graphical representation and its logical feature.
.LST	The ASSEMBLER module can generate a LISTING file as output of the assembly process.
.MAP	The LINKER module can generate a mapping file indicating the location of the modules loaded in the project.
.HEX	This file is generated as final output. Inside this file you have the processor code.

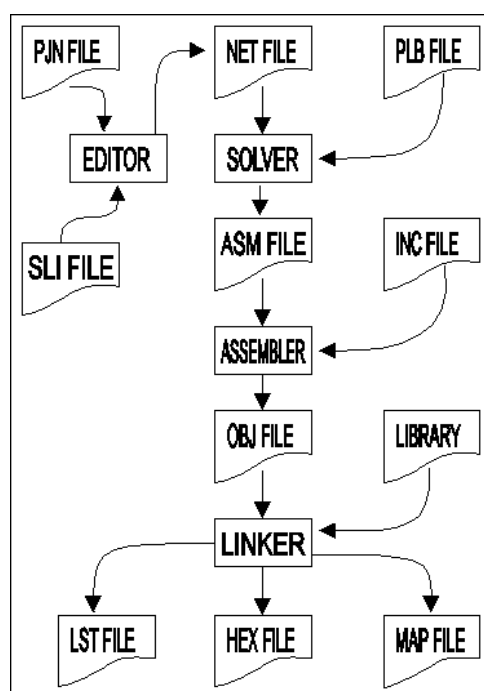
**FLOW PROCESS DIAGRAM**

Figure56 - Flow process diagram



## 7.5 - Logical links

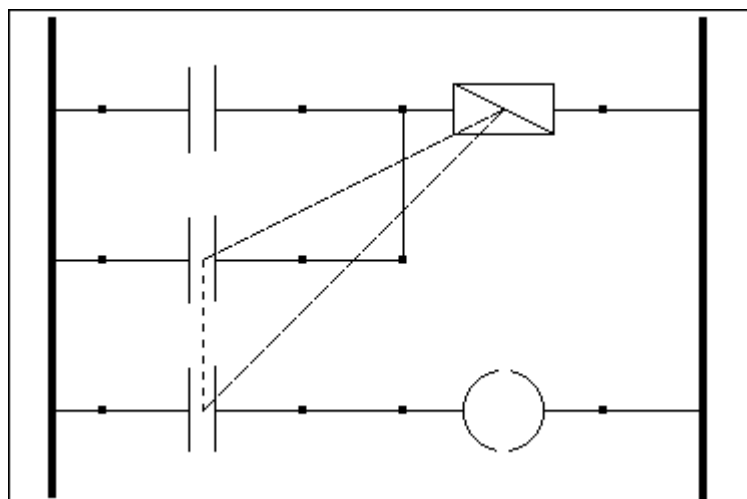


Figure57 - Logical links

The LOGICAL LINKS are particular connections which can be done between RELAY, OUTPUT, EOUTPUT components and INPUT, EINPUT, NCINPUT and ENCINPUT devices. For convention we will call the components RELAY, OUTPUT and EOUTPUT the output devices.

A LOGICAL LINKS is done assigning to an INPUT device the same REFERENCE code of an output device. In this way we tell the system to treat the output and the input as an unique object. The value assumed by the output device will be assumed also for all the input devices linked with. With the LOGICAL LINKS method it is possible to drive several sections in the net starting just from a master signal. In the example below we connected two inputs on a RELAY. The first input activates the RELAY and the second input, linked with the same RELAY, will sustain the RELAY activation. In the same way another input is logical linked to the RELAY so a generic output can be driven starting from the same signal. There's no limit on the number of the input that you can associate to an output device. In the LadderWORK software the LOGICAL LINKS are showed as a dashed line. This dashed line joint the components. The displaying of the LOGICAL LINKS is configurable by software.

## **SECTION 8 - ERROR MESSAGES**

---

**SFD0200**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_COUNTER_THRESHOLD_GT_BASE
<b>Category</b>	ERROR
<b>Description</b>	A COUNTER device was configured for down-counting with the THRESHOLD parameter greater than the BASE parameter.
<b>Possible cause</b>	Wrong configuration for COUNTER device.
<b>Possible solution</b>	When you operate win down-counters, the BASE value must be greater than the THRESHOLD value. For example, if i want a counting from 10 to 5 i have to configure BASE=10 and THRESHOLD=5 .

---

**SFD0201**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_COUNTER_BASE_GT_THRESHOLD
<b>Category</b>	ERROR
<b>Description</b>	A COUNTER device was configured for up-counting with the BASE parameter greater than the THRESHOLD rameter.
<b>Possible cause</b>	Wrong configuration for COUNTER device.
<b>Possible solution</b>	When you operate win up-counters, the THRESHOLD value must be greater than the BASE value. For example, if i want to perform a counting from 5 to 10 i have to configure BASE=5 and THRESHOLD=10.

---

**SFD0202**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_CLOCK_FREQ_ZERO
<b>Category</b>	ERROR
<b>Description</b>	A CLOCK device was configured with FREQUENCY equal to zero.
<b>Possible cause</b>	Wrong configuration for CLOCKdevice.
<b>Possible solution</b>	Frequency must be greater than zero.

---

**SFD0203**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_FIFO_SIZE_ZERO
<b>Category</b>	ERROR
<b>Description</b>	A FIFO device was configured with DEPTH equal to zero.
<b>Possible cause</b>	Wrong configuration for FIFO device.
<b>Possible solution</b>	The size of the queue must be greater than zero.

---

---

**SFD0204**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_FIFO_NEED_LARGE
<b>Category</b>	ERROR
<b>Description</b>	The actual memory model can't support the queue size you have programmed. In a typical 8051 system you have 64 - 256 bytes of RAM memory. If this amount isn't enough for your application you have to change memory model.
<b>Possible cause</b>	Memory model not suitable for your application.
<b>Possible solution</b>	Configure the compiler for LARGE memory model. The parameter DATA 8051 must be configured as EXTERNAL.

---

---

**SFD0205**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_LIFO_SIZE_ZERO
<b>Category</b>	ERROR
<b>Description</b>	A LIFO device was configured with DEPTH equal to zero.
<b>Possible cause</b>	Wrong configuration for LIFO device.
<b>Possible solution</b>	The size of the queue must be greater than zero.

---

---

**SFD0206**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SF_LIFO_SIZE_ZERO
<b>Category</b>	ERROR
<b>Description</b>	A LIFO device was configured with DEPTH equal to zero.
<b>Possible cause</b>	Wrong configuration for LIFO device.
<b>Possible solution</b>	The size of the queue must be greater than zero.

---

**SFD0207**

---

<b>Module</b>	SRCFEEDR.DLL
<b>Token</b>	SFSTATUS_SOURCELESS_INPUT
<b>Category</b>	ERROR
<b>Description</b>	A input device like INPUT, EINPUT, NCINPUT and ENCINPUT is not referred to any RELAY or (E)OUTPUT or PHYSICAL INPUT . This is a SOURCELESS condition. User must supply a source for that input.
<b>Possible cause</b>	See above.
<b>Possible solution</b>	Change the REFERENCE code, assigning a PHYSICAL resource or assign the same name of a OUTPUT, EOUTPUT or RELAY present in the schematic.

## **APPENDIX**



## Appendix A - Function block cross reference

**BAS** = Present in BASE version or higher class

**STD** = Present in STANDARD version or higher class

**ADV** = Present only in ADVANCED version

**UND** = Under development

Component	GRIFO GPC553	GRIFO GPC RT/94	PROCOE L ML46B	ELSIST PICOLOG	TECLAB PLC552	TECLA B TLMIC24
<b>AD_CON V</b>	ADV	NO	ADV	NO	ADV	NO
<b>CLOCK COUNT ER</b>	STD STD	STD STD	STD STD	STD STD	STD STD	STD STD
<b>DEBOUN CE</b>	BAS	BAS	BAS	BAS	BAS	BAS
<b>DELAY EINPUT ENCINPU T</b>	STD BAS BAS	STD BAS BAS	STD BAS BAS	STD BAS BAS	STD BAS BAS	STD BAS BAS
<b>EOUTPU T</b>	BAS	BAS	BAS	BAS	BAS	BAS
<b>FIFO INPUT LIFO</b>	ADV BAS ADV	ADV BAS ADV	ADV BAS ADV	ADV BAS ADV	ADV BAS ADV	ADV BAS ADV
<b>NCINPUT OUTPUT PWMOUT RELAY THRESH LD</b>	BAS BAS ADV BAS STD	BAS BAS NO BAS STD	BAS BAS NO BAS STD	BAS BAS NO BAS STD	BAS BAS ADV BAS STD	BAS BAS NO BAS STD
<b>AND FFD NOT OR</b>	STD STD STD STD	STD STD STD STD	STD STD STD STD	STD STD STD STD	STD STD STD STD	STD STD STD STD
<b>USER1 USER2 USER3</b>	ADV ADV ADV	ADV ADV ADV	ADV ADV ADV	ADV ADV ADV	ADV ADV ADV	ADV ADV ADV
<b>IPIN OPIN</b>	BAS BAS	NO NO	NO NO	NO NO	NO NO	NO NO
<b>ADD SUB MUL DIV MOD</b>	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV
<b>SHL SHR ROL ROR BIT</b>	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV	ADV ADV ADV ADV ADV
<b>DEC1-8</b>	ADV	ADV	ADV	ADV	ADV	ADV



<b>CTU</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>CTD</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>CTUD</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>TP</b>	STD	STD	STD	STD	STD	STD
<b>TON</b>	STD	STD	STD	STD	STD	STD
<b>TOF</b>	STD	STD	STD	STD	STD	STD
<b>TMI</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>TSQ</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>R_TRIG</b>	STD	STD	STD	STD	STD	STD
<b>F_TRIG</b>	STD	STD	STD	STD	STD	STD
<b>ASSIGN</b>	STD	STD	STD	STD	STD	STD
<b>READVA</b>	STD	STD	STD	STD	STD	STD
<b>R</b>						
<b>CONST</b>	STD	STD	STD	STD	STD	STD
<b>IDENT</b>	STD	STD	STD	STD	STD	STD
<b>SR</b>	STD	STD	STD	STD	STD	STD
<b>RS</b>	STD	STD	STD	STD	STD	STD
<b>SEMA</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>PFC_DSP</b>	NO	NO	NO	ADV	NO	NO
<b>Y</b>				Note 1		
<b>PFC_KEY</b>	NO	NO	NO	ADV	NO	NO
<b>B</b>				Note 1		
<b>QTP_DS</b>	ADV	NO	NO	NO	NO	NO
<b>PY</b>	Note 2					
<b>QTP_KE</b>	ADV	NO	NO	NO	NO	NO
<b>YB</b>	Note 2					
<b>SEVENS</b>	NO	NO	NO	NO	BAS	NO
<b>EG</b>						
<b>MBIN</b>	UND	NO	UND	UND	UND	UND
<b>MBOUT</b>	UND	NO	UND	UND	UND	UND
<b>MBCONF</b>	UND	NO	UND	UND	UND	UND
<b>MBSLAV</b>	UND	NO	UND	UND	UND	UND
<b>E</b>						
<b>DISPLAY</b>	ADV	NO	ADV	ADV	ADV	ADV
<b>KEYBOA</b>	ADV	NO	ADV	ADV	ADV	ADV
<b>RD</b>						
<b>FIELD</b>	ADV	NO	ADV	ADV	ADV	ADV
<b>PROBE</b>	BAS	NO	BAS	BAS	BAS	BAS
<b>SEL</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>MIN</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>MAX</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>LIMIT</b>	ADV	ADV	ADV	ADV	ADV	ADV
<b>MUX</b>	ADV	ADV	ADV	ADV	ADV	ADV

Note 1 : This feature requires PICOFACE user panel

Note 2 : This feature requires QTP16 or QTP22/24 panel



## Analytical Index

<i>8</i>		<i>L</i>	
8051 instruction set.....	166	LIFO .....	109
<i>A</i>		LIMIT .....	110
AD_CONV .....	76	Literals.....	165
ADD.....	75	Logical links.....	177
AND.....	77	<i>M</i>	
Assembler directives.....	165	Mathematical expressions.....	151
Assembler operators.....	165	MAX.....	111
ASSIGN.....	78	MBCONF.....	112
<i>B</i>		MBIN.....	113
BIT.....	79	MBOUT.....	114
<i>C</i>		MBSLAVE.....	115
CLOCK.....	80	Memory models.....	174
CONST.....	81	MIN.....	116
COUNTER.....	82	MOD.....	117
CTD.....	84	MUL.....	118
CTU.....	85	MUX.....	119
CTUD .....	86	<i>N</i>	
<i>D</i>		NCINPUT.....	120
DEBOUNCE.....	87	NOT.....	121
DEC1-8.....	88	<i>O</i>	
DELAY.....	89	OPIN.....	122
DISPLAY .....	91	OR.....	123
DIV .....	93	OUTPUT.....	124
<i>E</i>		<i>P</i>	
EINPUT .....	94	PWMOUT .....	125
ENCINPUT .....	95	<i>Q</i>	
EOUTPUT .....	96	QTP_DSPY.....	126
<i>F</i>		QTP_KEYB.....	127
F_TRIG.....	101	<i>R</i>	
FFD .....	97	R_TRIG .....	133
FIELD.....	98	READVAR .....	128
FIFO.....	99	RELAY.....	129
Flow process.....	175	ROL .....	130
<i>I</i>		ROR.....	131
IDENT.....	102	RS .....	132
INCLUDE.....	104	<i>S</i>	
INPUT.....	105	SEL.....	134
Interfacing with assembler.....	153	SEMA .....	135
IPIN .....	106	SEVENSEG.....	136
<i>K</i>		SHL.....	137
KEYBCTRL.....	107	SHR.....	138
KEYBOARD.....	108	SR .....	139
		SUB.....	140

*T*

Technical notes.....	167
THRESHLD.....	141
Timing resolution.....	174
TMI.....	142
TOF.....	143
TON.....	144

TP.....	145
TSQ.....	146

*U*

USER1.....	147
USER2.....	148
USER3.....	149

### References

- International Electrotechnical Commission (IEC), *Programmable Controllers Programming Languages*, IEC Standard IEC - 1131, Part 3, 1993. (Available in the U.S. from the American National Standards Institute, New York.)
- Allen Bradley, *PLC-5 Programming Software - Software Testing and Maintenance*, Publication 6200-6.4.10 November 1991a.
- Hughes, T.A., *Programmable Controllers*, Instrument Society of America, Research Triangle Park, NC, 1989.
- Intel Corporation, *PL/M-86 Programming Manual*, 9800466-02B, Chandler, Arizona, 1980.
- Intel Corporation, *8086 Software Tool Box, Volume II*, 122310-001, Chandler, Arizona, 1984.
- Intel Corporation, *PL/M-86 User's Guide*, 121636-004, Chandler, Arizona, 1985.
- Intel Corporation, *8086 Software Tool Box*, 122203-002, Chandler, Arizona, 1985.
- Intel Corporation, *PL/M-86 User's Guide for DOS Systems*, 481644-001, Chandler, Arizona, 1988.
- Intel Corporation, *PL/M-386 Programmer's Guide*, 611052-001, Chandler, Arizona, 1992.

