FUZZYSTUDIO™3.0

USER MANUAL
1st EDITION

MAY 1998

OWNERSHIP

STMicroelectronics is the sole owner of the Software contained in the package.

STMicroelectronics is the holder of the copyright to the Software, including without limitation such aspects of the Software as its code, sequence, organization, "look and feel", programming language and compilation names. Use of the Software unless pursuant to the terms of a licence granted by STMicroelectronics or as otherwise authorized by law is an infringment of the copyright.

PERMITTED USE

Provided that you fully accept the present conditions, STMicroelectronics grants you a non-exclusive non transferable licence to use the Product.

You are also authorized to:

a) install the Software on an on-line storage device (for example a hard disk drive);

b) maintain an archivial copy of the Software on off-line storage media (such as diskettes).

PROHIBITED USE

All uses of the Software and other elements of the Product not specifically allowed in the Permitted Uses section of this agreement are prohibited. The following is a partial list of prohibited uses of the Package. You are not allowed to:

a) decompile or reverse engineer the Software;

b) modify the Software in any manner;

c) sublicense, sell, lend, lease or rent the Software or any portion of the Software.

WARRANTIES

STMicroelectronics makes no warranties, express or implied, including without limitation any warranties of merchantability or fitness for a particular purpose, regarding the Software Development Tool and any related materials or their performance.

LIMITED DAMAGES

STMicroelectronics shall not be liable for any incidental, special consequential or exemplary damages including, but not limited to loss of anticipated profits or benefits.

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED

STMicroelectronics PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF STMicroelectronics. As used herein:

1. Life support devices or systems are those which (a) are intented for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonablyexpected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can reasonaly be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

Table of Contents

Before you Begin

Before you start using FUZZYSTUDIO™ 3.0, it is important to understand the terms and notational conventions used in this documentation.

General Conventions

The word "Choose" is used to carry out a menu command or a command button in a dialog box.

Bold type in text indicates words or characters you type.

Italic type indicates important terms introduced in the section.

UPPER CASE type in text indicates the names of commands and menus buttons of FUZZY-STUDIO™ 3.0.

A numbered list (1, 2, ...) indicates a procedure with two or more sequential steps.

A bullet symbol indicates a procedure with only one step.

Mouse Conventions

FUZZYSTUDIO™ 3.0 requires two mouse buttons. The default one is the left button but you can use the right button to perform some functions. For information on changing the mouse button, see your operating system documentation.

"POINT" means to position the mouse pointer until the tip of the pointer rests on what you want to point to on the screen.

"CLICK" means to press and immediately release the mouse button without moving the mouse.

"DOUBLE CLICK" means to press the mouse button twice in rapid succession.

Keyboard Conventions

A plus sign (+) used between two key names indicates that you must press both keys at the same time. For example ALT+F means that you press the ALT key and hold it down while you press the F key; this is the shortcut to choose the File menu.

A comma (,) between two keys' names indicates that you must press those keys sequentially. For example, "Press ALT, F, O" means that you press the ALT key and release it, press the F and release it, and then press the O and release it. This is the shortcut to choose the File OPEN command.

Installation and Configuration

The most-commonly-asked installation questions will be answered in the following pages, providing you with everything you ever wanted to know about the installation of FUZZY-STUDIO™ 3.0.

System Requirements

Before you install FUZZYSTUDIO™ 3.0, make sure you have all the hardware and software you need to run the program:

- Intel type 80386 processor or higher.
- 8 MBytes RAM memory.
- Hard Disk with at least 10 MBytes of free space.
- VGA or higher graphics card.
- Mouse.
- Windows 3.1 or Windows for Workgroups 3.11 or Windows 95.

Installing FUZZYSTUDIO™ 3.0

Your first step is to use the Setup program to install FUZZYSTUDIO™ 3.0 on your hard disk.

Be sure to start Microsoft Windows before to install FUZZYSTUDIO™ 3.0.

1 Insert the Installation Disk 1 into the floppy disk drive A or B.

2 Once Windows is running, select the Run option from the Program Manager File menu.

3 On the Run option's Command line, type:
A:\SETUP
then choose OK or press Enter

4 Window mask will appear, containing a window with the "*Initializing setup*" message. If you don't want to continue the installation choose the EXIT button, else choose CONTINUE.

5 A dialog box appears to set the directory where you want to install FUZZYSTUDIO™ 3.0. If a directory is not specified, the directory C:\FSTUDIO3.0 will be automatically generated along with the DEVICE sub-directory

6 After copying the necessary files, the setup program automatically generates the group and the icon to start the program.

When the setup procedure is completed, you will return back to the Program Manager.

Starting to use

After you have installed FUZZYSTUDIO™ 3.0, you can use the application.

To start FUZZYSTUDIO™ 3.0:

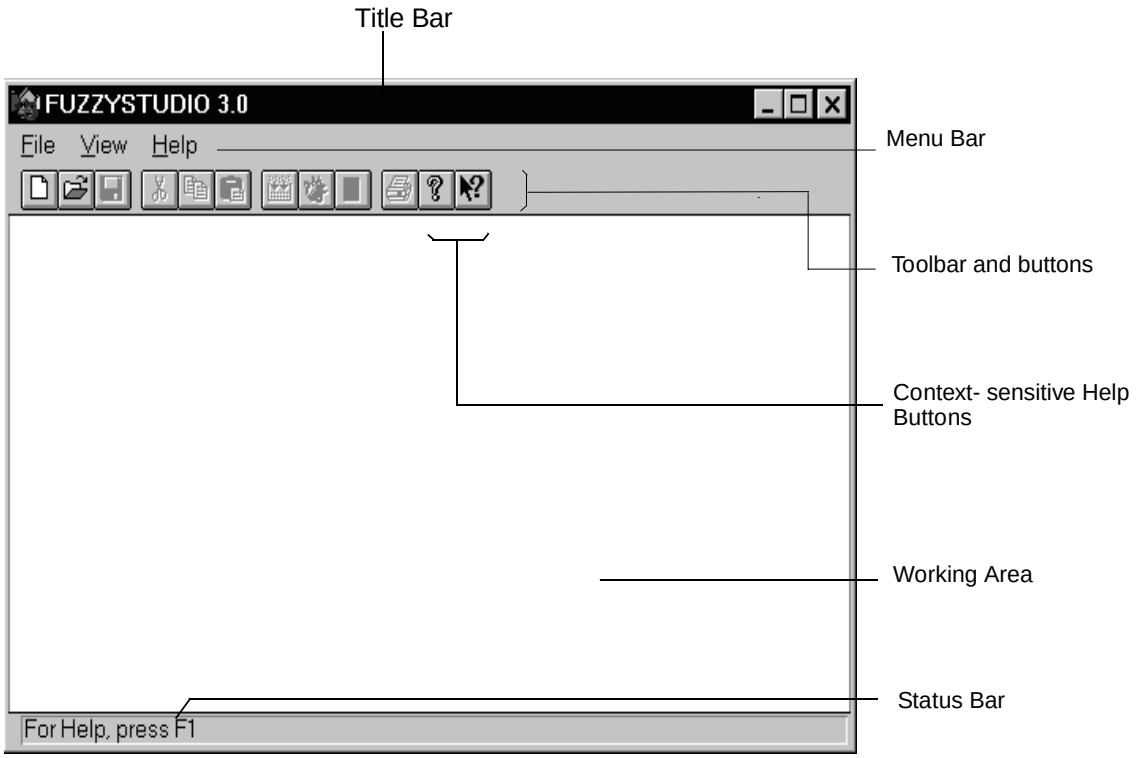
1 Double-click the FUZZYSTUDIO 3.0 group icon in the Program Manager window, or open the group that contains the FUZZYSTUDIO 3.0.

2 Double-click the FUZZYSTUDIO™ 3.0 icon to run the program.

User Interface

This chapter provides basic skills on the use of FUZZYSTUDIO™ 3.0 and explains the items you see on the screen. You'll learn how to choose commands and how to use FUZZYSTUDIO™ 3.0 online Help.

The first window that appears on the screen consists of the working area, the Menu Bar, the Toolbar and the Status Bar.



Choosing Commands

A command is an instruction that tells FUZZYSTUDIO™ 3.0 to do something. There are different ways to choose a command:

- Clicking a Toolbar button with the mouse.
- Choosing a command from a menu.
- Using shortcut keys.

Clicking a Toolbar button

The Toolbar consists of a number of icons, each representing a tool or option which you can use to create or modify your project.

The Toolbar is a user selectable item, it means that you can choose to hide or show it by using the related command of the VIEW menu.

Choosing commands from menus

FUZZYSTUDIO™ 3.0 commands are grouped in menus. Some commands carry out an action immediately; others display a dialog box so that you can select options.

To choose a command with either the mouse or the keyboard, you choose the menu and then the command name.

To select a menu or choose a command by using the keyboard, press the key for the underlined letter in a menu or number in the command name.

Using the mouse

If you use the mouse you have to follow these steps to choose a command:

1 To display a menu that contains the command you want, click the menu name in the menu bar.

2 Click the command name.
You can also point to the menu, drag to the command you want, and then release the mouse button.

3 If the command displays a dialog box, specify the information you need, (see "Working in a Dialog Box" section).

4 When you finish with the dialog box, click the appropriate button to carry out the command.

Using the keyboard

If you want to choose a command by using the keyboard:

1 To activate the menu bar, press ALT or F10. The FILE menu appears selected, indicating that the menu bar is active.

2 To open a menu press the key of the underlined letter of the command name you want to select or use the left or right arrows keys to select the menu name and the down arrow key to open it.

3 If the command displays a dialog box, specify the appropriate information that you need.

You can change field pressing the TAB key.

To close a menu, click anywhere outside the menu and the menu bar or press the ESC key.

To cancel a command, click the CANCEL button on the dialog box or press the ESC key.

A command name followed by the symbol > on a menu indicates that FUZZYSTUDIO™ 3.0 displays a sub-menu.

Using shortcut keys

You can choose some commands by pressing the shortcut keys listed on the menu to the right of the command. For example, to SAVE the current project, press CTRL+S.

Using Help

FUZZYSTUDIO™ 3.0 is provided with a complete online reference tool. Help is especially useful when you need information quickly or when your User Manual is not available. Help contains information about each command and each dialog box.

When you are working with FUZZYSTUDIO™ 3.0, you can select HELP using the menu name on the menu bar or choosing the help button that appears on the dialog box.

Once you are in Help, there are two ways you can move to other topics to find exactly the information you want, including:

Jump Terms. Jump terms are underlined with a solid line and are used to go to the topics in the help window.

Back button. The Back button is used to step back through all the topics you have viewed since opening the HELP window.

Index button. The Index button is used to display the list of Help Topics.

Context-Sensitive Help

To find out about an item on the screen, click the CONTEXT-SENSITIVE HELP button on the Toolbar.

When the pointer changes to a question mark, choose the command or click the window item on which you want help.

FUZZYSTUDIO™ 3.0 displays the Help topic for the selected command or window item in the HELP window.

Chapter 1

FUZZYSTUDIO™ 3.0

About ST52x301*

ST52x301 is an 8-bit fuzzy microcontroller with the following features:

- 8-bit Arithmetic Instruction Set.
- Fuzzy processing of systems with 4 Inputs and 2 Outputs.
- Programmable peripherals for the user application driving.
- Instruction Set to control the program.
- Programmable Interrupts.

For more information on ST52x301 refer to ST52x301 Data Sheet.

Introduction to FUZZYSTUDIO™ 3.0

FUZZYSTUDIO™ 3.0 is the development system that allows to program W.A.R.P. family of fuzzy processors. This *high level* tool helps you to: develop applications without Assembler programming, verify a developed project and program the microcontroller through the programming board supplied with FUZZYSTUDIO™ 3.0.

Thanks to the fuzzy logic approach, FUZZYSTUDIO™ 3.0 makes programming fast and easy even for those development phases which follow a traditional kind of approach. Since ST52x301 is able to manage fuzzy functions, arithmetic calculations and logic instructions, FUZZYSTUDIO™ 3.0 allows to develop a project and the management of all the various resources.

Moreover, FUZZYSTUDIO™ 3.0 provides a Visual programming approach to graphically define the program's logic flow by means of interconnected blocks. This can be achieved by designing the flow-chart which refers to your project by inserting the appropriate blocks. Each block you insert is already designed for a definite type of functionality which can be programmed either in a graphic way or with instructions at high level.

The links among the blocks determine the logic flow of the program. A double-click on the single block opens a programming environment specifically dedicated to the type of block.

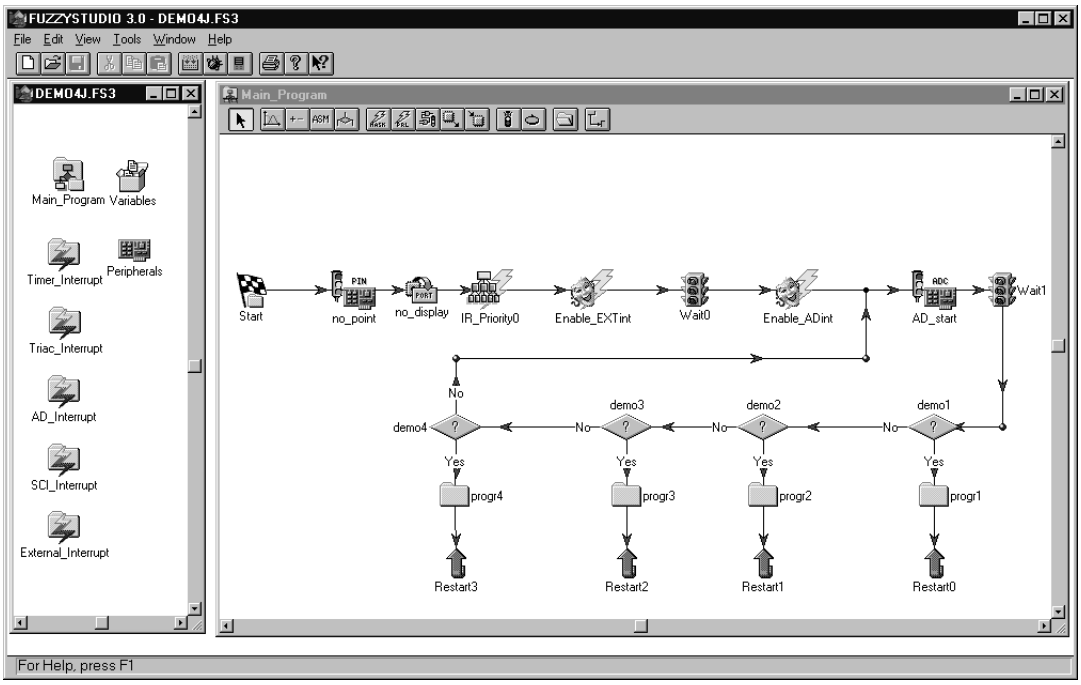
FUZZYSTUDIO™ 3.0 is equipped with tools that allow to generate the machine code and to perform the debugging of the program. It is characterized by the following main functionalities:

- Fuzzy programming.
- Arithmetic programming.
- Program's logic flow definition.
- Peripheral configuration and activation.
- Programming and activation of Interrupts and routines associated.
- Machine code generation.
- Full chip emulation in graphic environment.
- Chip EPROM memory programming.
- Possibility to program at low level using ST52x301 Assembler.

* Formerly W.A.R.P.3TC

FUZZYSTUDIO™ 3.0 main blocks are:

Start Block	Starting point of the program.
Fuzzy Block	Allows to performs the fuzzy functions.
Arithmetic Block	Carries out airthmetic and logic operations.
Assembler Block	Allows to insert arithmetic instructions in Assembler.
Conditional Block	It is a conditional block that modifies the program flow in relation to user-defined conditions.
Interrupt Mask Block	Enables interrupts.
Interrupt Priority Block	Manages interrupt priority.
Peripherals Block	Enables/disables and sets/resets the peripherals.
Send Block	Sends the value to a peripheral coming from a register.
Receive Block	Receives a value from a peripheral and stores it in a register.
Wait Block	Stops the program until the first interrupt signal.
Restart Block	Restarts the program.
Folder Block	Allows to group blocks.
Links	Connects two different parts of the program.



Thanks to these blocks it is possbile to realize the desired project by simply drawing the program flow-chart and fixing each block functionalities.

System Variables Overview

The systemvariables are managed through the chip's internal registers.

The fuzzy variables are those that are in input and output in the fuzzy computational unit.

The variables support the following types:

SIGNED BYTE	Integer values included in the range [-128, 127].
BYTE	Integer values included in the range [0, 255].

In addition, when you define a fuzzy variable and its associated Term Set with the *Fuzzy Variables Editor*, it is possible to define the Universe of Discourse respectively in the ranges [-128, 127], [0, 255], [x, y] where *x* and *y* are values defined by the user.

Note *There are 256 intervals available (8-bit).*

The arithmetic operations are performed on 8-bit variables with signed or unsigned values (signed byte [-128, 127] / byte [0, 255]) which refer to the internal registers of the machine.

The variables can be *global* or *local*. The global variables can be defined by using the "global variable window", while the local variables can be defined within an arithmetic block and can be used only inside that block. The operations between different types of variables are managed in a transparent way by the Compiler in order to see at an editor level the signed variable (signed byte type), although at a machine level only the type defined in the interval [0, 255] (byte type) exists.

FUZZYSTUDIO™ 3.0 also supports another type of variable: *predefined variables*. These are particular variables having reserved predefined names. They address particular registers of the chip (typically the peripheral's ones). The predefined variables must be considered as BYTE type and can be used to perform ordinary operations.

For More Information Refer to chapter 6 for a list of these variables.

Programming Approach

The FUZZYSTUDIO™ 3.0 Visual programming approach has been designed in order to help you in the development of a project. These are the main phases:

- **Peripherals' configuration.**
  The peripherals' configuration is obtained by means of dialog boxes recalled from the Project window. You can choose the peripherals' working mode by clicking on the relative check box.

  The configurations which are not allowed are automatically disabled and in case you do not carry out a choice, default selections are available. For example, the initial state of the peripheral is considered disabled by default and to enable it you have to open the Start/Stop block of the peripheral at the beginning of the program.

- **Global Variables definition.**
  You can define the Global variables' name, type and association to the register through an apposite dialog window recalled from the Project Window.

- **Flow-chart creation.**
  It is possible to design a program formed by a main program and subroutines directly defining the program's flow-chart. After opening the main Program window, or a subroutine's folder window, to draw the program's flow chart, you must choose the type of block you want to insert by clicking on the relative button of the Block Editor toolbar, and then click on the client-area to insert as many blocks as you desire.

  You can perform the same operations with the other blocks and connect them according to the logic flow of the program.

  The link between two blocks is performed by means of click & drag operations. The blocks designed and interconnected can be selected to be modified, moved, deleted and so on. The links can be extended, shortened, disconnected, or deleted.

- **Open each block to be able to establish the operations you want to perform in that block.**
  Each block you insert in the flow-chart, corresponds to a peculiar editing environemnt which can be opened by double-clicking on it (except for the *Wait* , *Start* and *Restart* blocks).

  For example:

  The editing tool of the *Arithmetic Block* is a text editor which allows to write the arithmetic operations that the processor has to perform. It is possible to insert the conditional statement IF among the arithmetic instructions inside an Arithmetic Block of this type. Refer to further paragraphs for the description of the possibile operations and for their grammar. All is written by the programmer can be checked by means of a check command which performs a syntactic and semantic check providing a list of the errors: with a double-click on this list you can visualise immediately the code line which has produced the error.

  A double-click on the Fuzzy Block allows to open the *Fuzzy System Editor*: the environment of the variables definition, of the Membership Functions and of the Fuzzy Rules. A project can have more than one Fuzzy Block defined in different points of the program and this allows to apply fuzzy adaptive control algorithms. The structure of the fuzzy controller and the Membership Functions are defined in a graphic way, while the rules are defined through the *Rule Editor*.

  The *Conditional Block* allows to modify the program flow according to defined conditions.

The *Assembler Block* allows to perform the programming of the device directly in the assembler of the chip.

The *Folder Block* allows to insert a group of blocks in a separate folder window, so as to simplify the flow-chart's routine.

Moreover, it is possible to manage the interrupts and peripherals Start/Stop by means of apposite blocks.

- **Define the interrupts service routines.**
  ST52x301 can manage the Interrupts which derive from the peripherals (Timer, A/D, SCI and Triac Driver) and from the External Interrupt pin and perform the relative service routines. To be able to define the service routines, in terms of instructions or commands, you work as in the main program, but inside the client area related to each interrupt, recalled from the Project window.

- **Compile the project to generate the debugging and the machine codes.**
  The project compilation is carried out choosing COMPILER from the Tools menu. Then, a report window will open displaying the program errors, if any, or a message of successful compilation. After the compilation, some files are generated: a file containing the program listing in a WCL (W.A.R.P. Control Language) script file; an Assembler code file; a file containing information used by the *Debugger*; and the code to be loaded in ST52x301 memories in the standard format or in the one specified in the compilation options.

  In case of errors in the compilation, you have to correct and compile your program again.

- **Use the Debugger to validate the program.**
  After the compilation has been succesfully completed, the program can be tested by using the Debugger. The Debugger is a tool that allows to emulate the processor and its on-chip peripherals according to the defined program. The simulation step considered by the Debugger is the single clock cycle, then the graphics that it is able to produce are in function of the time. You can choose the signals and/or the registers to visualize, as well as the time interval to simulate. Moreover, it is possible to establish the breakpoints on the program and go step by step to better examine the behaviour of the device according to the program. You can also supply the input set to the chip to simulate external connections to the input pins.

  The registers values and the status of the signals chosen can be visualised in a table through tracing in temporal function.

  If the results of the simulation are not satisfying, appropriate changes have to be performed to the current program stored in memory. Recalling the debugger it automatically starts the compilation.

- **Program ST52x301 with the board supplied with FUZZYSTUDIO™ 3.0.**
  The final step is to program ST52x301 memory by means of the programming functions and the electronic board connected to the PC through the parallel port.

- **Testing of the application.**
  Now you can insert the programmed chip in your own application and test it.

Chapter 2

Project Management

Project Files Management

FUZZYSTUDIO™ 3.0 stores all the project data into a single file with the extension .FS3. To select the commands for the project file management (for example, SAVE or OPEN), choose from the FILE menu . The content of the FILE menu is different according if there is an opened project or not.

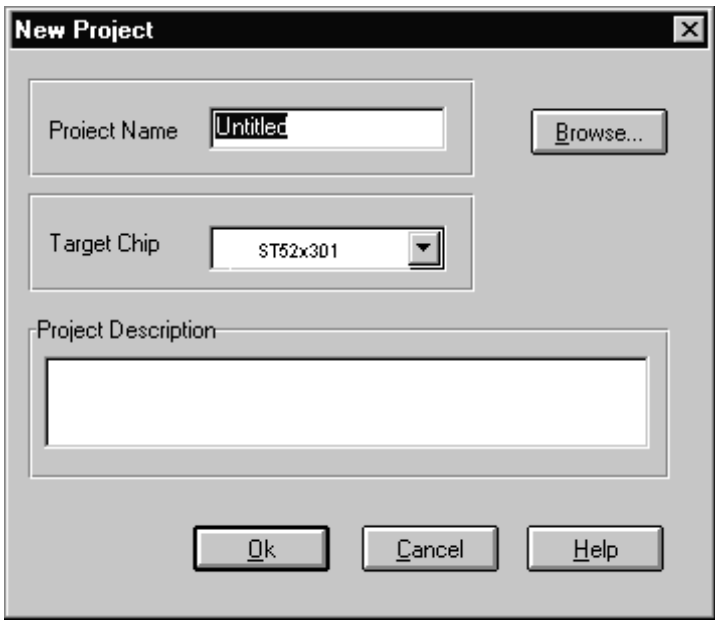
Working with a New Project

If the project has not been opened yet, the commands available for the project file management are the NEW and the OPEN commands:

New

The NEW command allows to start a new project. When you choose this command, FUZZYSTUDIO™ 3.0 displays the New Project dialog box to define the project name, specify the target chip and describe the project (these information can be read later by choosing PROJECT INFO). The file name has to be compatible with a standard file name because to this will be added the extension .FS3 when saving the file. It is also possible to specify a path to locate a project file in a directory different from the default one. The BROWSE button opens a dialog box that allows to quickly select the directory of destination of the project file.

Note *This file is not generated when performing the command NEW, but after a SAVE or SAVE AS command. In particular, the SAVE AS command allows to change the name of the file (and the name of the project) and its destination. It is easier to keep the default name i.e. UNTITLED and then decide later where to store it and its name. To carry out this command in a quicker way it is possible to use the toolbar button or press* **CTRL+N**.



Working with an Existing Project

Open

The OPEN command allows to open a project previously saved. On the OPEN command a dialog box will pop up, allowing you to select the project file to be loaded. Acting on the provided scroll down lists, you must choose drive, directory and file name, using the OK button to finally execute command. Into a networked environment, the NETWORK button can be used to connect to remote drives before selecting the data to be loaded. To carry out this command quickly it is possible to use the toolbar button or press CTRL+O.

If there is an active project, the FILE menu is modified and the following commands are available:

Close Project

The CLOSE PROJECT command allows to quit the current FUZZYSTUDIO 3.0 project. The same effect is obtained when Fuzzystudio 3 is closed from the main window or when closing the Project window. Then, you are asked to save the project if you have carried out some modifications not saved yet.

Saving and Printing Files

Save

The SAVE command will directly update the project file. A simple backup mechanism is implemented by copying the project file to a file with .BAK extension before updating it. In this way, you can open the current project data and the previous version. To quickly perform this command use the relative toolbar button or press CTRL+S. If no modification has been carried out after the last saving, the SAVE command is disabled.

Save AS

On the SAVE AS command a dialog box will pop up, allowing you to change current project name. The dialog box looks as the open project box, but the selected file will be used to store project data and not to retrieve from them. SAVE command will prompt for confirmation before overwriting an existing file.

Project Info

Allows to read the information relative to the project (Target Chip, name, working directory) and to read / write the project user description.

Print

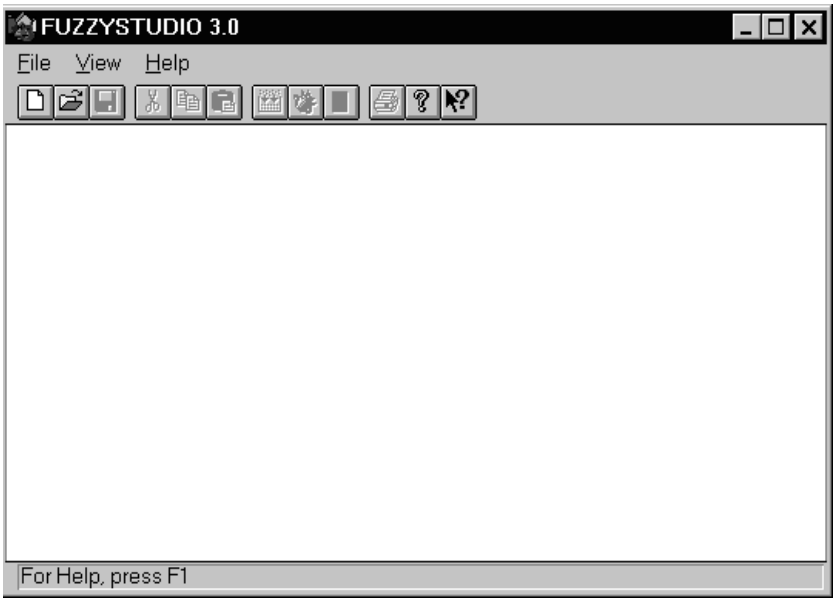
To print the project use the PRINT command in FILE menu. To quickly perform this command press CTRL+P. PRINTER SETUP and PRINT PREVIEW commands are also available in FILE menu.

The FUZZYSTUDIO™ 3.0 Main Window

This section provides an overview of the major elements of the FUZZYSTUDIO 3.0 Main Window, such as menus, toolbar and status bar. For additional information, see the index and online Help.

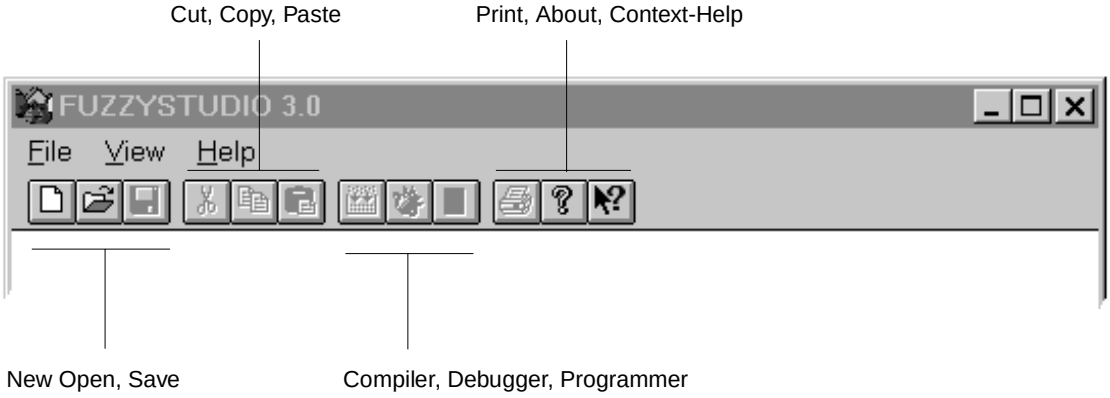
The FUZZYSTUDIO™ 3.0 Window Application menus

FILE	Contains commands to create a new project, to open an existing one or to change the printer and printing options, a list of recently used files.
VIEW	Contains commands to show or hide Main toolbar and Status bar.
HELP	Contains help commands.



The FUZZYSTUDIO™ 3.0 Main Windows Toolbar

For convenience, the most frequently used commands can be executed by choosing the corresponding buttons available on the toolbar.



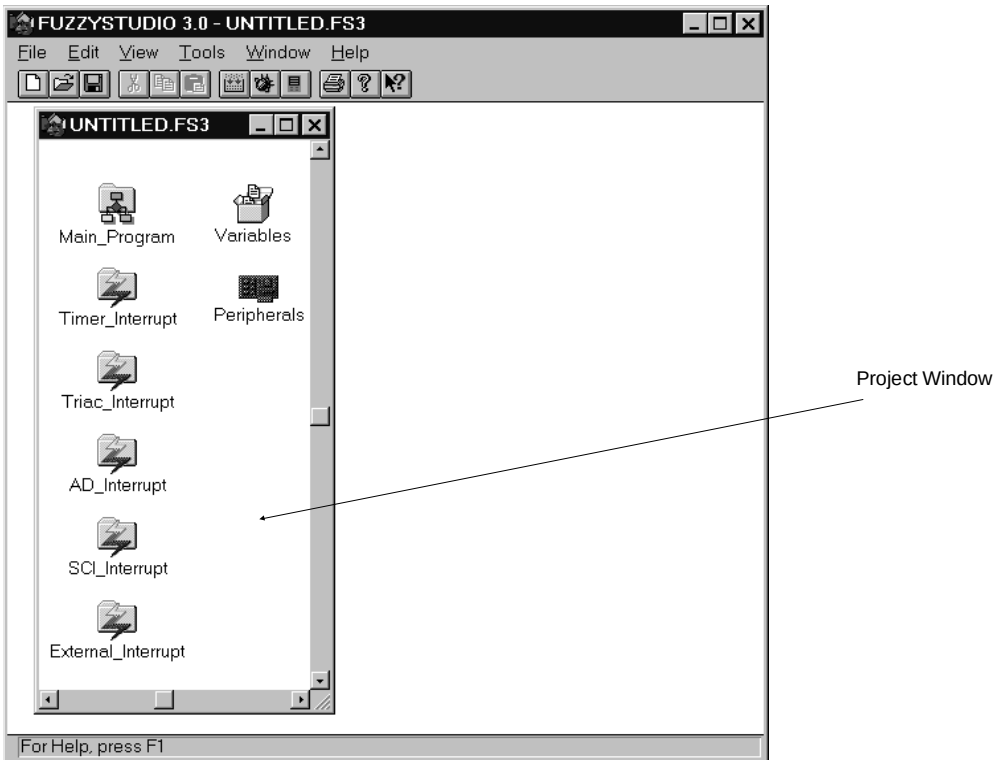
The FUZZYSTUDIO™ 3.0 Main Windows Status Bar

The Status Bar displayed at the bottom of the Main Window provides a brief description of the currently selected command.

The FUZZYSTUDIO™ 3.0 Project Window

A project definition environment can be accessed by means of the Project Window, that you see at a first glance when you start a new project or open an existing one. This window contains all the objects that make up the program: main program, sub-routines, interrupt routines, global variables and peripherals' settings.

In the Project Window you will find the icons that allow to access the various development and editing environments of the main parts of the project: a double-click on these icons allows to open the relative editing window.



Working Environments

The Project Window icons allow to easily access the FUZZYSTUDIO™ 3.0 working environments. The following list briefly describes the function of the each Project Window's icon:

Main Program	Opens the Block Editor's Main_Program window .
Variables	Opens the Global Variables definition and editing environment window.
Peripherals	Opens the peripherals programming environment.
Timer Interrupt	Opens the window for the programming of the Timer Interrupt's service routine.
Triac Interrupt	Opens the window for the programming of the Triac Interrupt's service routine.
A/D Interrupt	Opens the window for the programming of the A/D interrupt's service routine.
SCI Interrupt	Opens the window for the programming of the SCI interrupt's service routine.
External Interrupt	Opens the window for the programming of the External interrupt's service routine.

Note: Any window can be recalled by the WINDOW menu .

Chapter 3

Initial Settings

This chapter explains how to carry out the initial settings of your project. The first actions you have to perform to start a project will be the configuration of the device's peripherals, according to the selection of the target processor, and the definition of the program's Global Variables.

These operations are performed through the dialog boxes which can be invoked from the Peripherals and Variable's icons of the Project window.

Global Variables

Global Variables are used in the Arithmetic and Conditional blocks. At low level, they refer to the processor's registers and contain 8-bit values:

- Byte type (from 0 to 255)
- Signed Byte type (from -128 to 127)

How to insert a global variable

The procedure to insert the global variables is the following:

1 Click on the icon named VARIABLES in the Project Window to open the Global Variables dialog box.

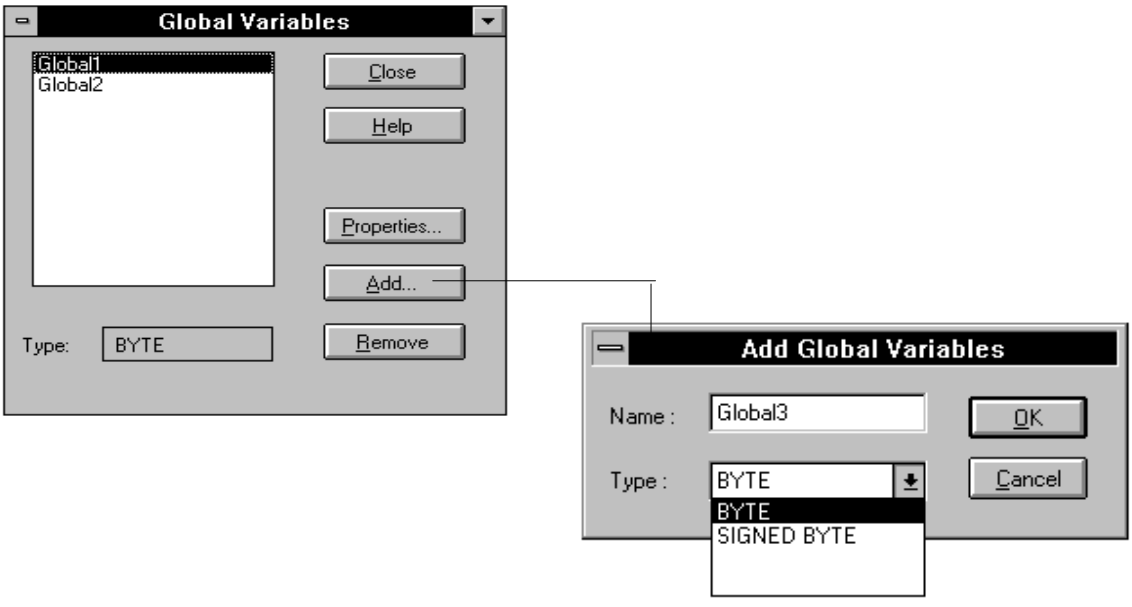
2 Click ADD button to open the dialog box ADD GLOBAL VARIABLES.

3 Insert the name of the variable.

4 Select the type (Byte or Signed Byte).

5 Click OK. The dialog box closes and the variable appears in the defined variables' list.

6 Repeat for the other variables and then click the button CLOSE.



Deleting a Global Variable

You can delete a global variable in the following way:

1 In the Global Variables dialog-box select the variable to delete from the apposite list.

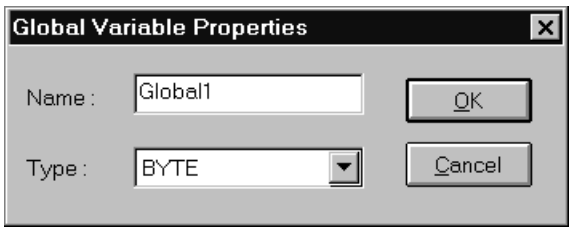
2 Click the REMOVE button. The variable is deleted from the list and will not be available anymore.

3 Click CLOSE to exit and confirm the deletion.

Note *In case you delete a global variable which is already being used in Arithmetic or Conditional blocks, the program remains unchanged and then in case of check or compilation, an error of the kind "not-defined variable" is generated.*

Global Variable Properties

Clicking on the button PROPERTIES, the GLOBAL VARIABLE PROPERTIES dialog box appears. This is similar to the dialog box for the insertion of a new global variable (ADD button), and allows to visualize and eventually modify the name and type of the global variable selected.



When you select a global variable from the list you are shown its type at the bottom of the dialog-box.

Note *The variables names have to be made up by alphanumeric characters, they cannot start with a number, they must not be equal to keywords (of the WCL description language and the assembler) and must have a maximum length of 32 characters. The names are case-sensitive. You can define 14 global variables at most, if you are using two fuzzy antecedent data memory banks. If you are using three or four fuzzy antecedent data memory banks, you can define respectively 13 or 12 global variables at most. A warning message informs you that you are defining too many variables without deleting them.*

Frequency Setting

From the Peripherals' icon in the Project window, it is possible to specify the microcontroller's clock frequency by selecting the value from the drop-down list available on the Peripherals window.

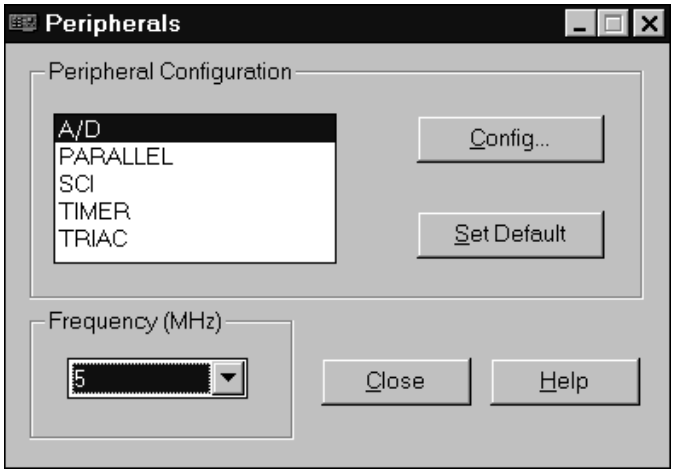
Note *For ST52x301 the available frequencies are 5 MHz, 10 MHz and 20 MHz.*

For More Information *Refer to data-sheets for further information on ST52x301 clock frequency.*

Peripherals Configuration

You can configure ST52x301 peripherals by double-clicking on the Peripherals' icon in the Project window and in the appearing dialog-box, selecting the peripheral among the ones available in the following list:

- **A/D CONVERTER**
- **PARALLEL PORT**
- **TIMER**
- **SCI** (Serial Communication Interface)
- **TRIAC DRIVER**



The configuration of the peripherals can be performed by selecting the check-boxes and the radio-buttons in apposite dialog-boxes, in order to set the functionality of the chosen peripheral.

To open the peripheral's setting dialog-box and to perform the configuration it is necessary to select the peripheral from the list and click on the button CONFIG..., or you can directly open the setting dialog box with a double click on the peripheral name.

For More Information Refer to the following paragrahs and ST52x301 Data Sheet for information on a specific peripheral's configuration.

After having completed the settings of the peripherals, click OK button to confirm the settings (if you press CANCEL you eliminate the changes performed). If no setting has been made or if the configuration of the peripheral selected has been partially performed, only the default values are activated. The button SET DEFAULT confirms the default configuration of the selected peripheral.

The chosen configuration corresponds to a series of Assembler instructions of the following type:
LDCF reg_conf, value

where: **reg_conf** is a value from 0 to 15 and refers to the address of the peripherals' configuration registers. **Value** is a value from 0 to 255 depending from the setting performed and from the configuration register. See Appendix C for further information on Assembler Instructions.

Timer Configuration

It is possible to configure the Timer through the dialog-box shown below: You can note different sections of configuration each having a different meaning:

Frequency

Allows to visualize the selected working frequency of the processor clock. The possible choices are 5 MHz,10 MHz or 20 MHz and can be selected from the Peripherals window.

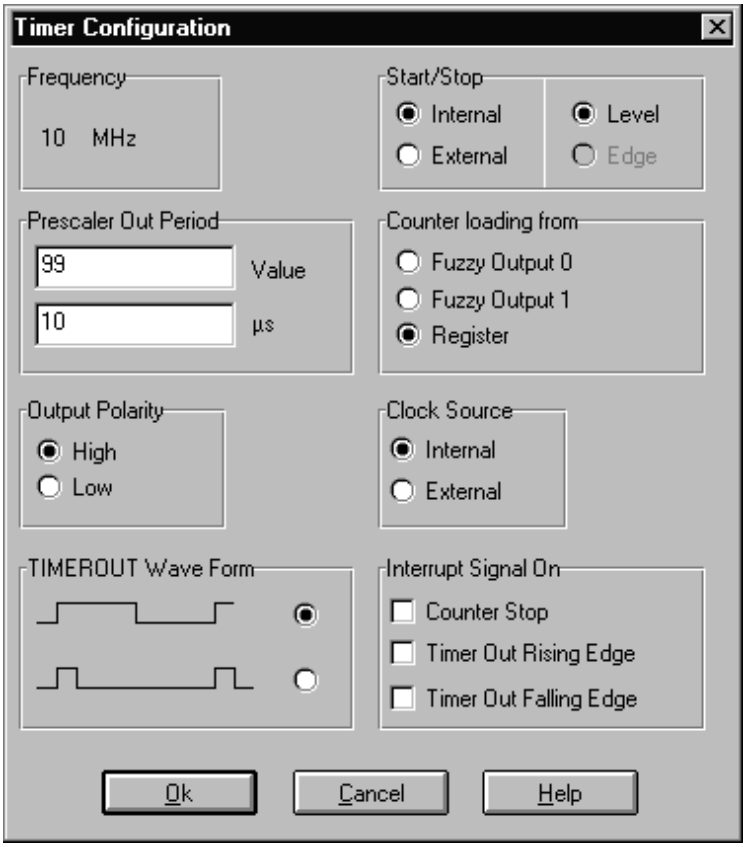
Prescaler Out Period

Allows the setting of the Prescaler Timer, i.e. the value for which the clock frequency is divided before being passed to the counter. It is possible to specify the value (included in the range 0 and 65535) that is written in the configuration registers of the Timer and visualize the corresponding value of the period in µs of the output signal of the Prescaler in case the source clock is internal.

Viceversa, it is possible to specify the time value thus automatically obtaining the value loaded into the register. Take care that if you enter a time value, it will be performed a conversion for approximation. Then this value can slightly differ during the implementation in the device.

Output Polarity

Allows to establish if the output signal logic level of the Timer in the pin TIMEROUT has to be high or low.



TIMEROUT Wave Form

Allows to choose the output wave form of the signal on the TIMEROUT pin. The possible selections are square or pulse wave form, as it is shown in the figures present on the dialog-box.

In the first case the Timer output is equal to a square wave with a duty-cycle equal to 50 % and period equal to the period of the Prescaler multiplied by the value of the count to perform.

In the second case, the Timer output is a pulse with length equal to a the output signal period of the Prescaler.

Start/Stop

Allows to choose if the Timer count can be started/stopped by an external signal or by a program and if this occurs on EDGE or LEVEL of the external signal.

Counter loading from

Allows to choose the data source to load on the Timer Counter. The source of this data can be one of the two fuzzy outputs units, or a variable specified from the program.

Clock Source

Allows to choose the origin of the input clock signal to the Prescaler. The possible choices are the internal clock or an external clock coming from the TIMER_CLOCK pin. In this last configuration the Timer can work as an event counter.

Interrupt Signal On

Allows to establish the event generating the Timer Interrupt signal.

The possible choices are:

- Interrupt on the Counter Stop
- Interrupt on the Rising Edge of the signal TIMEROUT
- Interrupt on the Falling Edge of the signal TIMEROUT
- Interrupt on both edges of the signal TIMEROUT

Selecting Counter Stop the other two possibilities are automatically excluded and viceversa. The counter stop interrupt signal can occur both with an internal stop from the program and external stop from the Timer Reset pin. In this case, it is possible to use Timer interrupt as External interrupt.

Triac Driver Configuration

The Triac Driver is a peripheral able to work in three different modes: Burst Mode,Phase Partialization , PWM Driver (refer to Data Sheet).

The dialog-box used to configure the Triac Driver changes according to the working modality chosen.

Sections common to the three modes

Frequency

Allows to visualize the processor's frequency: 5, 10 or 20 MHz.

Output Polarity

Allows to establish the polarity of the Triac output pulse in the TROUT pin . The possible choices are positive (logic level HIGH) or Negative (Logic level LOW)

Mode

Allows to establish the working modality of the Triac Driver. The possible configurations are PWM, Burst and Phase Partialization. Modifying the selection, the lower part of the dialog-box is modified too in order to supply the configurations sections.

Interrupt Source

Allows to establish the event generating the Triac Driver interrupt signal. You can choose among the following modes:

PWM: The Rising Edge and/or the Falling Edge of the counter signals (see data sheets).

BURST MODE:The Rising Edge and/or the Falling Edge of the counter signals (see data sheets).

PHASE PARTIALIZATION: The Rising Edge and/or the Falling Edge zero crossing signals of the power line positive and negative wave, coming from the MAIN1 and MAIN2 pins.

Counter Loading from

Allows to choose the data source to load on theTriac Driver counter. The data source can be one of the fuzzy unit outputs , or a Global Variable stored in a register, loaded from the program.

PWM mode

Prescaler Setting

Allows to set the Triac Driver Prescaler. It is possible to specify the value (0 - 65535) that is written in the Triac Driver configuration registers and visualize the corresponding value in µs of the whole control period, that means 256 *Tckp (where Tckp is the period of the Prescaler output). The value loaded in the counter is proportional to the TON period of the wave. See datasheet for further details.

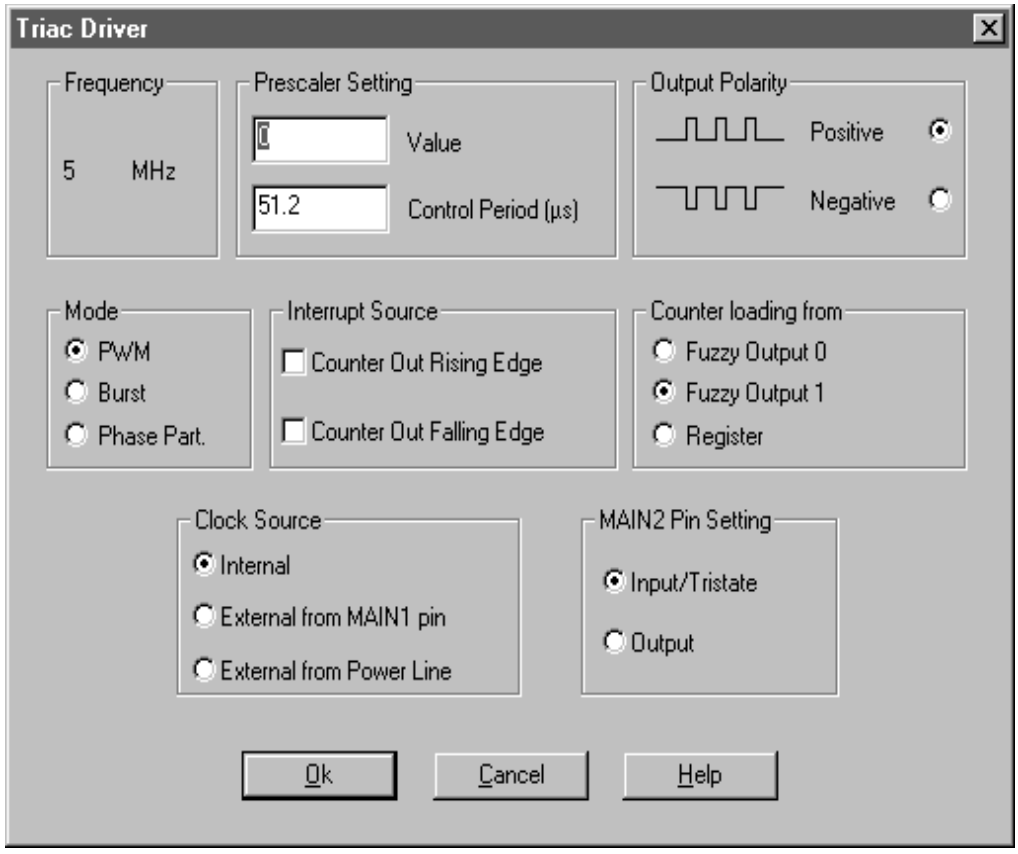
In the same way it is possible to directly set the value of the period in µs.

Clock Source

Allows to choose the origin of the clock signal in input to the Prescaler. The possible choices are: Internal clock, External from the MAIN 1 pin or the Power line frequency through the MAIN1 and MAIN2 pins.

MAIN2 Pin Setting

Allows to configure the MAIN2 pin as input/tristate to receive the frequency as external clock (see above) or in output supplying to the pin the prescaler output, that is the clock that drives the Triac Driver counter. Then, if you set a control period of 256 µs, the MAIN2 pin output will have a period of 1 µs.



Burst mode

Square Wave Period

Allows to set the Triac Driver Prescaler. In this case the the Prescaler is driven by the power line frequency through MAIN1 and MAIN2 pins, then it is possible to specify the value (included in the range between 0 and 65535) written in the Triac Driver configuration registers and visualize the corresponding period value in seconds of the whole control period, that means 256*Tckp where Tckp is the period of the prescaler output. The value loaded in the counter is proportional to the TON period of the wave. See datasheet for further details.

In the same way it is possible to directly set the value of the control period in seconds.

TROUT Impulse Width

Allows to establish the output pulse width to drive the Triac. It is possible to specify the value (included in the range between 0 and 16383) that is written in the configuration registers of the Triac Driver and visualise the corresponding value of the period in ms of the pulse. In the same way it is possibl eto directly set the value of pulse width in ms.

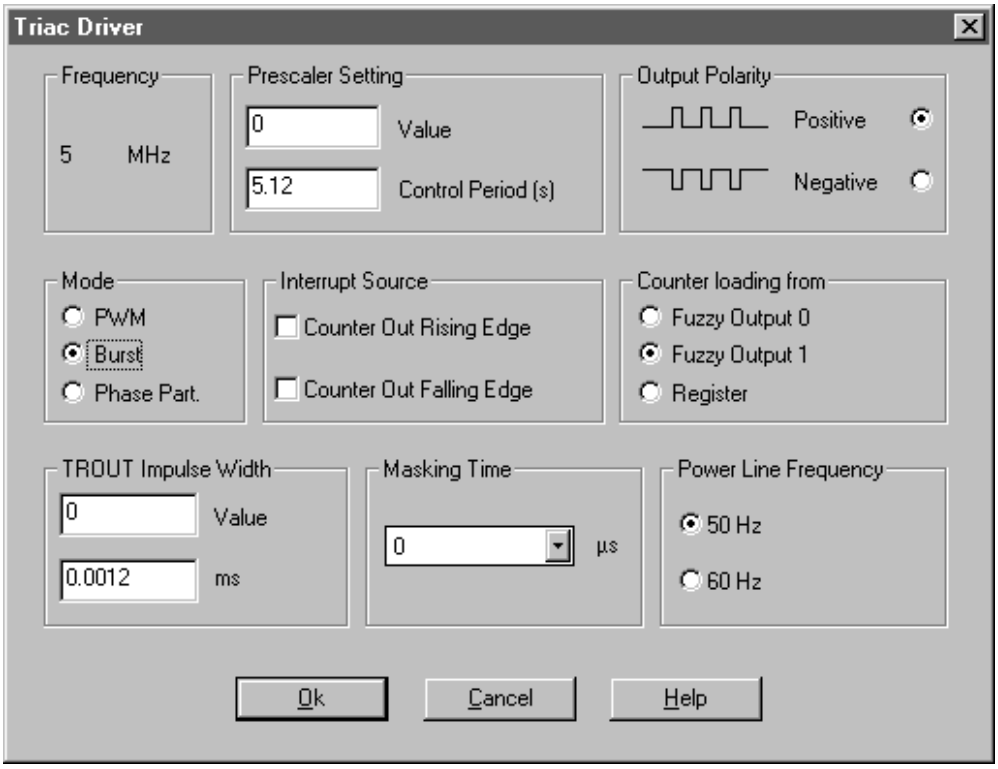
Power Line frequency

Allows to specify the frequency of the mains voltage of the system controlled. The possible choices are 50 Hz (european standards) or 60 Hz (american standards). Modifying such a data, the control period will change.

Masking Time

Allows to set a minimum delay time between two firing pulses, masking during this period zero crossing signals due to noise.

Refer to ST52x301 Data Sheet for further information.



Phase Partialization

Prescaler Setting

In Phase Partialization mode this parameter is not required then the check box is disabled.

TROUT Impulse Width

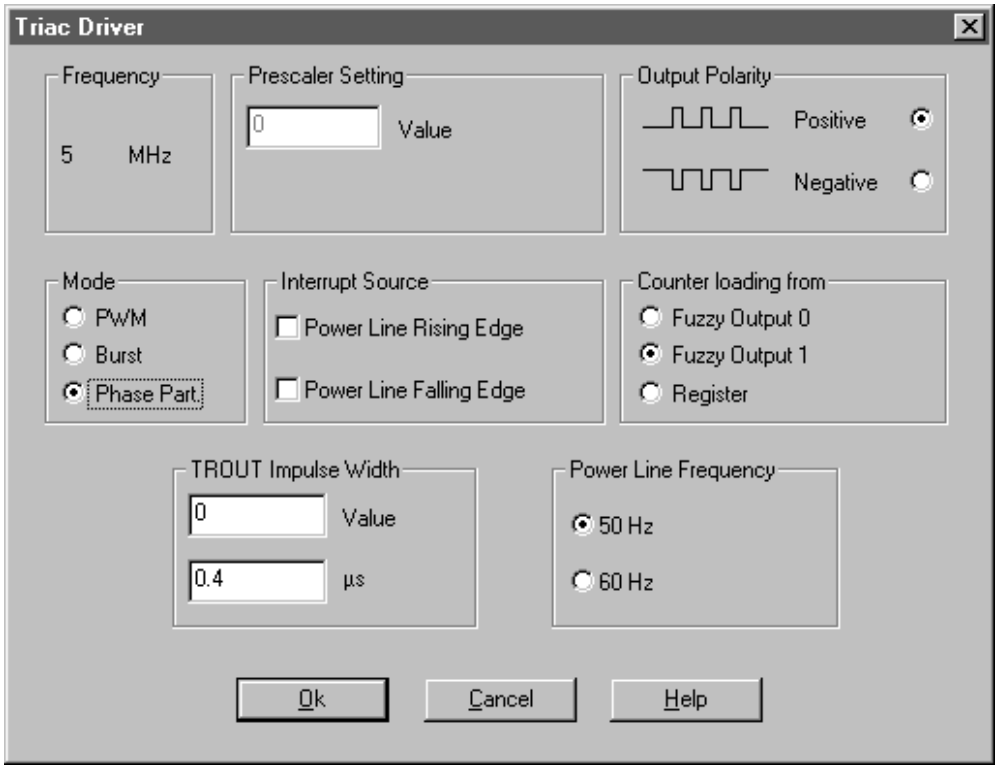
Allows to establish the length of the output pulse to drive the Triac. It is possible to specify the value that is written in the configuration registers of the Triac Driver and visualize the corresponding value of the period of the pulse in ms. The minumum value you can specify is 0, while the maximum value depends on the frequency of the clock master:

- 624 for 5 MHz
- 1249 for 10 MHz
- 2499 for 20 MHz (that corresponds to 250µs)

In the same way it is possible to directly set the value of a pulse width in µs.

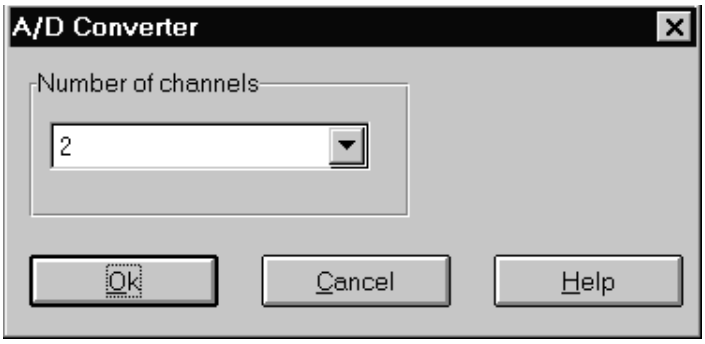
Power Line Frequency

Allows to specify the the power line frequency. The possible choices are 50 MHz (european standards) or 60 Hz (american standards).



A/D Configuration

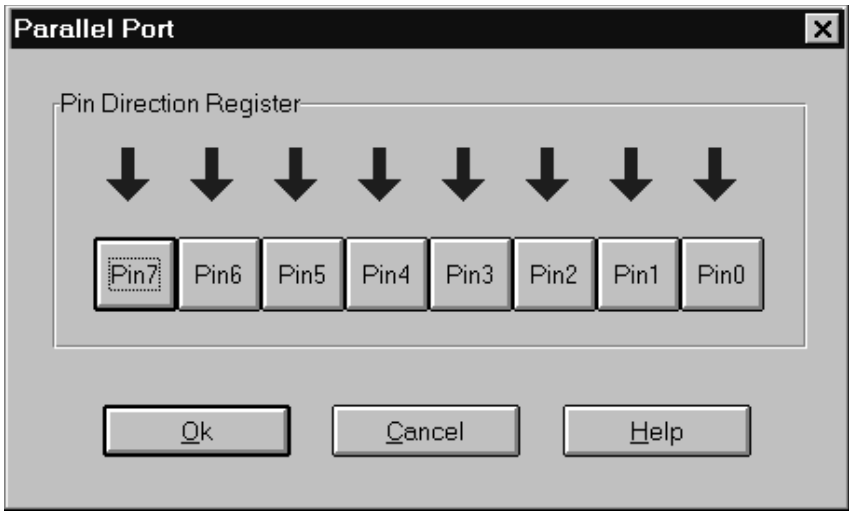
It is possible to configure the A/D Converter by means of the dialog-box shown below:



The only information to set is the number of the channels to convert.

Parallel Port Configuration

It is possible to configure the Parallel Port through the following dialog box:

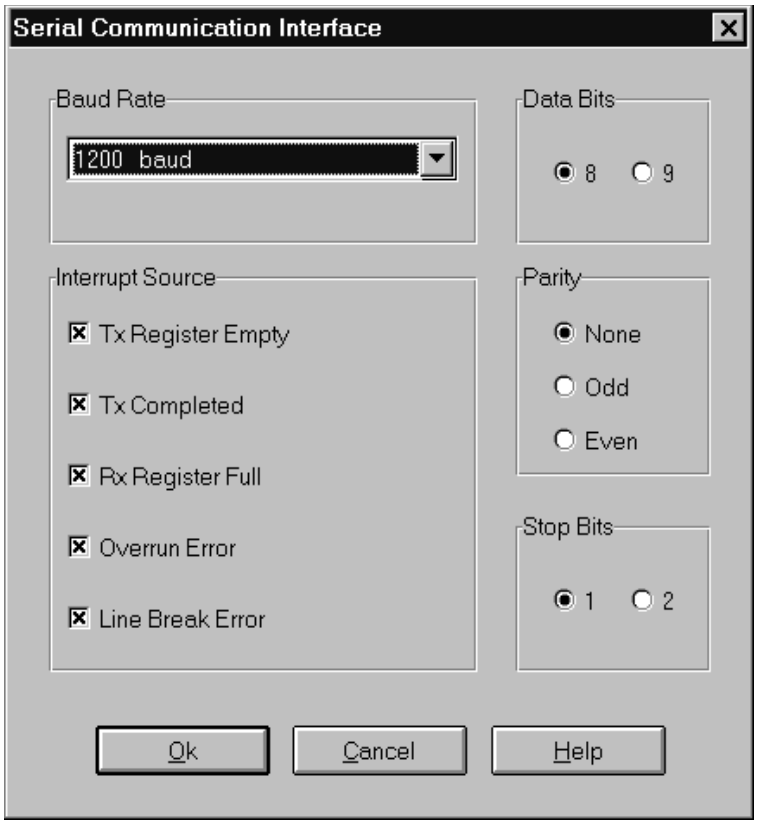


It is possible to utilize each pin of the Parallel Port as input or output. It is possible to specify the direction of a single parallel port pin by clicking on the pin button. Read arrows mean output pin, blue arrows indicate input pin.

Serial Communication Interface (SCI) Configuration.

The SCI is a hardware implementation of the standard RS-232 communication protocol. The programming of this peripheral allows to establish the characteristics of the data frame, transmission speed and the events that can generate interrupts.

It is possible to configure the SCI by means of the following dialog-box:



Baud Rate
Determines the communication protocol speed. The followings are the acceptable values: 600, 1200, 2400, 4800, 9600, 38400 baud. Moreover it is possible to specify that the speed is driven by an external clock.

Data Bits
Determines the data bit length. The possible choices are 8 and 9 bits. It is not possible to choose 9 bits when choosing to use the parity check or 2 stop bits.

Parity
Allows to configure the peripheral with or without the parity check. It is possible to choose also the parity type you want to obtain that is to say odd or even. It is not possible to use at the same time the parity check with a number of Data bits equal to 9 bits or 2 stop bits.

Stop Bits
Determines the number of stop bits of the frame: 1 or 2 bits. This last option can be selected only in case the parity check is not used or in case of 8 Data bits.

Interrupt Source

Determines the events generating an interrupt:

Tx register Empty:	The transmission buffer has been read by the shift register for the transmission.
Tx Completed:	Data transmission is complete.
Rx Register Full:	The data reception has been completed and the reception buffer is full.
Overrun Error:	An overrun error occurred. A new data item has been received before reading the one received previously.
Line Break Error:	An error due to the interruption of communication has occurred.

Note *More than an event can be chosen to generate an interrupt signal. However, this last is unique and to discriminate the event that has generated the interrupt it is possible to use the instruction SciStatus with an arithmetic block (see relative paragraph). Besides establishing if a specific event has generated the interrupt, allows to manage (in polling) other events that do not generate the interrupt: the "Frame Error", "Noise Error" and the 9th bit data contents if the peripheral has been set appropriately. For further details refer to related chapters.*

Chapter 4

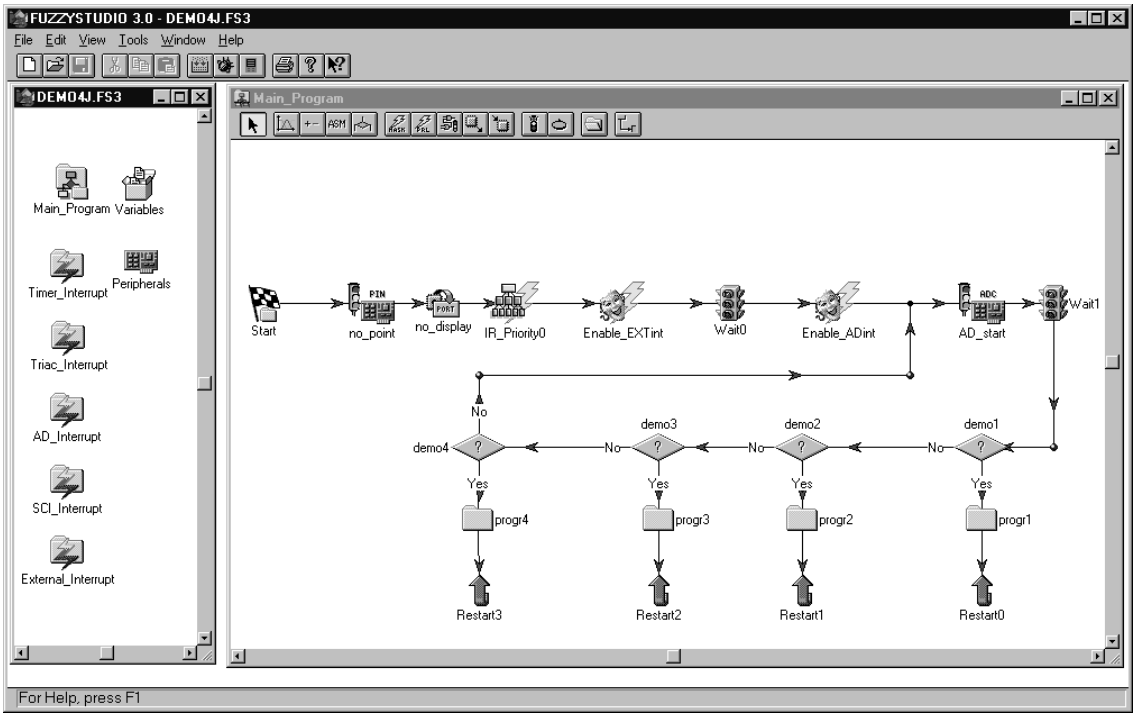
Block Editor Tool

The Block Editor is the tool for the "Visual" programming of the chip, that allows to specify the logic flow of the project and the access to the various definition environments of the program.

Each part of the program (main, interrupt routines and folders) is obtained by inserting logic blocks appositely interconnected that create the routine's flow-chart. This operation is performed by the Block Editor that is activated in the windows related to the main program, interrupt routines and subroutines.

The Block Editor Tool

This section provides an overview of the major elements of the Block Editor such as menus, toolbar and status bar. For additional information, refer to the index and on-line Help.



Block Editor menus

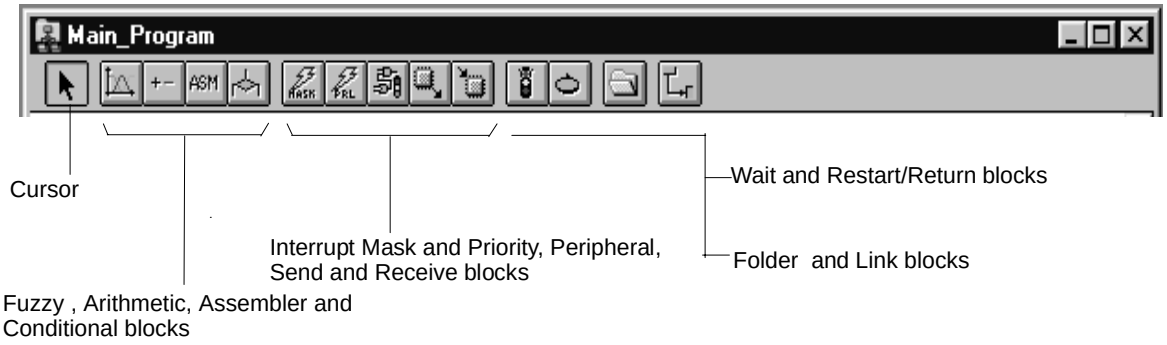
All Block Editor commands are grouped into a menu hierarchy accessible from the main window menu. Menus' items are context-sensitive: at each time only relevant commands for the selected objects or for the current project status will be enabled.

Available menus are:

File	Contains commands to create, open, close,save and print a project and to visualize the project's information.
Edit	Provides standard editing commands.
View	Contains commands for toolbar and status bar display or hide.
Tools	Contains commands to run Debugger, Compiler or Programmer tools.
Insert	Allows to insert the blocks for the creation of a flow-chart.
Window	Contains commands related to window management.
Help	Contains help commands.

Block Editor Toolbar

The Block Editor includes a toolbar to help you perform the most frequently used commands quickly. To execute a task by means of a button, just click the related button on Toolbar.



The cursor allows to select and move blocks and links within the client area. The blocks you can select from the toolbar are:

Cursor	Selects and moves blocks and links within the client area.
Fuzzy Block	Includes a Fuzzy System.
Arithmetic Block	Inserts arithmetic and logic instructions.
Assembler Block	Inserts assembler instructions.
Conditional Block	Determines program Jumps according to the specified conditions.
Interrupts Mask Block	Enables or disables the processor Interrupts.
Interrupts Priority Block	Specifies the Interrupt Priority level.
Peripheral Block	Enables or disables the peripherals.
Wait Block	Stops the processor waiting for an Interrupt.
Restart/Return Block	Restarts the main program immediately after the Start Block or, in a folder returns to the main program.
Folder Block	Includes a part of the logic flow.
Link Block	Links the blocks among each other.

Block Editor Status Bar

The Status Bar displayed at the bottom of the Block Editor window contains some information about the task you are working on.

Program Starting Point

When you open the Main Program, Interrupt or Folder's window the first block in your flow-chart will be the Start Block.

The Start Block is inserted automatically and is the beginning of the related program. At this point, it is possible to insert the logic blocks and to carry out the actions listed below:

How to insert a Block

1 From the Project window toolbar, you can select the block you want to insert in your project.

2 Position the mouse cursor anywhere on the project window.

3 Click the left mouse button. The new block is inserted in your project.

4 Drag the mouse to the desired position and then release it.

How to Link Two Blocks

A link is characterized by an arrow indicating the sequence of the program execution.

1 From the project window toolbar, select the LINK button. In this way you commute to the link insertion modality.

2 Position the mouse cursor on the first block to be linked.

3 Click the left mouse button

4 Drag the mouse on the second block and :

- if you still haven't released the mouse button you can release it now.
- or click again

You can add a link to any block, without necessarily linking the block to another one. To do this, click twice on the clinet area.

The links allow to carry out cyclical programs without using RESTART command simply connecting the last block of the program to the point from which you want to make it proceed.

Labels

A label is associated to each block, i.e a symbolic name. Such label is representative of the program address in which the instructions contained in that block start.

Default names are inserted automatically at the moment of the insertion of the block, but they can be changed anytime by selecting the item RENAME from the pop up menu that appears when you click on the block with the right mouse button.

The labels can be moved around the block by simply click & drag: they can be snapped on the cardinal points or they can be positioned anywhere around the block according to the activation of the item SNAP ON or SNAP OFF you can choose from the pop up menu which appears by clicking on the label with the right mouse button.

Note *Labels must contain no more than 32 characters.*

Single and Multiple Selection of a Block

To select a block, it is necessary to click on it. Take note that this operation cannot be performed in link insertion modality. In this case, you will have to make the cursor active before to select the block.

In cursor modality, it is possible to simultaneously select more than one block by a click & drag operation. This will open a frame in which the blocks are selected. It is also possible to select more blocks one by one by clicking the mouse on the block and holding down the CTRL button until the end of the selection.

The blocks and the links selected can be easily recognized because they change colour: the blocks become grey and the links become green.

Note *You can activate cursor modality by clicking the right mouse button on window client area. When already in cursor modality, ths action will open a pop-up menu.*

Basic Commands

A block, or a set of items selected at the same time, can be freely moved inside the window by using the mouse.

Moreover, any block can be:

Open	Opens the block's editing environment.
Close	Closes the editing environment if previously opened.
Rename	Allows to change the label associated to the block.
Delete	Deletes the block and its content (this command can also be perfomed by pressing the DELETE button on the keyboard). You are asked for confirmation before deleting the block.

This command list is a pop up menu obtained by clicking on the selected block with the right mouse button.

In case of multiple selection, clicking with the right mouse button on one of the selected blocks (with the exception of the Start block since it cannot be deleted), the pop up menu described above will appear. As you can notice, the item DELETE ALL SELECTED is added to this list. This command allows to delete all the selected blocks and links.

How to use Links

The link's interface is similar to the blocks' one. A link can be selected,deleted and moved (to move a link involves the moving of the linked blocks at the same time).

A link has a pop up menu that can be otained by clicking with the right mouse button over a link to:

Delete	Delete the links
Disconnect from start block	Disconnect the link from the start block
Disconnect from end block	Disconnect the link from the destination block
Disconnect from both	Disconnect the link from both blocks

Moreover, a link can be the target of another link.

Other commands

Clicking on the Block Editor client-area when in cursor modality with the right mouse button a pop up menu appears containing the following items:

Grid	Enables or disables the grid
Line up	Alignes the Block Editor objects on the grid

Chapter 5

Fuzzy Block

The Fuzzy Block allows to carry out the fuzzy functions to be performed by the program of control implemented in ST52x301.

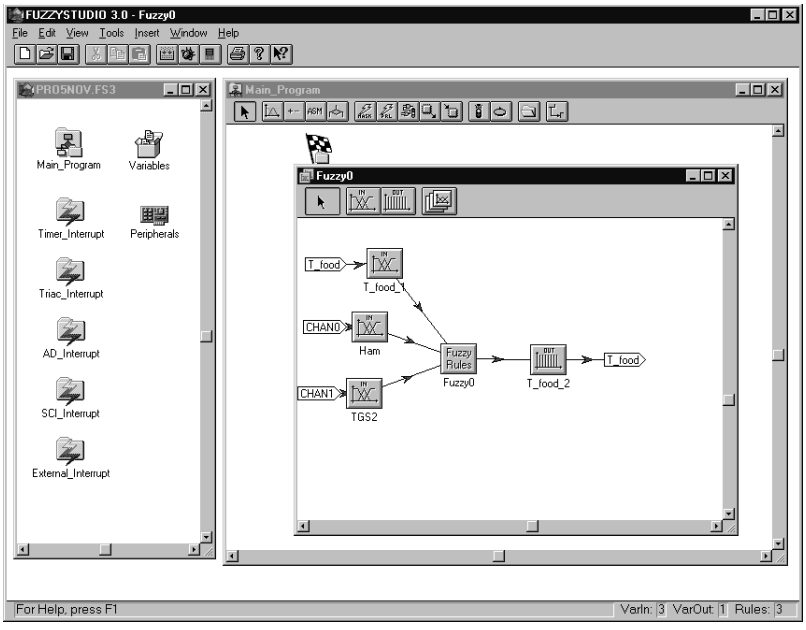
The possibility to define more than one Fuzzy Block allows to use, in the block diagram, the fuzzy system which is more appropriate to the general conditions of the system and then to realize an adaptive fuzzy control system.

Overview

You can insert a Fuzzy Block in your project by choosing FUZZY BLOCK from the INSERT menu. (Or by clicking on the Fuzzy Block Toolbar button from the Main Program Window). The mouse pointer changes its status and allows you to insert one Fuzzy Block anywhere on the Main Program Window by clicking on the desired position. In this way, you are able to run the Fuzzy System Editor: the development environment, dedicated to the fuzzy functions, in which it is possible to define the variables and the set of rules associated to the fuzzy function.

The Fuzzy System Editor interface employs a block structure with three different types of blocks: two for the definition of Input and Output variables and one for the editing of the rules.

Double-clicking on a Variable Block, a window pops up allowing you to define the Variable Properties: Name, Universe of Discourse boundaries and the related MBFs.



Running the RULE EDITOR, it is possible to edit the rules which define a fuzzy function. But before doing this, it is necessary to define at least one Input and one Output variable. You can run the RULE EDITOR by double-clicking on the Fuzzy Rules Block.

When you insert a rule containing a new variable, a link between this variable and the relative Block of Rules automatically appears.

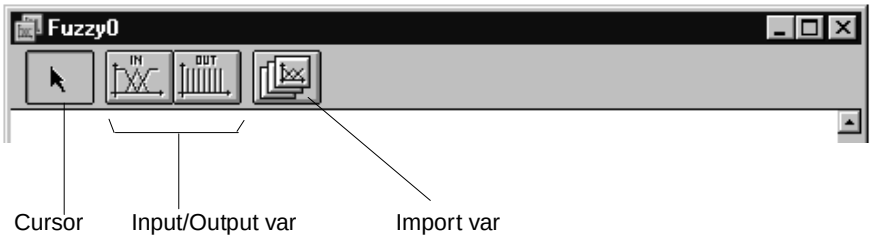
The Fuzzy System Editor Window

The first time you open a Fuzzy Block, the Fuzzy System Editor window shows an empty block named Rules Block. Then it is possible to insert, anywhere on the window, up to 4 Input variables and up to 2 Output variables.

This section provides an overview of the major elements of the Fuzzy System Editor window, such as menus, toolbar and status bar. For additional information, see the index and online Help.

Fuzzy System Editor Toolbar

The Fuzzy System Editor includes a Toolbar to help you to quickly perform the most frequently used commands. To execute a task by means of a button, just click the related button on the Toolbar.



Fuzzy System Editor Status Bar

The Status Bar displayed at the bottom of the Fuzzy System Editor window contains information on the Fuzzy Block you are currently working on and displays a brief description of the selected command.

The status bar provides the number of variables (input and output) and and rules in the current Fuzzy Block.

Fuzzy System Editor menus

Fuzzy System Editor menus are:

FILE	Provides new, open, save and print file utilities, a list of recently used files and the exit command.
EDIT	Provides standard editing commands.
VIEW	Contains commands to show or hide Block Chart Toolbar, Main Toolbar or Status bar.
TOOLS	Contains commands to switch to FUZZYSTUDIO™ 3.0 tools: Debugger, Compiler and Programmer.
INSERT	Allows to insert Input and Output variables, link variables from other fuzzy blocks and import fuzzy systems in F.U.L.L. language.
WINDOW	Contains commands related to window management allowing to arrange or activate windows.
HELP	Contains help commands.

About Fuzzy Variables

The possibility to define the number of input variables and MBFs, within each block, depends on what has been already defined in the other Fuzzy Blocks. This is due to the chip memory area used to store these information, which is limited and common to all the Fuzzy Blocks (refer to "Variable Constraints definition in a Fuzzy Block").

You can open a Variable Block by double-clicking on it or by activating the menu with the right mouse button. In this way the Fuzzy Variable Editor window will be open. This allows you to enter into the development environment dedicated to the fuzzy variables, in which it is possible to define the Universe of Discourse and the MBFs.

The first step to create a fuzzy project is to define its variables. For each variable you need to define:

- Name
- Universe of Discourse (UD)
- Membership Function (MBF)

Default settings are available.

During a session an existing variable window could be:

Open	The variable window is open on the FUZZYSTUDIO™ 3.0 window.
Active	The variable window is in foreground. All commands working on variable parameters modify it. An Open or Iconized variable becomes active by clicking on it.
Iconized	The variable window is iconized on the FUZZYSTUDIO™ 3.0 window.
Close	The variable window does not appear on the FUZZYSTUDIO™ 3.0 window.

Variable Constraints Definition in a Fuzzy Block

Generally, in a Fuzzy Block it is possible to define up to 4 input and 2 output variables. This possibility is sometimes limited by the presence of other variables defined in other Fuzzy blocks.

All the Input variables defined inside the Fuzzy Blocks share the same on-chip memory banks. This imposes some constraints in the number of the variables and MBFs to be added in each Fuzzy Block.

The Fuzzy System Editor automatically prevents to exceed the chip memory limits, according to what has been previously defined in each Fuzzy Block. In this case, the Fuzzy System Editor disables the commands which allow to create new variables and MBFs.

ST52x301 contains 4 memory banks dedicated to the storage of 4 * 16 MBFs each one associated to one input to the Fuzzy Core.

You can define:

- up to 4 variables, each one may contain up to 16 MBFs
- up to 3 variables, one of which may contain up to 32 MBFs
- up to 2 variables may contain up to 32 MBFs
- 1 variable containing up to 64 MBFs

All chip memory is available when you define the first Fuzzy Block of your project. Then, there will be some constraints for the definition of other Fuzzy Blocks since part of the memory is now busy. in this case, the maximum number of the variables which can be defined is:

$$n = \sum_{i=1}^{4} \min(1, 16 - m_i)$$

where m_i is the number of MBFs defined on each bank of memory from all the variables already present on the previous blocks.

If you define n variables, it is possible to associate a maximum number of MBFs $16 - m_i$ (i = 1, ..., n) where i is such that $16 - m_i$ different from 0

$$'n = \sum_{i=1}^{n} (16 - m_i)$$ where i is such that $16 - m_i$ different from 0

The available MBFs can be used in a different way by defining a number s < n of new variables; in this case it is possible to associate a number greater than 16 m_i MBFs.

In this case for a single variable it is possible to define a maximum number of MBFs equal to:

$$\sum_{i=1}^{n-s+1} (16 - m_i)$$ where i is such that $16 - m_i$ different from 0

When s=1 all remaining MBFs can be used for the variable in question.

How to Insert a Fuzzy Variable Block

Inside the Fuzzy Block it is necessary to define the variables you need to use in your project. The definition might occur in three different ways: creating a new fuzzy variable, sharing or copying an existing variable.

- **To create a new fuzzy variable :**

1 From the INSERT menu, choose the kind of variable by selecting INPUT VARor OUTPUT VAR. You can also click the Input or Output toolbar buttons available on the FUZZY SYSTEM EDITOR window.

After this operation the mouse pointer changes its status.

2 Click the mouse button anywhere on the window area in order to insert a new fuzzy variable block.

- **To share an existing fuzzy variable:**

A variable can be shared among two or more Fuzzy Blocks. For More Information refer to paragraph " Shared and Copied Variables" in this chapter.

- **To copy an existing fuzzy variable**

A variable can be copied from another Fuzzy Block using copy, cut and paste commands. For More Information refer to paragraph " Shared and Copied Variables" in this chapter.

How to Delete a Fuzzy Variable Block

You can delete unnecessary or unwanted variables in the following way:

1 From the FUZZY SYSTEM EDITOR window, select the variable block or the variable blocks you want to delete. In case of multiple selection, you can either use the mouse to create a frame or select each variable block by clicking without releasing the CTRL button.

2 Press the DELETE button on the keyword. You can either choose DELETE from the EDIT menu or click the right mouse button to select the item DELETE or DELETE ALL SELECTED from the pop up menu.

3 You are asked for confirmation before deleting. CLick YES or YES TO ALL to complete the operation; click NO to skip current variables deleting. Quit to end operations.

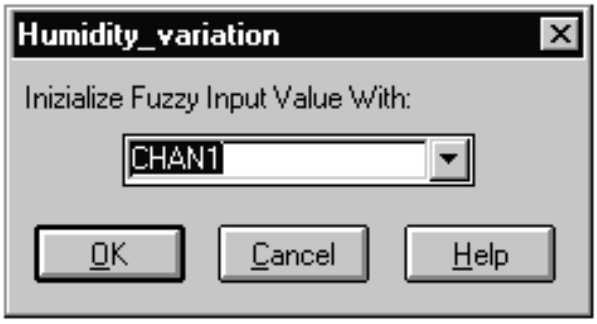
How to Close a Variable Window

An Open variable window can be closed via the CLOSE ACTIVE command of the WINDOWS menu. This command can only be performed in the current variable window.

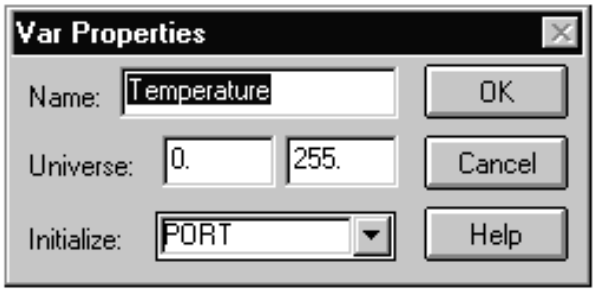
Fuzzy Variable Initialization and Storage

To assign a value, to be computed, to the fuzzy input variable by using FUZZYSTUDIO™ 3.0 it is necessary to associate it to a Global Variable, or an input register by using a predefined variable (refer to Arithmetic block chapter for more information on predefined variables), containing the crisp value .

- Click with the right mouse button on the Input Variable block you want to initialize.
- Choose Initialize from the appearing pop-up menu. A window containing a drop down list box will open.
- The list-box contains the available Global and predefined variables.



During the compilation, if the initialization of the variable is omitted, an error message will appear. The same operation can be carried out opening the Fuzzy Input Variable with a double-clicking on the input variable block and choosing VAR PROPERTIES from the VIEW menu or from the toolbar. Besides the editing fields of the Variable Name and Universe of Discourse, you will find a drop down list-box that works in the same way.



To store the fuzzy output in a Global variable or send it to a peripheral, it is possible to act as for the initialization.

- Click with the right mouse button on the output variable block where you want to store the value.
- Choose "Store in ..." from the appearing pop-up menu. A window containing a drop down list-box will be open.
- The list-box contains the available Global or predefined variables (see Arithmetic block chapter for explanation about predefined variables). Select the one you are interested in.

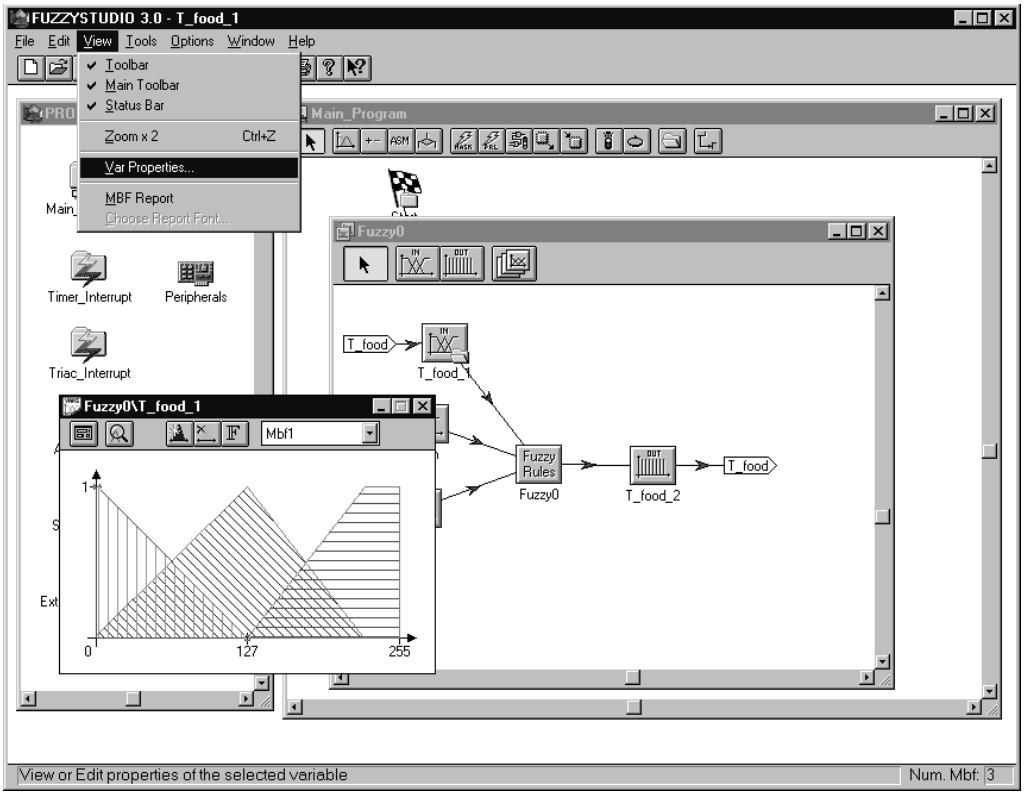
The same operations can be carried out opening the Fuzzy output variable double-clicking on the block and choosing VAR PROPERTIES from the VIEW menu. Besides the editing field of the Variable Name and Universe of Discourse, you will find the drop down list-box that works in the same way.

After initialization and storage phase, in the Fuzzy System Editor window, labels linked to the fuzzy variables will appear. Such labels will contain the Global or Predefined Variables, or Peripherals names, used to initialize and store the fuzzy variables.

Note *The fuzzy outputs can be sent directly to the Timer and Triac counter automatically by setting the peripherals in the apposite Peripherals Configuration dialog-boxes (see related chapter).*

The Fuzzy Variable Editor Window

Fuzzy Variable Editor is the tool devoted to manage with fuzzy variables and their MBFs. This section provides a description of the features of this tool.



Fuzzy Variable Editor menu

File Provides new, open, save, print file utilities, a list of recently used files, the project Info and the Exit command.

Mbf Allows to choose commands for the creation, modification and deletion of Mbfs and the copy on Clipboard as 'Bitamap' command.

View Allows to choose commands to show or hide toolbar and status bar; double the size of the window and to open the windows of the variable properties and a Mbf report.

Tools Contains commands to switch to FUZZYSTUDIO™ 3.0 tools: Compiler, Debugger and Programmer.

Options Allows to choose commands about the Mbfs colours, show/hide of coordinates and hide of printing names.

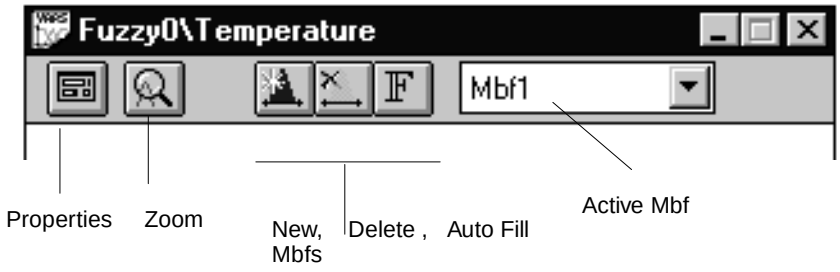
Window Contains commands related to window management.

Help Contains help commands.

Fuzzy Variable Editor Toolbar

The Fuzzy Variable Editor includes a Toolbar to help you to quickly perform the most frequently used commands . To execute a task by means of a button, just click the related button on the Toolbar.

Fuzzy Variable Editor Status Bar



The Status Bar displayed at the bottom of the Fuzzy Variable Editor window contains some information about the number of rules you are currently working on and displays a brief description of the selected command.

How to open a Fuzzy Variable Block

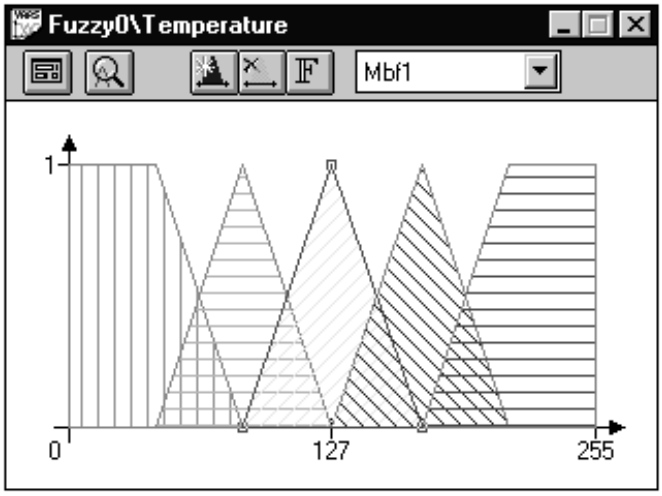
■ **To open a variable block:**

- Double-click on the Fuzzy Variable block.

or

- click on it with the right mouse button and choose OPEN from the pop up menu.

The variable window appears allowing you to edit the properties and the Mbfs associated.



Once opened, a variable window can be magnified (**CTRL+Z**) via the ZOOM X 2 command of the VIEW menu. This menu item and the related toolbar button acts as on/off switches, allowing users to magnify and de-magnify a window by repeatedly clicking on it.
This command acts only for the active window.

Note *The caption of the variable windos shows the name of the variable and the Fuzzy Block to which it belongs using the format FuzzyBlockname\VarName.*

How to Modify the Fuzzy Variable Properties

The variable properites are:

- Name
- Universe of Discourse Boundaries
- Intialize / Store in Variables

Once the variable has been created, you can modify its properties so as to associate a more explicative name and to set the lower and upper bound of its Universe of Discourse according to the range of the physical value.

■ **To modify the fuzzy variable properties:**

1 Open the Variable Window.

2 From the VIEW menu, choose VAR PROPERTIES (or simply click on the apposite FUZZY VARIABLE EDITOR toolbar button).

3 Modify the Name, the lower and upper bound of the Universe of Discourse by inserting the appropriate information on the dialog box.

4 Choose OK.

By default the variable name is set to "Var*x*", where *x* is a progressive integer value starting from 1, and the Universe of Discourse is set to [0, 255].

Variable names may contain only alphabetical characters, numerical digits and the underscore symbol ('_'). The first character must be an alphabetical one or the underscore. (Names must contain no more than 32 characters).

Due to its internal resolution, ST52x301 takes into account only 256 points in the Universe of Discourse. Since each point is represented with 9 digits and results from the division of the Universe width by 256, then the minimum Universe width must be $0\text{-}255*10^{-6}$.

Note *You must be aware that, in general, all the existing Mbfs, both for input and output variables, are automatically stretched or shrunk to the new Universe dimension. Only in case the output variable Mbfs have been defined directly in Fuzzy Rule Editor (see "Creating and managing membership functions"), the Universe resizing does not modify the Mbf crisp value and may cut off the Mbf and its related rules, if the Mbf value is outside the new Universe boundaries.*

Refer to "Fuzzy Variable Initialization and Storing" paragraph for the initialization and storing properties.

Creating and Managing Membership Functions

As stated earlier, a fuzzy variable is characterized by the name, the Universe of discourse and the Term Set, that is the set of all related Mbfs.

The Mbf related commands are grouped into the EDIT > MBF menu. These commands allow you to manage the Term Set of the active variable, so the EDIT > MBF menu is available only if the window is active.

To select one of the existing Mbfs you may click on it, in the drawing area, or use the toolbar drop-down list containing all the names of the Mbfs defined for the Active variable.

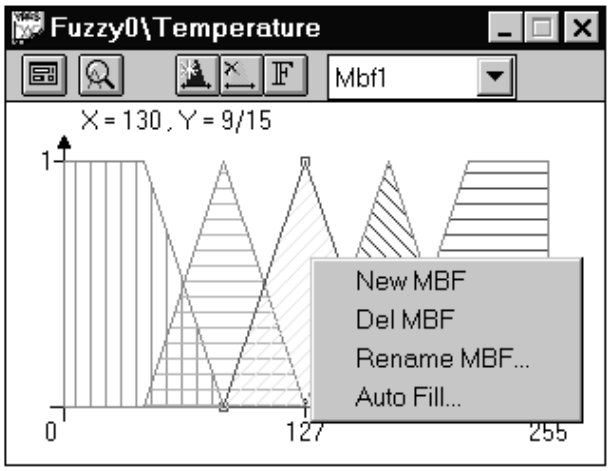
When a Mbfs is selected, its vertices are contained in a little square.

When the mouse pointer is into the drawing area, its position coordinates are shown both on top of the drawing area, as X and Y values and on the status bar, as Value and Belief.

Due to Mbf vertical resolution, the Y values are given as a x/15, showing mouse pointer position when into variable window drawing area.

You can enable/disable X and Y values display for the active variable by using the XY ON THE SCREEN command from the OPTIONS menu.

Clicking the right mouse button into the drawing area of a variable window, a pop up menu with the main Mbf commands appears.



By default, membership shapes are filled up with random coloured patterns. User can enable/disable this feature via the HIDE COLORS command from the OPTIONS menu. This menu item acts as an on/off global switch, whose state can be modified. The changes will apply to all the defined variables.

After opening a variable window, you can create its Mbfs by using:

- the command NEW MBF from EDIT menu (or pop up menu)
- the Autofill utility
- the Rule Editor (only in case of output variables)

Creating a New MBF

To create a new Mbf, the variable window to which it relates must be Open.

- Choose NEW MBF from EDIT > MBF menu (INS) or click the toolbar button or click the right mouse button.

The Mbf position into the variable Universe of Discourse is set as follows:

- In the centre of the Universe of Discourse, if Mbf is the first to be defined.
- Coinciding with the current cursor position in the drawing area, if using the right mouse button a menu pops up and if this position does not coincide with an existing Mbf.
- Shifted of the AutoShift value with respect to the previously defined Mbf.

For More Information *See paragraph "AutoShift and Semibase" chapter for further information.*

Creating a New MBF Using the Rule Editor (only in case of output variables)

The definition of Mbfs for output variables is also possible by using the Rule Editor. In this way you can set the output crisp value of the Mbf, by writing directly in a rule in editing.

However, this Mbf does not appear either in the drawing area of the relative output variable nor in the Mbf Report and cannot be managed through Variable Editor related commands. The Universe of Discourse resizing does not modify this Mbf crisp value but may cut off the Mbf itself and its related rules, if the Mbf value is outside the new Universe of Discourse boundaries.

For further information about the definition of a Mbf by using the Rule Editor, see the "Rule Editor" paragraph.

How to customize a MBF

To customize a Mbf you can change its name, position and shape. Membership name is set by default to 'Mbf*x*', were *x* is an integer value starting from 1 and it is incremented each time a new membership is created.

- **To change name:**

1 Select the Mbf clicking on it on the drawing area or selecting its name on the drop down list.

2 Choose RENAME MBF from EDIT > MBF menu or from right mouse button pop up menu.

3 Put the new name in the dialog box .

4 Choose OK.

- **To change position:**

1 Select the Mbf clicking on it on the drawing area or selecting its name on the drop down list.

2 Click into the membership area and drag it to the desired position by moving mouse cursor without releasing mouse button or using the key arrows without releasing the mouse button

3 Release the left button.

Due to Variable Editor internal organization, a Mbf cannot be positioned into the same position of another membership with equal properties (i.e. shape and dimensions).

- **To change shape:**

1 Select the Mbf vertex to be moved, clicking on it with the left mouse button.

2 Move the vertex to the desired position dragging it or using the key arrows while retaining the left mouse button. Release the left button.

If the vertex is forced to go outside the variable domain, mouse cursor leaves x axis to run on the domain boundary, pointing to the intersection between the related shape side and this boundary.

Using AUTOFILL MBF (only for input variables)

To define a set of Mbfs equally spaced, the Autofill command can be used. A dialog box allows to specify the desired options.

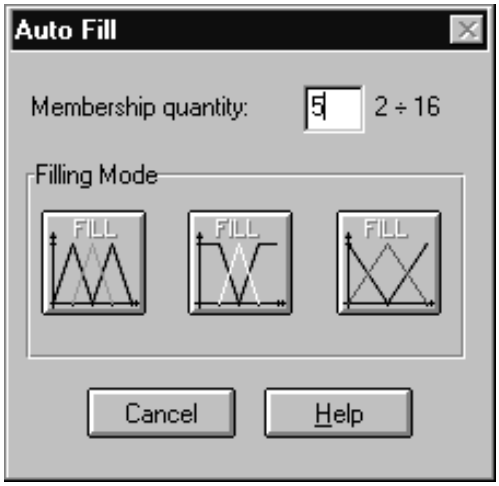
To draw the Mbfs set:

1 Select the variable, clicking on its window or opening it.

2 Choose AUTOFILL from EDIT > MBF menu.

3 Enter the desired number of fuzzy sets (num Mbf).

4 Press the desired Filling Mode button.

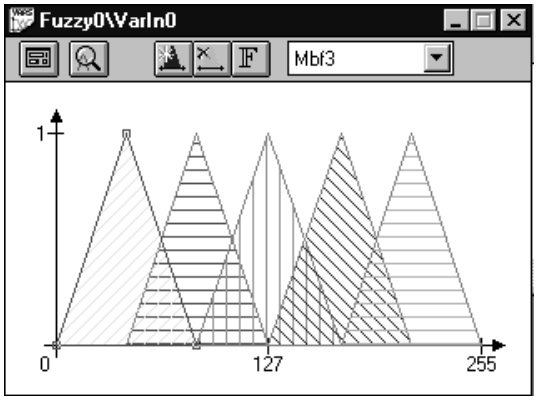


The AutoFill mode defines at least two Mbfs. Therefore it is active only if less than MaxRules-1 Mbf have already been defined. Max_Rules being the maximum allowed number of rules.

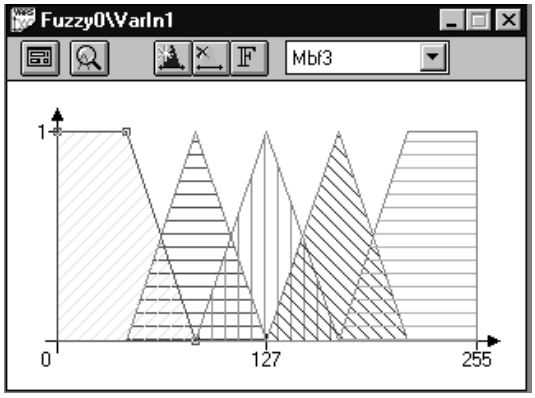
The Max_Rules depends on the number and allocation of Membership Functions already defined inside the same and the other Fuzzy Blocks. Anyway, no more than 16 Mbfs per time can be defined.

The filling modes work as follows:

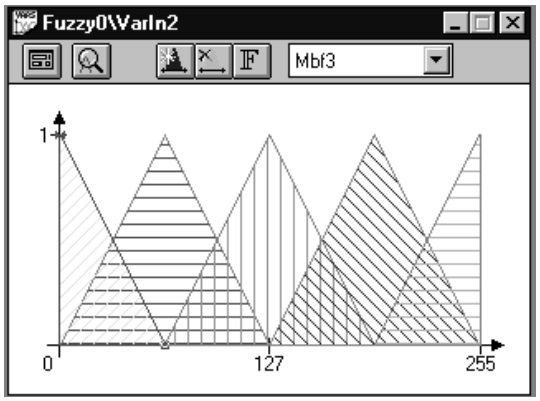
Draws n isosceles triangular Mbfs. The semibase of new Mbfs is calculated splitting the Universe in n+1 equal intervals. Since x-values must be integers, the remainder of the integer division of the domain width by n+1 is added to the centrail interval (or is shared out between the two central ones, if n=1 is even).



Draws two external rectangle trapeziodal Mbfs. All other Mbfs are triangular. The semibase of new Mbfs is calculated splitting the universe in n+1 equal intervals. Since x-values must be integers, the remainder of the integer division of the domain width by n+1, if any, is added to the central interval (or is shared out between the two central ones, if n is even).



Draws two external rectangle triangular Mbfs. All other memberships are triangular. The semibase of new Mbfs is calculated by splitting the Universe in n-1 equal intervals. Since x-values must be integers, the remainder of the integer division of the domain width by n-1, if any, is added to the central interval (or is shared out between the two central ones, if n is even).



How to delete a Mbf

- **To delete a Mbf:**

1 Select the Mbf clicking on it on the drawing area or selecting its name on the drop down list

2 Choose DELETE MBF from EDIT > MBF menu or from the right mouse button popup menu or press DELETE key.

When deleting a membership used by existing fuzzy rules you are asked for confirmation. By erasing such a membership these rules are deleted too. The number of fuzzy rules involved into the membership deletion is provided with the confirmation request.

Note *Deleting a Mbf belonging to a shared variable will cause the elimination of the Mbf in each fuzzy system when the variable is used (see shared and copied variables paragraph).*

AutoShift & SemiBase

When a new MBF is generated by means of the MBF NEW command, the base width and the distance of each Mbf is set by default to the SemiBase and AutoShift values.

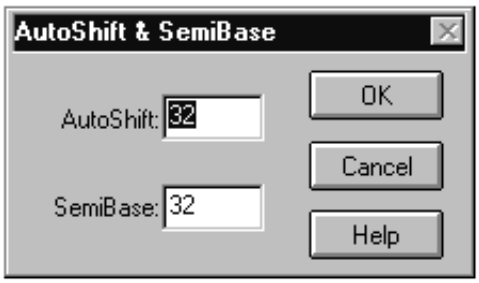
- **To change these values:**

1 Choose SET AUTOSHIFT & SEMIBASE from EDIT menu.

2 Change values.

3 Press OK.

The AutoShift and SemiBase options are related to the creation of a new Mbf. When a new Mbf is generated by means of MBF NEW command, its centroid will be placed:



- On the cursor position, if a Mbf didn't already defined in the same position.
- At a distance equal to the AutoShift value from the closest Mbf, otherwise.

Triangular Mbfs have by default base half-width equal to the SemiBase value. Different bases can be obtained changing SemiBase option or customizing each Mbf. You can modify default values for AutoShift and SemiBase values via the SET AUTOSHIFT & SEMIBASE command from the EDIT menu.

On this command a dialog box pops up, showing current default values and allowing users to modify them. These values are set using a ST52x301 internal measurement unit, and are independent from the chosen dimension of the variable domain.

Due to hardware constraints of the ST52x301 processor family, the AutoShift and SemiBase values must belong to the [1,255] integer interval.

Mbf Report

Variable Editor gives a textual description of variable Term Set.

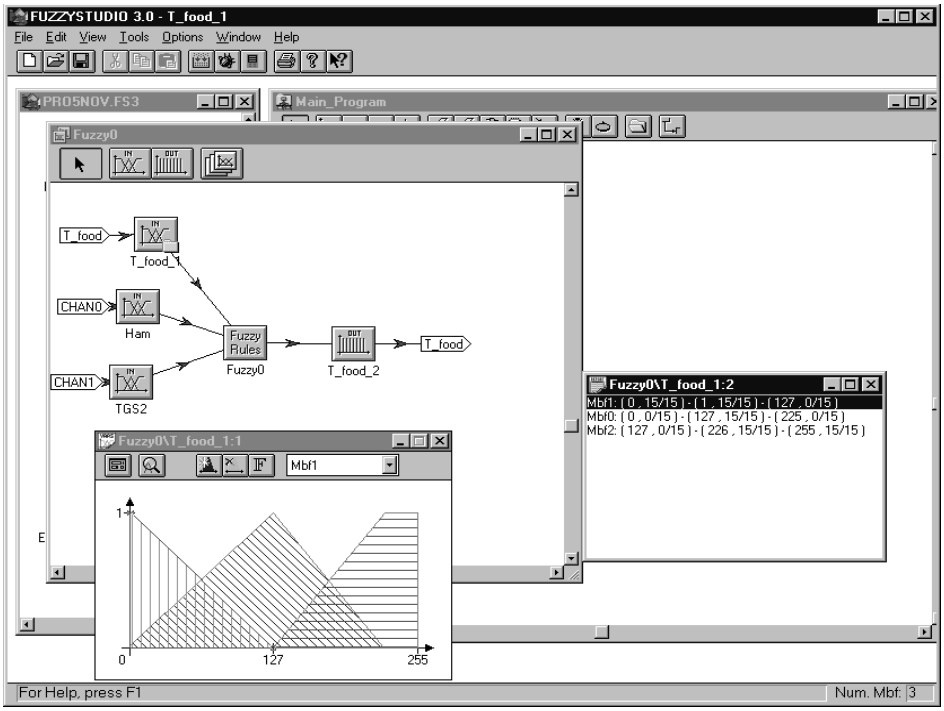
- **To visualize a variable report:**

1 Select the variable clicking on its windows or opening it.

2 Choose MBF REPORT from VIEW menu

Report window content is organized by rows, each row containing:

- Mbf name and its vertices coordinates, in case of input variable;
- Mbf name and its crisp value, in case of output variable;



Report window rows are scrollable and mouse selectable. You can run membership related commands either from a variable window or from the related report one. Membership pop-up menu can be activated pressing the right mouse command while on the report window.

The active report window can be closed by using again the MBF REPORT command from the VIEW menu (this command acts as an on/off local switch, related to the active variable window only) or via the CLOSE command from system menu (accessible using the button on the window left upper corner). Closing a report window not affects the related variable window, while closing a variable window closes the related report too.

When a report window is open, its fonts may be changed, performing the following operations:

- Click the right mouse button and select CHOOSE REPORT FONT from VIEW menu.

or

- When the Report window is in foreground select the "Choose report Font" command from the View menu.

Shared and Copied Variables

Besides the creation of a new variable, within a fuzzy block, it is possible to share or copy existing variables within other fuzzy blocks.

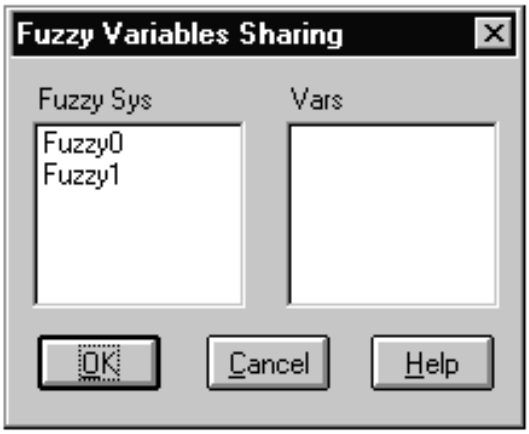
Shared Variables

A shared variable is a variable used in several different Fuzzy Blocks, with the same characteristics, and will therefore use the same memory locations to store the Membership Functions, this means that the variable is the same. Then, each change performed on one of the shared variables will have effect on all of them.

To insert a shared variable:

- Select SHARE VAR from the INSERT menu in the Fuzzy System Editor window or click the apposite toolbar button of this window.
- A selection window is open, where on one side you will find the list of fuzzy systems defined and on the other one there will be the variables defined inside the selected block.
- Choose the fuzzy system to which belongs the variable to share and select it.

Note *The selection of a variable belonging to the current Fuzzy Block will not produce any effect.*



- The window disappears and the mouse pointer is enabled to insert the variable on the fuzzy system.
- In both fuzzy systems that share the variable, the icon that represents it change its shape indicating that such a variable is shared by several blocks.

The name of the shared variable is the same in all the blocks in which it is used: a rename of one of them will determine the change in the other fuzzy systems in which it is shared.

Note *It is not possible to insert a shared variable in a fuzzy system that has the same name of an already existing fuzzy variable. In this case, rename this last variable and try again to insert the shared variable. Moreover, consider that it might not be possible to insert a shared variable when new variables, using the same address space of the variable to share, have been defined.*

Copied Variables

The copied variables are variables already defined in a fuzzy system. This means that their characteristics can be copied in other fuzzy systems. The substantial difference between shared and copied variables is that a copied variable is an other variable different from the original one. A copied variable can be inserted provided that a memory space is available (refer to the chapter "Variable Constraints Definition in a Fuzzy Block).

The editing environment of a copied variable is distinct from the variable of origin. If a variable having the same name already exists in the fuzzy system, the name of the variable to be copied is modified adding a progressive number to its name.

Finally, each change performed on the original variable or on the copied variable is not reflected since, as stated earlier, these are distinct variables. This can be useful whenever you want to define a variable with similar characteristics but not coincident with another: first copy the variable and then make the necessary changes.

It is possible to copy, cut and paste a Fuzzy Variables Block with its content as a normal block:

- Select the Variable(s) to COPY/CUT selecting the command in the EDIT menu or clicking the related toolbar button.
- Open the destination Fuzzy Block.
- Select PASTE command from the EDIT menu or click the related toolbar button.
- The window disappear and the pointer is enabled to insert the variable in the fuzzy system.

How to Export Data to Clipboard

The Variable Editor allows to export data to the system clipboard. Both membership drawings and MBF reports can be exported. Perform the following commands:

1 Select the variable window to export.

2 Choose COPY ON CLIPBOARD from EDIT menu.

The COPY ON CLIPBOARD command, invoked by the CTRL+C short key too, acts on the active variable and is available only if at least one variable or a report window are opened or iconized. This is a sensitive command. Its label is COPY ON CLIPBOARD AS "BITMAP", when you are working on bitmap images, or COPY ON CLIPBOARD AS "TEXT", when you are working on a text editor.

You can then manage these data as any other clipboard content: as the clipboard is a global Windows object, shared among all the currently running applications, users can paste these data into documents of other applications such as Windows Write or Microsoft Word for Windows (e.g., to produce a detailed project documentation), or can put these data in the Windows Clipboard application to store them for later use. At the same time you must be aware that any copy operation, from any running application, replace the clipboard content with new data.

Fu.L.L. Importer

Fu.L.L. (Fuzzy Logic Language) Importer is a tool that allows to import, in a Fuzzy Block, fuzzy systems generated by the fuzzy software tools produced by STMicroelectronics : A.F.M.1 & 2 (Adaptive Fuzzy Modeller) for the automatic generation of the fuzzy models by starting from the input/output patterns and FUZZYSTUDIO™ 2.0, the development system for the programming of W.A.R.P. 2.0 processor.

Fu.L.L. (Fuzzy Logic Language) is a description language of the fuzzy systems allowing to exchange data between the various Fuzzy Logic software tools.

To import a fuzzy system in the project:

1 Insert a Fuzzy Block, by using the blocks editor.

2 Open the block keeping the fuzzy system window in foreground.

3 Select the item Import Fu.L.L. from INSERT menu.

4 An Open File dialog box is open.

5 Search and select the file .ful you are interested in.

6 Press OK button or double click on the file name.

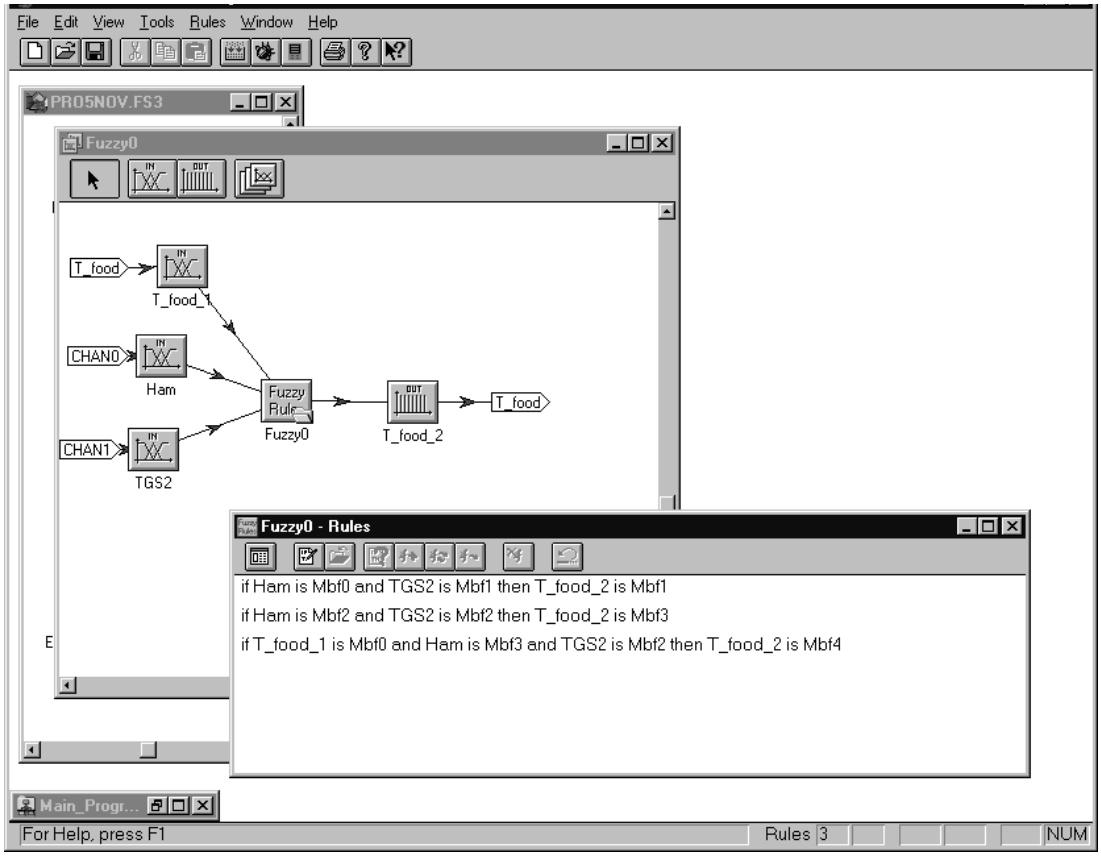
A fuzzy system in Fu.L.L., besides being generated automatically by above tools, can also be generated by using a normal text editor. For further information on the F.u.L.L. syntax and semantic refer to Appendix E. - F.U.L.L. - Fuzzy Logic Language.

Rule Editor

Rule Editor is the tool devoted to manage with fuzzy rules. To run Rule Editor, double-click on the RULES BLOCK from the Fuzzy System Editor window.

The Rule Editor environment is opened showing, if already defined, the list of the rules.

In order to define fuzzy rules, using Rule Editor, it is necessary to define at least one input and one output Variable with associated Membership Functions. Rule Editor cannot be activated until these minimal definitions have not been done.



Defining Fuzzy Rules with Rule Editor

The rules related commands are grouped into the RULES menu.

There are two ways to write rules: by using Guided Editing or Manual Editing.

In Guided Editing a paddle containing keywords, if-then operators, variables and membership functions is used for quick definition. This editing mode allows you to select automatically only syntactically correct objects.

In Manual Editing, rules must be defined manually using the keyboard as a normal text editor. Before being inserted on the list, the rule is syntactically checked to guarantee its correctness: if an error occurs, its description is given in the Output window. Manual Editing allows to delete or to modify already defined rule. After being modified, a rule can replace its previous version or be added to the list. Multiple deleting is possible if a multiple selection is done.

It is possible to go to a precise rule by using EDIT -> GO TO command o pressing F5. A dialog is shown allowing the user to choose the rule number.

To copy one or more rules onto the Clipboard, select them from the rule list using the mouse, then choose COPY from EDIT menu or use CTRL + C. You can also perform the cut commands choosing it from the EDIT menu or using CTRL + X.

You can also print all rules information about the current project by using the PRINT command or **CTRL+P**.

The Rule Editor Window

This section provides an overview of the major elements of the Rule Editor window, such as menus, toolbar and status bar. For additional information, see the index and online Help.

Rule Editor menus

All Rule Editor commands are grouped into a menu hierarchy accessible from the main window menu. Menus and items are context-sensitive: at each time only relevant commands for the selected object or for the current project status will be enabled.

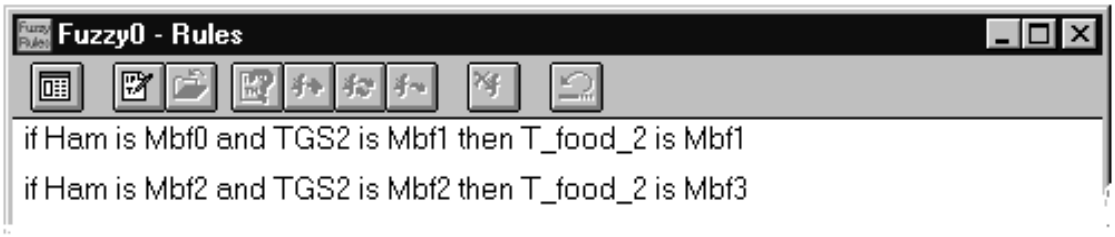
Available menus are:

FILE	Provides save and print utilities, a list of recently used files and the exit command.
EDIT	Provides standard editing and GO TO commands.
VIEW	Contains commands for control bars display/hide, font selection and move through the error list.
TOOLS	Contains commands for switching to FUZZYSTUDIO™ 3.0 tools: Compiler, Debugger and Programmer.
RULES	Contains rule related commands, for Manual and Guided editing.
WINDOW	Contains commands related to window management.
HELP	Contains commands for help on topics and information on FUZZYSTUDIO™ 3.0.

Rule Editor Toolbar

For convenience, the most frequently used commands of Rule Editor can be executed by choosing the corresponding buttons on the toolbar. Toolbar buttons are context-sensitive: only the commands in relation with current project status are enabled.

Toolbar is a user selectable item: you can choose to hide or show it by using the related commands of VIEW menu.



Rule Editor Status Bar

A status bar is available at the bottom of the application main window. This bar displays a brief description of the selected command, the number of rules defined by the user, the number of the selected rule in the list, the status of the CAPSLOCK and NUMLOCK keys.

To get a quick help on a command item (i.e., a menu command or a toolbar button), just position the mouse pointer on the toolbar without clicking, a brief command description will be shown on the leftmost field of the status bar, in addition to the type shown near the button.

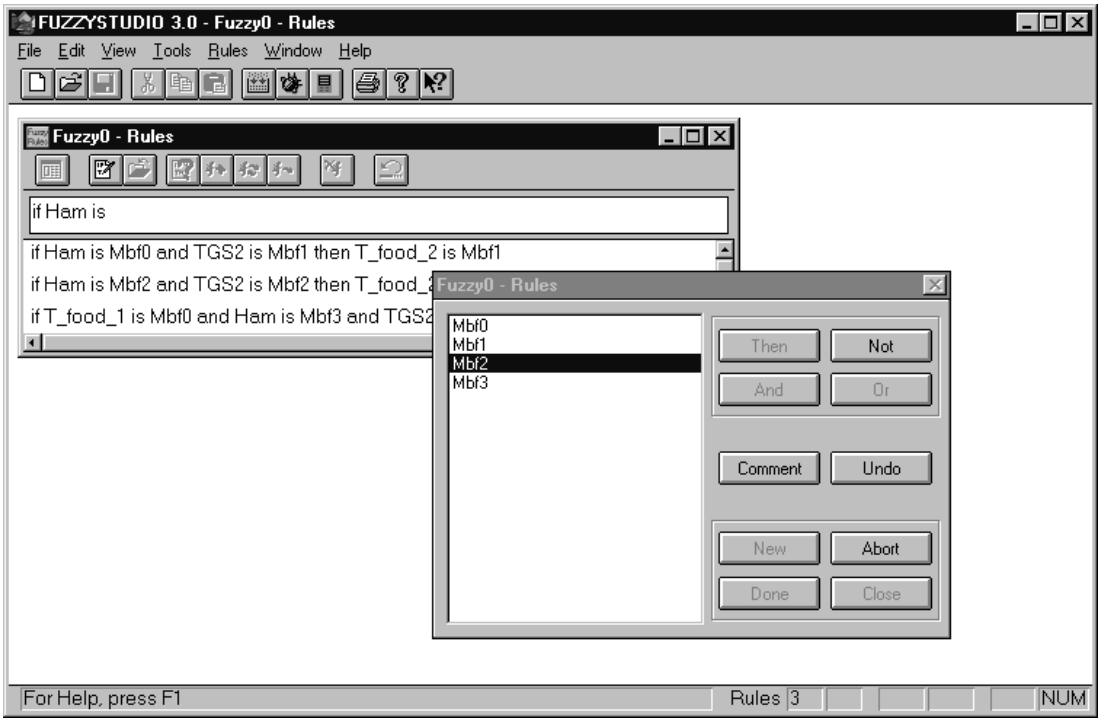
How to Write a Rule Using Guided Editor

Guided Editor uses the editing paddle that can be activated choosing RULES GUIDED or pressing **CRTL+G** or the toolbar button.

Select keywords, operators and others by clicking on paddle buttons.

Note that during Guided Editing:

- Keywords IF and IS are automatically included when necessary, that is, respectively, at the start of the rule and after selecting a variable name.
- The rule is shown in the upper side of the edit box. It is not possible to write in this edit-box during Guided editing.
- It is possible to correct manually the rule switching to Manual editing.
- If a mistake occurs it is possible to UNDO the previous operations, or ABORT the rule editing. Notice that UNDO could not be possible after modification of variables or membership functions names using the Variable Editor.
- To include comments inside the rule, click the COMMENT button in the paddle to edit the comment text. Comments can be included anywhere in the rule.



Editing the Antecedent

Notice that in this list-box, Variables or Membership Functions are shown according to the syntactical correctness of the rule; in addition, the Membership Function list is always related with the selected Variable.

1 Click New button in the paddle: the IF keyword is shown on the edit-box.

2 Choose one of the input variables listed on the left side of the paddle, double clicking on it. The input variable name will be shown in the edit-box followed by the keyword IS, whereas in the list-box the related Mbf list appears. Note that only variables having at least one membership function are listed.

3 Add NOT modifier, if necessary, clicking on the apposite button.

4 Select Mbf, double-clicking on its name on the list-box.

5 Click AND or OR button if rule has more antecedent terms and go back to point 2, for further insertion of input variables, otherwise go to the following section Editing the Consequent.

Note: AND operator has higher priority than OR operator.

Editing the Consequent

1 Click THEN on the paddle.

2 Select output variable double clicking on the variable name in the list-box.

3 Select the variable crisp value, double clicking over the associated label. You can also click over the Mbf label or write the crisp value and push ENTER button or click over Accept button.

4 Click AND if the rule has more consequents and repeat the selection of an output variable and the association of its crisp value.

5 Click DONE: the rule will be added in the rule list and the status bar is updated.

Note: If the crisp value is written directly in the dialog-box, this operation does not determine its graphical representation in the Mbf drawing window because the value is directly translated by Compiler machine code, without the need of a point-by-point graphical definition of the related membership function.

Using Manual Editor

Manual editing mode can be activated choosing MANUAL > ACTIVATE from RULES menu or pressing CTRL+M or the toolbar button. An edit box will appear in the upper side of the window.

Manual Editor allows you to edit more than one rule at the same time. These rules might be syntactically checked and/or inserted in the list.

By means of the MANUAL EDITOR it is possible to manually edit a new rule or to modify already existing ones.

How to write a Rule

To use Manual Editor for the definition of fuzzy rules follow these steps:

1 Write the rule in the edit-box above the rule.

2 You can now check the rule syntactic correctness, pressing MANUAL CHECK from RULES menu or using F3 or clicking the toolbar button.

3 Write, if necessary, comments including the text between these characters:

/**/ the string between two sequences of characters is excluded by syntactical check

.// the string after two sequences of characters is excluded by syntactical check

You can write more than one rule at the same time either by writing them on one row and separating each rule by inserting a semicolon or by using one row for each rule.

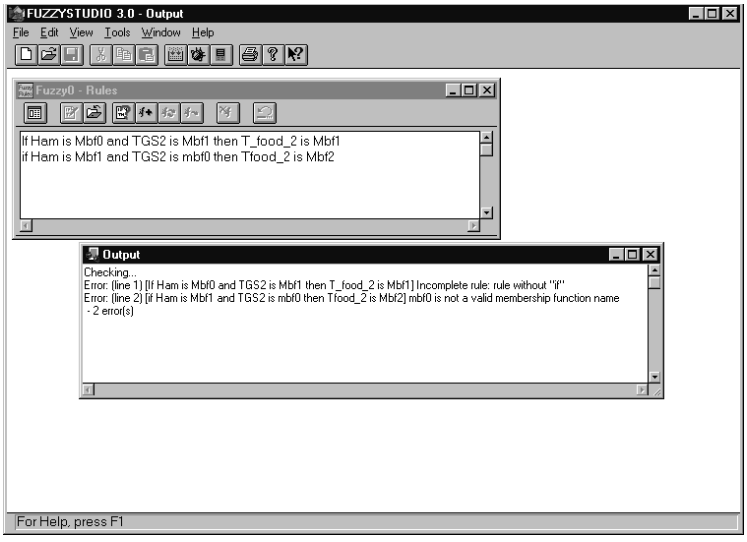
You can also write one or more rules by modifying the existing ones. In this way the rule editing is performed quickly because the selected rule(s) is copied in the edit-box and it won't be necessary to write the entire rule(s) again:

1 If you want to change only one rule: select the one you want to use as a model by double-clicking on it.

If you need to modify more than one rule: copy the selected rules in the clipboard and then past them in the Manual Editor windows.

2 Modify the parts of the rule(s) you need to change.

3 You can now check the syntactic correctness, pressing MANUAL CHECK from RULES menu or using F3 or clicking on the toolbar button.



List updating

When the rule(s) definition has been completed, the rule(s) can be inserted in the rule list by means of the following MANUAL commands in the RULES MANUAL menu:

How to add rule(s)
Choose RULES MANUAL ADD or click the toolbar button. The current rule is syntactically checked and inserted at the end of the rule list.

How to insert rule(s)
Rule inserting is possible in manual Editing mode. it consists in the insertion of the edited rule before the selected ones:

1 Select the rule by clicking on it.

2 Choose INSERT from RULES MANUAL or use INS or the toolbar button.
The current rule is syntactically checked and inserted just before the selected rule(s).

How to replace rule(s)
Rule replacing is possible only in Manual editing.

1 Select the rules that must be replaced, clicking on it in the rule list.

2 Choose RULES MANUAL REPLACE or press **CTRL+R** or the toolbar button.

How to Delete Rules

1 Select one or more rules.

2 Choose DELETE from EDIT menu or use DEL or click the toolbar button.

3 Press YES, on the Rule Editor confirmation box.

To select multiple sequential rules in the list, click the first one and then drag the cursor to the liast item, or click the first rule you want to select, press and hold down SHIFT, and then the last rule (or via Keyboard by pressing SHIFT + arrow keys).

To select multiple nonsequential rules in the list, press and hold down CTRL, and click each item you want to select.

To cancel the selection, release SHIFT, and then click any item.

How to Print the Rules' List

1 Check the correctness of the printing options for the printer, choosing FILE PRINT SETUP

2 Choose PRINT or press **CTRL+P** or the toolbar button
Before printing, the user can view the document choosing PRINT PREVIEW in the FILE menu.

Rule Editor View Options

The VIEW menu allows you to customize the RULE EDITOR window. You can display or hide the toolbar and the status bar choosing the related command from the VIEW menu.

You can set the font used to show the rules by means of the FONT VIEW command. Performing this task a standard MS-Windows dialog box will pop up and you can change font type and size.

Rule Editor Constraints

Rule Editor allows to write rules having a maximum of 8 antecedents and 2 consequents.

However, rules with more than 4 antecedents or more than 1 consequent are splitted, during compilation, into equivalent simpler rules having up to 4 antecedents and one consequent. ST52x301 executes partial computations and then evaluates the total antecedent part of the rule.

A maximum of 256 rules can be defined.

Rule Grammar

<RULE> ::= **if** <ANTECEDENT> **then** <CONSEQUENT>.

<CONSEQUENT> ::= <CONSEQUENCE> **and** <CONSEQUENCE> **|**
<CONSEQUENCE>.

<CONSEQUENCE> ::= <IDENTIFIER> **is** <OUTPUT>.

<OUTPUT> ::= <IDENTIFIER> **|**
<CONSTANT>.

<ANTECEDENT> ::= <ANTECEDENT> **or** <ALTERNATIVE> **|**
<ALTERNATIVE>.

<ALTERNATIVE> ::= <ALTERNATIVE> **and** <CONTRIBUTE> |
<CONTRIBUTE>.

<CONTRIBUTE> ::= (<ANTECEDENT>) **|**
<PREMISE>.

<PREMISE> ::= <IDENTIFIER> IS <MODIFIEDMBS>.

<MODIFIEDMBS> ::= NOT <IDENTIFIER> **|**
<IDENTIFIER>.

<IDENTIFIER>::=<LETTER><DIGITLETTER> |
<LETTER>.

<DIGITLETTER> ::= <LETTERS><DIGITLETTER> **|**
<DIGIT><DIGITLETTER>|
<LETTERS> **|**
<DIGIT>.

<LETTERS> ::= <LETTER> |

_.

<CONSTANT> ::= <INTEGERPART><SECONDPART> **|**
<INTEGERPART>.

<INTEGERPART> ::= <SIGN><DIGITSEQUENCE> **|**
<DIGITSEQUENCE>.

<SIGN> ::= **+** **|** **-**.

<DIGITSEQUENCE> ::= <DIGIT><DIGITSEQUENCE> **|**
<DIGIT>.

<SECONDPART> ::= <DECIMALPART> |
<EXPONENT>.

<DECIMALPART> ::= <DIGITSEQUENCE><EXPONENT> |
<DIGITSEQUENCE>.

<EXPONENT> ::= **e** <INTEGERPART> |
E <INTEGERPART>.

Error Messages

Using FUZZYSTUDIO™ 3.0 Rule Editor the following messages and warnings can occur:

"char" not allowed character
The character "char" is not allowed in the context.

Generic check error
Internal Error: contact STMicroelectronics, Fuzzy Logic B.U.

Incomplete rule: a membership function name was expected.
A membership function term is missing in the rule. Complete the rule with the appropriate term.

Incomplete rule: an input variable name was expected
A variable name is missing before a "is" keyword in the entered rule. Add the keyword in the appropriate place or check for sintax errors.

Incomplete rule: an output variable name was expected
An output variable term is missing in the rule. Complete the rule with the appropriate term.

Incomplete rule: "is" keyword was expected
The keyword "is" has not been written after a variable name to specify the membership function. Add the keyword where it is missing.

Incomplete rule:not closed comment
The comment statement in the rule was not closed with the */ character. Add the character */ at the end of the comment.

Incomplete rule or empty rule
The rule in editing has not been completed before entering it or a empty row has been entered. Complete the rule with the correct syntax.

Incomplete rule: rule without "if"
The keyword " if " is missing at the start of the entered rule. Add the keyword in the appropriate place or check for syntax errors.

Incomplete rule: rule without "then"
The keyword "then" is missing at the start of the consequent part of the entered rule. Add the keyword in the appropriate place or check for syntax errors.

"name" is an invalid keyword
The specified keyword "name" is not allowed in the context. Check for syntax errors.

"name" is not an input variable
The specified name for antecedent term does not belong to the list of input variables. Check for syntax errors.

"name" is not an output variable

The specified name for consequent term does not belong to the list of output variables. Check for syntax errors.

"name" is not a valid membership function name

The specified name does not belong to the list of membership functions associated to the variable. Check for syntax errors.

Not enough memory

The available memory is not enough to perform the operation. Try again after closing some application in your computer.

Too many terms in antecedent; 8 at most

More than 8 antecedent terms have been specified in the rule. Try to split the rule in two or more equivalent rules.

Too many terms in consequent; 2 at most

More than 2 consequent terms have been specified in the rule. Try to split the rule in two or more equivalent rules.

Unexpected string at the end of the rule

A not allowed string of characters has been found at the end of rule. Check for syntax errors or if the /* character for starting a comment is missing.

"value" is out of defined Universe of Discourse

The specified value for consequent is not in the specified range of the output variable.

Chapter 6

Arithmetic Block

The Arithmetic Block allows to carry out the arithmetic and logic operations that ST52x301 is able to perform.

To run the Arithmetic Block, double-click on the apposite icon in your flow-chart.

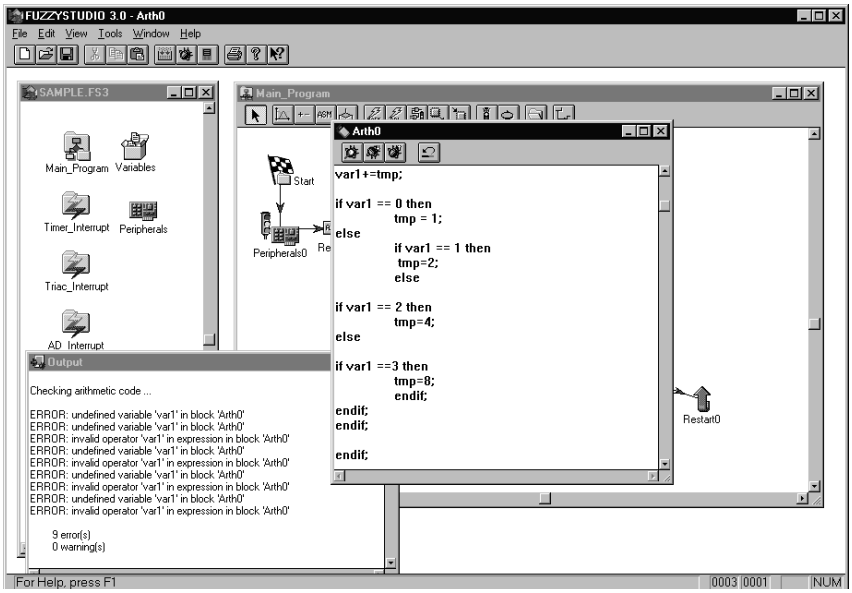
The Arithmetic Block Window

This section provides an overview of the major elements of the Arithmetic Block window, such as menus, toolbar and status bar. For additional information see the index and online Help.

Arithmetic Block menus

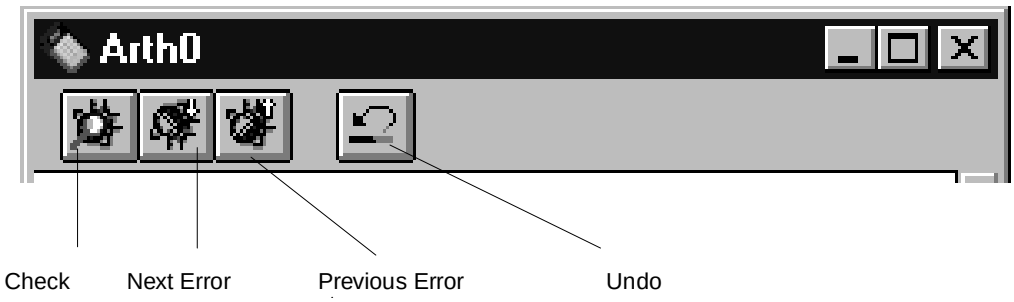
The Arithmetic Block menus are:

FILE	Contains commands to create, open, close and print.
EDIT	Provides standard editing and check commands.
VIEW	Contains commands to show or hide Toolbar, Status bar, go to previous or next error and font selection.
TOOLS	Contains commands for switching to Debugger, Compiler and Programmer.
WINDOW	Contains commands related to windows management.
HELP	Provides help on topics and information on the Arithmetic Block.



Arithmetic Block Toolbar

The Arithmetic Block includes a Toolbar to help you to perform the most frequently used commands quickly.



Arithmetic Block Status Bar

The Status Bar displayed at the bottom of the Arithmetic Block window contains some information about the task you are working on and the position of the cursor.

Using the Arithmetic Block

The Arithmetic Block is a free text editor that performs the following operations:

- UNDO (CTRL+Z) is a multilevel option that allows you to undo most of the text editing and formatting operations accomplished using items of the EDIT menu:
- CUT (CTRL+X), COPY (CTRL+C) and PASTE (CTRL+V) to delete, move and copy text onto the clipboard.
- FIND locates a specified part of the text and Find Next (F3) looks for the next one.
- REPLACE is a function that allows to replace the specified text.
- SELECT ALL to select the whole text.

Moreover, it is also possible to change the font settings:

- Select FONT from VIEW menu.
- Select the character's setting and color from the FONT dialog box.

Instructions

The instructions allowed in the Arithmetic Block can be grouped as follows:

Mathematical instructions:
Assignment, sum and subtraction that use the following operators:

=, +=, -=, +, -

Conditional instructions
Instructions that use the following keywords: if, then, else, endif.

While else is optional, endif is compulsory to close each IF statement since there can be many different IF levels. The following comparison operators are allowed:

==, >=, >, <=, <, !=

Some logical functions can be used in a conditional expression:

IsBitSet, IsBitReset, IsOverflow, IsUnderflow, IsOutOfRange, TimerStatus, SciStatus.

Logic instructions
They use the following operators & (AND), | (OR), ~ (NOT), ^ (XOR).

The following functions are accepted for bit manipulation:

BitSet, BitReset, BitNot.

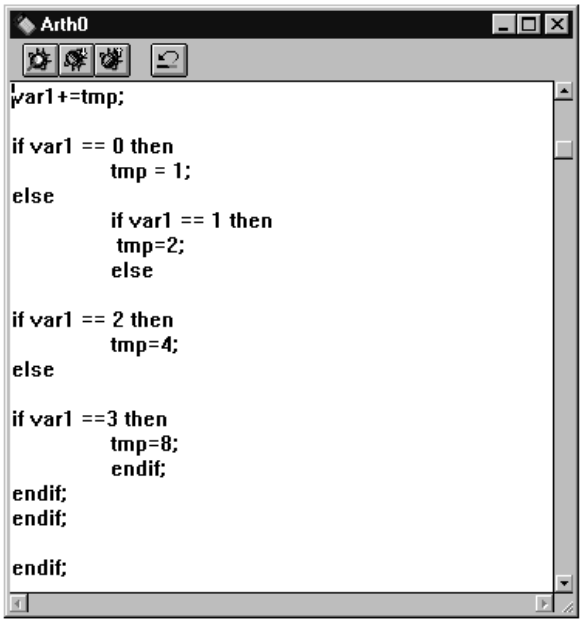
& has higher priority than |.

- Instructions must end with a semi-colon.
- Expressions must have no more than one operator (two operands).
- No brackets are accepted.
- The comment lines are allowed and preceded by //. It is possible to insert a comment text in several lines starting the comment block with /* characters and ending it with */.
- Operands allowed are the global variables and the local variables defined within the block and the predefined variables representing some peripheral's registers (see later).
- It is not possible to use the fuzzy variables within an Arithmetic Block since they can only be used in a Fuzzy Block.

Instructions Grammar

The Arithmetic block instructions operate on the Global and local variables declared in the same block by means of the apposite instructions of declaration.

The following is a description of the instructions syntax in which the objects between "[...]" indicate objects that can be omitted if not necessary while the ones between "{..." contain alternative objects separated by the symbol "|" . The objects that are not inserted between brackets cannot be omitted and must necessarily be present.



Expressions

The expressions must have the following syntax:

var assignment_operator [unary_operator] {var | constant} [operator {var | constant}] ;

Where:

- *var* is a Global variable, a local variable with an apposite instruction of declaration at the beginning of the block or a predefined variable.

The list of the predefined variables is the following:

Read-only variables:

CHAN0	Identifies the variable in which the value of the A/D 0 channel is set.
CHAN1	Identifies the variable in which the value of the A/D 1 channel is set.
CHAN2	Identifies the variable in which the value of the A/D 2 channel is set.
CHAN3	Identifies the variable in which the value of the A/D 3 channel is set.
TIMER_STATUS	Identifies the variable in which the Timer status is set.
SCI_STATUS	Identifies the variable in which the SCI status is set.
FUZZY0	Identifies the first Fuzzy output.
FUZZY1	Identifies the second Fuzzy output.
ADC_STATUS	Identifies the variable in which the value of the A/D Converter status is set.

Write-only variable

TRIAC_COUNT	Identifies the variable in which the Triac Counter value is set.

Read/Write Variables

TIMER_COUNT	Identifies the variable in which the Timer Counter value is set.
PORT	Identifies the variable with the values written/read in the parallel port.
SCI_BUFFER	Identifies the variable with the values written/read in the serial.

- *operator* is one of the following:

+	Sum
-	Subtraction
&	Logic AND
|	Logic OR
^	Logic XOR

- *assignment_operator* is one of the following:

=	*Simple assignment*	*a = b*	*b -> a*
+=	Assignment with sum	a += b	a = a + b
-=	Assignment with subtraction	a -= b	a = a - b

- *unary_operator*

-	Negation
~	Logic NOT

- *constant*: a constant value in the range [-128, 127] (signed byte) or [0, 255] (byte)

The expressions can contain at most one operator among *operator* and *unary_operator*. The expressions must always end with the character ";".

Note *Due to the fact that values are stored in 8 bit registers, they can easily go out of the range [0, 255], giving a result equal to -256 in case of overflow and 256 -|result| in case of underflow. The user can manage this situation by means of the functions IsOutofRange, IsOverflow, IsUnderflow, described later in this chapter.*

Examples:

```
a = b + c;
a += c;
b = 30;
b = a  |; c
c = - a;
b = ~ c;
```

Declarations

Declaration instructions allow to create temporary variables that are visible locally only within the Arithmetic Block in which they are declared. They have to be placed at the beginning of the block before any other instruction. An assignment can be associated to the declaration by means of a value or expression. This is the grammar:

{byte | signed |signed byte } { var | expression };

where:

byte	Byte type local variable declarator.
signed	Signed byte type local variable declarator.
signed byte	It is equivalent to signed.
var	A variable symbolic name.
expression	An arithmetic expression.

The instruction declaration must always end with the character ";".

Examples:

```
byte a;
signed b = 30;
```

Conditional instructions

The conditional instructions allow to modify the logic flow of the program within an arithmetic block. The grammar of these instructions is the following one:

```
if {var [relational_operator {var|cost}]| condit_function}
instruction_list
[else]
instruction_list
.
endif ;
```

where relational_operator can be one of the following:

==	equality
!=	disequality
>	greater than
<	less than
>=	greater equal to
<=	less equal to

Condit_function is one of the following functions of the language standard library:

- *IsBitSet (index_bit, variable_name)*

Verifies if the variable bit value is 1.

The parameter 'bit' varies from '0' to '7', while the parameter 'variable' can indicate any variable visible within the procedure that contains the conditional expression.

- *IsBitReset (bit, variable);*

Verifies if the value of a variable bit is 0.

The parameter 'bit' varies from '0' to '7', while the parameter 'variable' can indicate any variable visible within the procedure that contains the conditional expression .

- *IsOverflow();*

Verifies if the last logic-arithmetic instruction performed has generated an eventual overflow.

- *IsUnderflow()*;

Verifies if the last logic-arithmetic instruction performed has generated an eventual underflow.

- *IsOutOfRange();*

Verifies if the last logic-arithmetic instruction performed has generated an eventual overflow or underflow.

- *TimerStatus(param);*

Verifies theTimer status; the parameter indicates what to inspect about the peripheral status.

The possible parameters available for this function are the following:

SET	The function returns true if the timer is in Set status, False if in Reset status.
START	The function returns true if the Timer is in Start status, Flase if in Stop status.

- *SciStatus(param);*

Verifies the SCI status; the parameter 'param' indicates what to inspect about the peripheral status.

The possible available parameters for this function are the following:

TX_END	Verifies the end of transmission
TX_EMPTY	Verifies if the transmission buffer has been emptied.
NINTH_BIT	Returns 'true' if the ninth bit of the frame is 1.
OVERRUN	Verifies if an Overrun Error has occurred.
RX_FULL	Verifies if the reception buffer is full.
FRAME_ERROR	Verifies if a Frame Error has occurred.
NOISE_ERROR	Verifies if an error due to a noise has occurred.

These library functions can be used instead of one of the generic conditional expression operands. Their result will be 'true' in case of positive verification, 'false' otherwise.

Instruction_list is a list of arithmetic instructions. In case a constant or a variable is specified as conditional expression, then this will be false if the constant or the variable is 0, it will be true for all the other values.

For example:

```
if 0 then
a = b;
else
a = c;
endif;
```

Note *The instruction a = c is always performed*

```
if a<=60 then
a-=b;
endif;

if a != b then
a = b + c;
c = 0;
b = a + 10;
else
a = 0;
b += c;
endif
if a>=10 then
c=30;
else
c=80;
endif;

if a=b  then
c=b;
else
c=a;
endif;
```

```
if TimerStatus (START) then
c = 0;
else
c=128;
endif;
```

How to check the Instructions

The syntactic check of the text is carried out only under a user request. To start the check of the arithmetic instrunctions set:

- Select the item CHECK from the menu EDIT

or

- Click on the apposite toolbar button

The text can be saved inside a project even if not correct.
When you request a check an Output window will open displaying either eventual error messages and warnings or simply the message of successful check.

Double-clicking on an explanation error row in the Output window , the editor is automatically displayed in foreground and the row in which the error has been detected is highlighted allowing you to easily identify the errors.

To go to next error message line, you can do one of the following:

- Select NEXT ERROR from the menu VIEW.
- Click the apposite toolbar button.
- Press the short-cut key F4.

To go to the previous error message line do the one of the following:

- Select PREVIOUS ERROR from the menu VIEW.
- Click the apposite toolbar button.
- Press the short-cut key SHIFT + F4.

Chapter 7

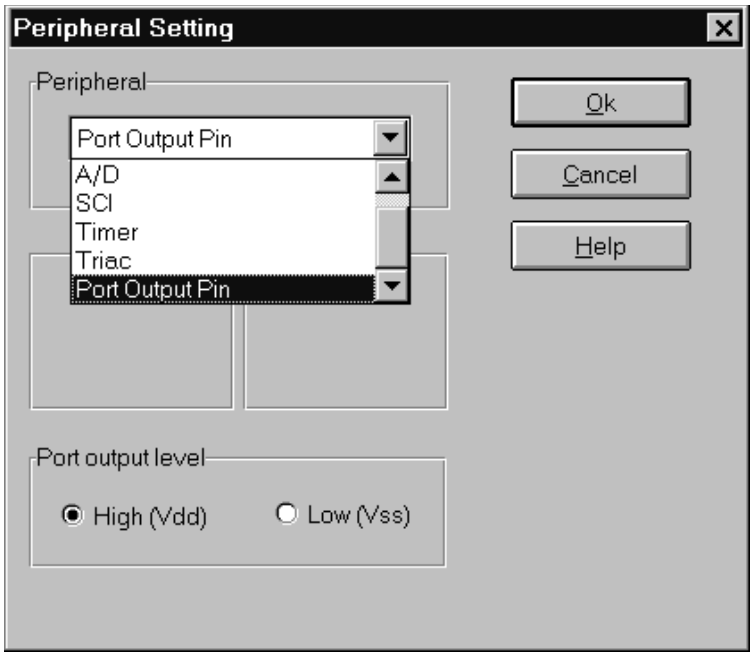
Peripherals Block

The Peripherals Block is used to enable or disable the peripherals' functionalities except for the Parallel Port, of which the Peripherals Block allows to write the 9th bit. It is important to notice that each peripheral block can act on a single peripheral.

Double-clicking on the icon representing the Peripherals Block, the Peripheral Settings dialog-box will appear. This consists of two main parts: a drop down list that allows to select the peripheral to be set and the relative selection check boxes that vary according to the peripheral you choose.

The peripherals you can choose are the following ones:

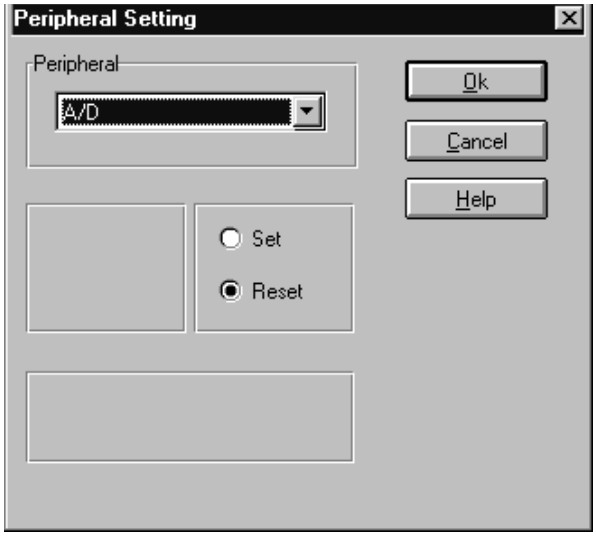
- A/D (Analog to Digital Converter)
- SCI (Serial Communication Interface)
- Timer
- Triac (Triac Driver)
- Port Output Pin



Note *By default, when you open the Peripherals Block for the first time, no peripheral is selected and the label NONE appears in the drop-down list box. Notice that no selection tool is available. Otherwise, the last settings performed on that block are displayed.*

A/D

When choosing A/D, the following dialog box is displayed on the screen:



The available settings are A/D Set and A/D Reset.

Note *When the program encounters a block that performs the conversion Stop, the peripheral stops reducing the current consumption and all its registers are reset, including the registers containing the converted data.*

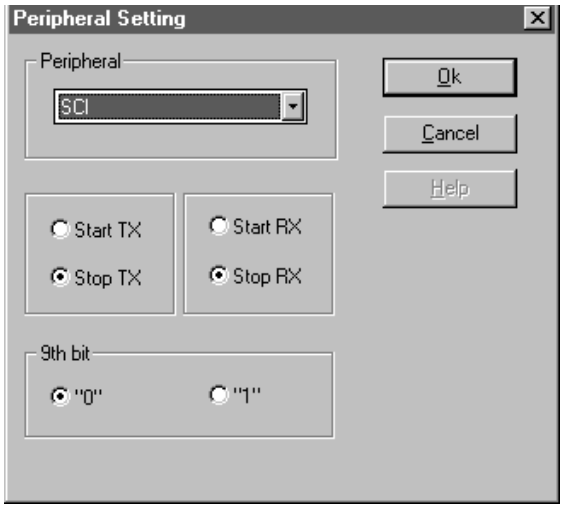
SCI

When choosing SCI, the dialog box will be the following one and the available command are:

Start / Stop Tx: Enables/disables serial transmission

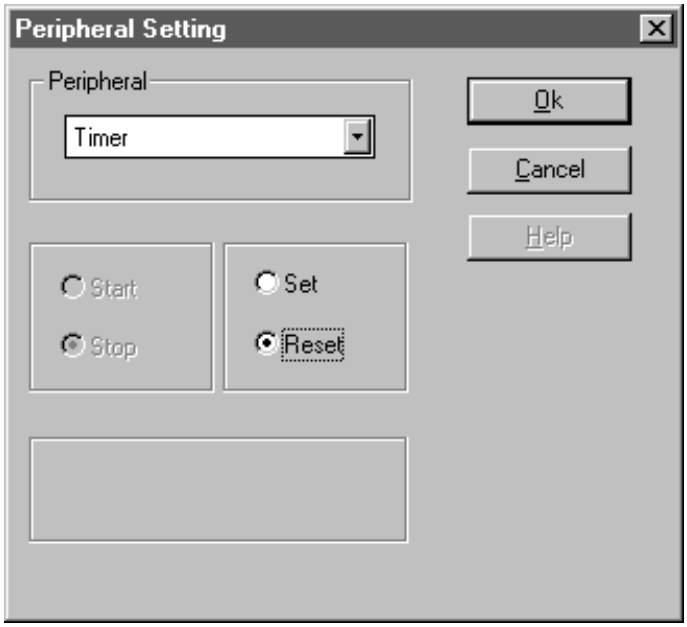
Start / Stop Rx: Enables / disables serial reception

9th bit: Specifies the 9th bit data value to be transmitted if this mode is activated by the peripheral setting.



Timer

The dialog box relative to the Timer will be the following:



It is possible to manage the following commands:

Set The peripheral is ready to start count.

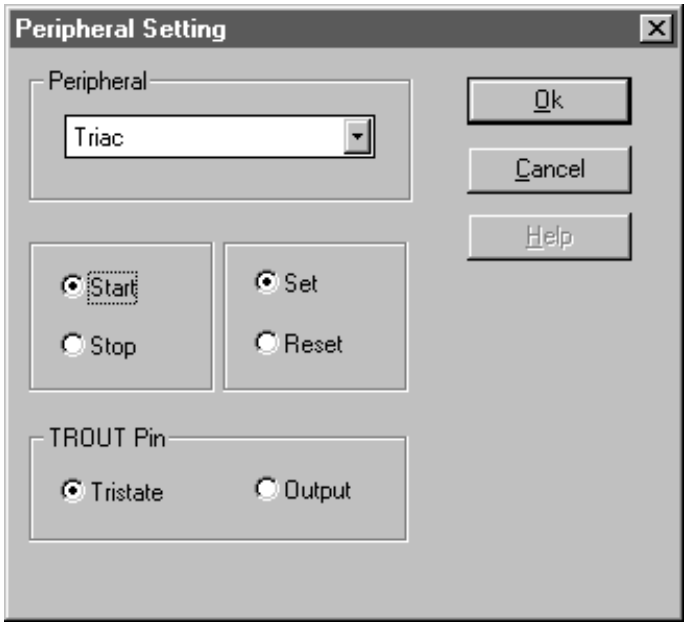
Reset Turns the peripheral off decreasing current consumption.

Start/Stop Starts/stops Timer count without resetting the counter, this means that a Start after the Stop resumes the count where it stopped.

This section is enabled only if the peripheral is in Set status.

Triac

Choosing the Triac, the dialog box will be the following one:



It is possible to manage the following commands:

Set Sets the peripheral in wait status for the count start.

Reset Turns the peripheral off decreasing current consumption and stopping it immediately.

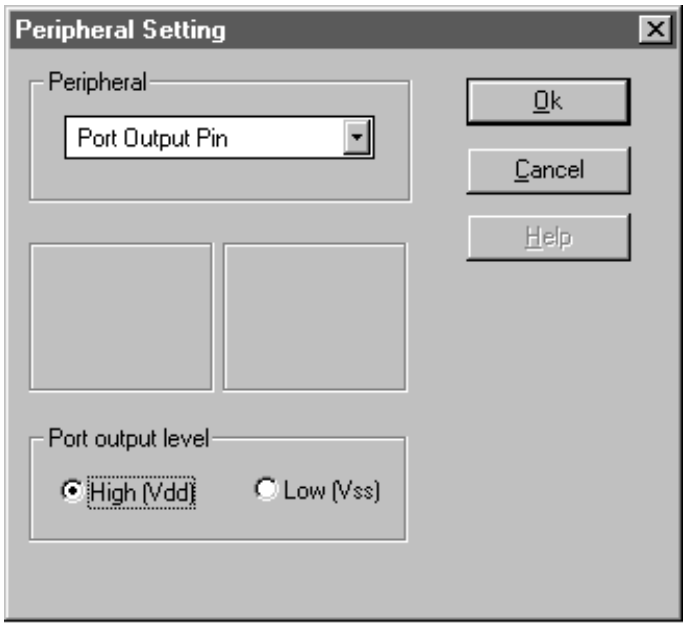
Start / Stop Starts / Stops the Triac functionment. This section is enabled only if the peripheral is in Set status.

Note *When the Triac Driver is stopped, it will stop at the finish of the counting in progress to avoid noise and control problems.*

TROUT Pin Allows to set the Triac Driver signal output pin in Tristate status to avoid eventual conflicts or in Output status to carry out the normal functionality. Notice that at the device start-up, this pin is in Tristate, then to drive the system it is necessary to put, by means of these controls, the TROUT pin in Output.

Port Output Pin

When choosing Port Output Pin, the dialog box will be the following one:



It is possible to set the output data to the port pin, that can take the value "High" (logic 1) or "Low' (0 logic).

Chapter 8

Interrupts Service Routines

The Interrupts signals determine, if enabled (not masked), a jump from the main program to the relative service routine.

The interrupts routines have the same characteristics of the main program and use a similar interface. The only difference, regarding the main program's routines, is that the routine ends with return from the interrupts command (RETI instruction) that determines the return to the main program where interrupted.

To write an interrupt service routine use the same technique of the main program, but in the window dedicated to it. Do as follows:

1 Double click on the icon in the Project window representing the interrupt you are interested in.

2 The interrupt routine editing window opens.

3 In this window, the IRQ block is automatically inserted. This represents the entry point of the editing routine: the first block has to be linked to this one.

4 Write the program using the blocks, as you do with the main program.

5 End the flow-chart with a RETI block. If this block has not been inserted, the Compiler generates an error message. More RETI blocks can be inserted when neccessary.

6 Do the same for the service routines of the other interrupts in use.

Note: *The interrupt routine windows can be open also by means of the commands in Windows Interrupts menu. A vector is associated to each interrupt. This is the address in which the service routine starts. This vector is automatically fixed by the Compiler in a transparent way for the user.*

Interrupts Configuration

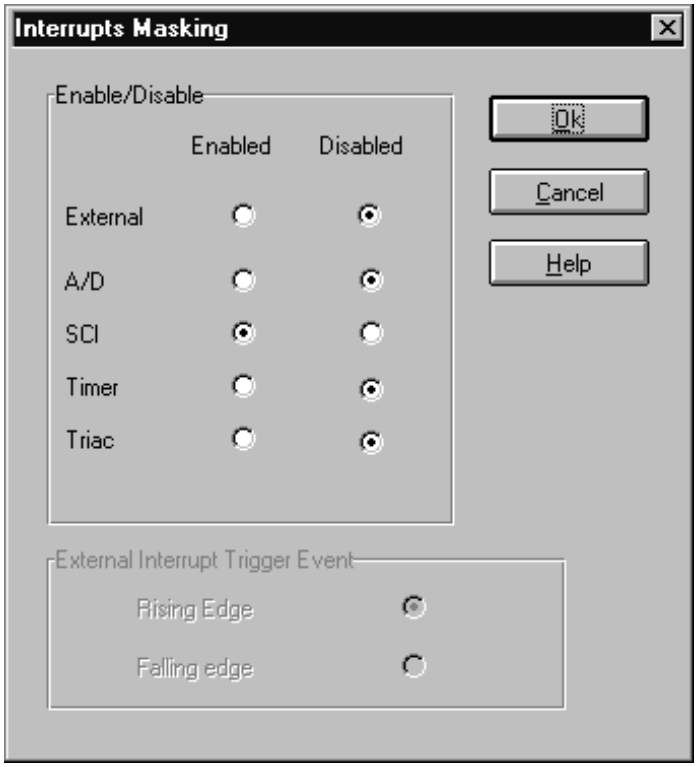
There are two blocks for the interrupt configuration: one block allows to enable/disable an interrupt and another one allows to change the interrupt priority.

Interrupts Mask Block

The Interrupts Mask Block allows to enable / disable the interrupts and to fix the External Interrupt Trigger, if this is enabled.

The dialog-box associated to this block is the one shown below:

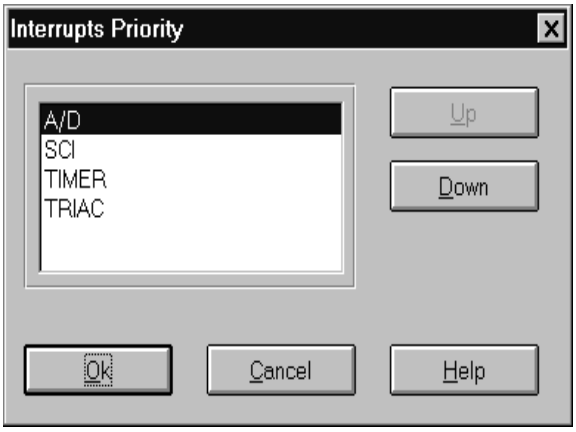
All interrupts can be enabled or disabled. However they are all disabled as default.



In case the external Interrupt is enabled, it is necessary to establish in the apposite 'External Interrupt Trigger Event" if this has to be determined by a Rising Edge (default) or Falling Edge.

Priority Configuration Block

This block allows to fix the interrupt priority, by using the following dialog-box:



The list shows the current priority order of the interrupts: the higher the interrupt is in the list, the higher the priority will be:

To change the priority:

1 Select the interrupt from the list.

2 Click the UP button to raise up the Interrupts priority and DOWN button to lower it.

3 Do the same with all the Interrupts until you reach the desired order.

Note: *The dialog window summarizes the priority of all the interrupts that will take the order fixed in the list after you click OK button.*

Chapter 9

Wait for Interrupt Block

The insertion of the Wait for Interrupt Block in the flow-chart makes the processor stops to wait for an interrupt signal when the instruction relative to this block is performed.

During the wait, no instruction is performed and the core of the processor is stopped, while the peripherals continue to work regularly.

When you have a request of interrupt, this is served performing the relative service routine. After the interrupt routine, the program restarts from the next block.

The Wait for Interrupt Block has not an editing environment associated because no specifics or settings are required.

It is useful to insert this block if you want to synchronize the program with events that can generate interrupt signals.

Note: *If there are no interrupt enabled or no interrupt signals is generated, the insertion of a Wait Block determines the processor stopping, that can be unlocked only through an external reset.*

Chapter 10

Assembler Block

The Assembler Block has been designed to allow the expert user to use the processor in all its potentialities, including the masked ones or the ones that are not allowed at a high level, writing directly in Assembler language.

A double-click on the Assembler block icon, opens the editing environment allowing to specify the program's instructions directly in ST52x301 Assembler.

The Assembler Block Window

This section provides an overview of the major elements of the Assembler Block window, such as menus, toolbar and status bar.

Assembler Block menus

The Assembler Block menus are:

FILE Contains commands to create, open, close and print.

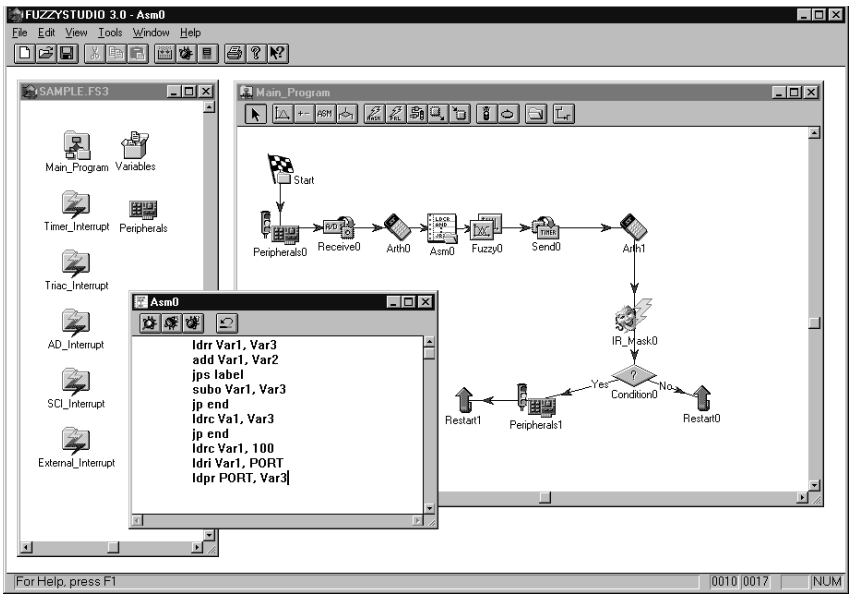
EDIT Provides standard editing and check commands.

VIEW Contains commands to show or hide Toolbar, Status bar, go to previous or next error and font selection.

TOOLS Contains commands for switching to Debugger, Compiler or programmer tool.

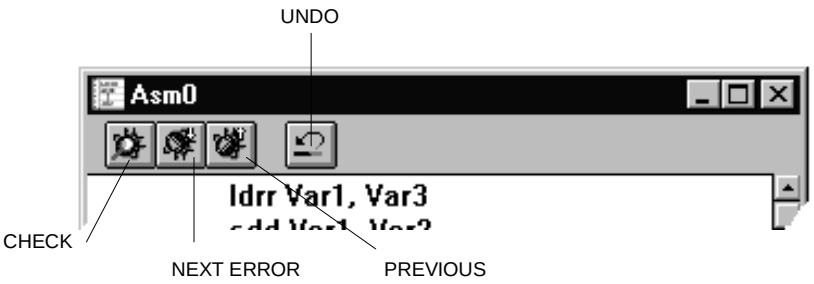
WINDOW Contains commands related to windows management.

HELP Provides help on topics and information on the Assembler block.



Assembler Block Toolbar

The Assembler block includes a Toolbar to help you to quickly perform the most frequently used commands.



Assembler Block Status Bar

The Status Bar displayed at the bottom of the Assembler Block window supplies information about the task you are working on and the position of the cursor. The bar also displays the status of the CAPS, NUM and SCROLL LOCK keys.

Using the Assembler Block

The Assembler Block is a free text editor.

- Undo (**CTRL+Z**) is a multilevel option that allows you to undo most of the text editing and formatting operations performed using items of the EDIT menu:
- Cut (**CTRL+X**), Copy (**CTRL+C**) and Paste (**CTRL+V**) functions let you delete, move and copy text onto the clipboard.
- Find locates a specified part of the text and Find Next (**F3**) looks for the next one.
- Replace is a function that allows to replace the specified text.
- Select All allows to select the whole text.

Note: Using Windows 95 you can access to the desired options by clicking the right mouse button over the client area.

Moreover, it is also possible to change the font settings.

- Select FONT from the VIEW menu.
- Select the character's setting and color from the FONT dialog box.

How to check Instructions

The syntactic check of the text is carried out at the user request. To start the check of the instructions:

- Select CHECK from the EDIT menu.

or

- Click on the apposite toolbar button.

The text can be saved in a project even if not correct.

When you request a check, an Output window will open displaying either error messages and warnings or the message of successful check.

Double-clicking on an explanation error row in the Output window, the editor is automatically displayed in foreground and the row in which the error has been detected is highlighted, allowing to easily identify the error.

To go to the next error message line, you can do one of the following:

- Select NEXT ERROR from the VIEW menu.
- Click the button on the local toolbar.
- Press the short-cut key **F4**.

To go to the previous error message line you can do the following:

- Select PREVIOUS ERROR from the VIEW menu .
- Click the button on the local toolbar.
- Press the short-cut key **SHIFT + F4**.

Assembler Block Instruction Syntax

The Assembler instructions written by means of the Assembler block follow a higher level syntax that allows to use the Global variables and Predefined variables. This leads to some constraints and the main ones are:

- it is not possible to insert assembler instructions relative to fuzzy functionalities.
- it is not possible to jump to labels external to the Assembler Block, in editing phase.

In order to optimise the use of the Assembler it is possible to use the FS3ASM.EXE tool, supplied with FUZZYSTUDIO™ 3.0 (Appendix D).

These are the instructions and relative syntax used in the Assembler block:

ldcf	REG_CONFxx	NN	es:	ldcf REG_CONF2 45
ldpr	Pred	Var	es:	ldpr PORT var1
ldrc	Var	NN	es:	ldcr var2 230
ldri	Var	Pred	es:	ldri var0 CHAN0
ldrr	Var	Var	es:	ldrr var1 var2
add	Var	Var	es:	add var1 var2
and	Var	Var	es:	and var1 var2
sub	Var	Var	es:	sub var1 var2
subo	Var	Var	es:	subo var1 var2
srx	Var		es:	srx var3
stx	Var		es:	rtx var2
jp	Label		es:	jp proc1
jpns	Label		es:	jpns proc0
jpnz	Label		es:	jpnz proc2
jps	Label		es:	jps proc5
jpz	Label		es:	jpz proc3
rint	N		es:	rint 2
udgi				
uegi				
mdgi				
megi				

where

REG_CONFxx =	a predefined variable, suitable for the registers' configuration and xx a number between 0 and 15.
NN	an integer decimal number included between 0 and 255.
Pred	a Predefined Variable
Var	a Global Variable.
Label	a label defined inside the block.
N	an integer decimal number included between 0 and 3.

For further details refer to Appendix C and ST52x301 Data Sheet.

Chapter 11

Conditional Block

A conditional block is inserted to modify the logic flow of the program according to a specified condition operated on the Global Variables defined by the user. From a semantic point of view, the Conditional Block is connected to a link in input and two in outputs, named YES and NO. The output links determine the logic flow of the program according to the condition contained in the block.

How to insert a Conditional Block

1 Choose CONDITIONAL BLOCK from INSERT menu or click the relative toolbar button.

2 Mouse pointer changes its status.

3 Click on the client area.

4 Connect the Conditional Block with the other blocks by using the links in the following way: at first, connect to the block to be performed if the condition is true; the first output link is YES by default.

To define the conditional expression:

1 Double-click on the conditional block or choose OPEN from the pop up menu that appears by clicking on the block with the right mouse button.

2 In the appearing edit-box, edit the conditional expression following the syntax and semantic described in the next paragraph.

3 Press CHECK button to test the correctness of the statement: the output window opens showing the errors if any.

4 Press OK button to confirm and close the window.

Note *The Keyword IF and THEN are considered as already inserted so you must omit them in the conditional statement, writing only the condition. See next paragraph.*

Conditional Block Grammar

The instructions of the conditional block operate on the Global Variables and Predefined Variables determine the logic flow of the program according to the condition specified; if this is true, the program will perform the part of the program relative to the blocks connected to the YES link otherwise it will perform the program connected to the NO link.

The objects between "[...]" can be omitted if not necessary while those between "{... }" are alternative objects and are separated by the symbol " |". The objects that are not inserted between brackets cannot be omitted.

The instructions have to be inserted in the conditional block edit box and express the conditional expression that has to ve evaluated.

This is the syntax:

{ *var* |*const*} [relational_operator { *var* | *const*}]

where the relational_operator can be one of the following:

==	equal to
!=	unequal to
>	greater than
<	less than
>=	greater equal to
<=	less equal to

var is a global variable or a predefined variable

const is a constant value in the range [-128, 127] (signed byte) or [0, 255] (byte).

If only a constant or a variable is specified as conditional expression, then this will be false if the constant or the variable is 0 and it will be true for all the other values.

The conditional expressions must **not** end with the character ";".

For example:

a <= b

a == b

a

0 (always false)

1 (always true)

a ! = b

a > 20

a >=10

a=b

Chapter 12

Send - Receive Blocks

Send block

The Send and Receive blocks let the program interact with the peripherals' registers. In particular, the Send Block allows to load the value of a Global variable in a peripheral's register while the Receive Block allows to load the value of a peripheral's register into a Global variable.

To insert a block of this kind, choose the relative toolbar button or select SEND BLOCK or RECEIVE BLOCK from the INSERT menu.

The icon of the Send and Receive blocks is context-sensitive, for instance it changes according to the block's settings and indicates the peripheral it interacts with.

Receive block

The icon that is inserted at the moment of the creation of the block is named NONE and this shows that no peripheral has been considered yet.

The label contained in the block representing the peripheral, changes contextually to the settings performed.

Note: *Predefined variables can also be used to interact with peripherals having Send-Receive blocks.*

How to Use the Send and Receive Blocks

Double-clicking on the block, you can open the command setting environment, that consists in two drop-down lists (see relative figures) from which you can select the Global Variables defined and the peripherals enabled according to the definitions in the Peripheral Configuration. Then, for example, if the A/D is set so as to convert only two channels, in the devices list you will find only A/D Channel 0 and A/D Channel 1.

At this point, you will have to select the source variable (or the destination one) and the peripheral to which it is destinated.

In addition you can specify a constant value to be sent to the peripheral. The constant must be in the [0, 255] range, with the exception of the Triac and Timer Prescaler where you can specify a 16 bit value (i.e. in the range [0, 65535].

The peripherals that can interact with the Global variables are described below:

SEND BLOCK

Serial Port
Copies the value of the variable in the transmission register of the SCI.

Parallel Port:
Copies the value of the variable on the output register of the parallel port.

Triac Counter
Sets the value of the Triac Driver Counter with the value of the variable.

Timer Counter
Sets the value of the Timer counter with the value of the variable.

Triac Prescaler
Sets the value of the Triac Driver Counter Prescaler. Only a constant value between 0 and 65535 can be specified.

Timer Prescaler
Sets the value of the Timer Counter Prescaler. Only a constant value between 0 and 65535 can be specified.

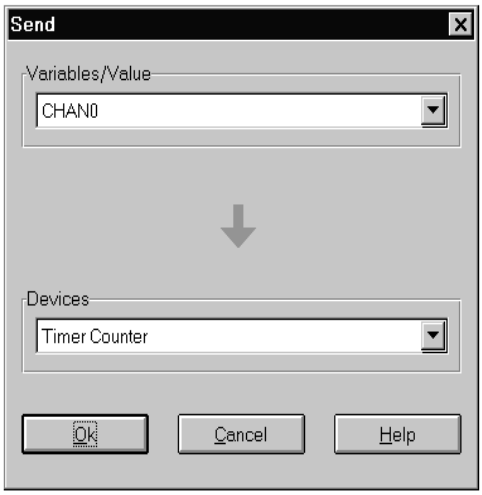
RECEIVE BLOCK

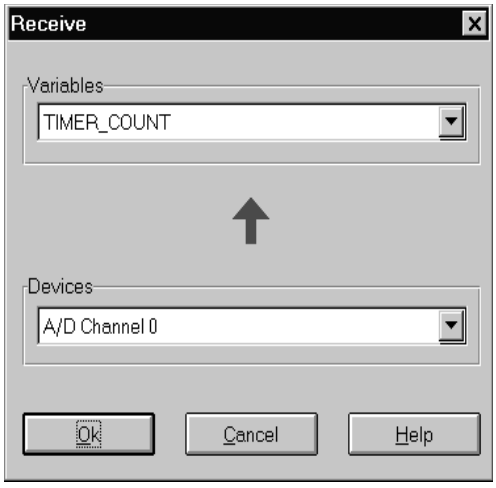
Parallel Port
Reads from the Parallel Port and assigns the value to the specified variable.

Timer Status:
Reads the Timer status register and assigns it to the specified variable. Only the first 2 bits are significative ones: bit 0 indicates if the status of the Timer is Start (1) or Stop (0) and bit 1 indicates if the status is Set (1) or Reset (0).

Serial Port
Read from the Receive register of the SCI

Timer Counter:
Reads the Timer Counter and assigns it to the specified variable.





SCI Status

Reads the status of the SCI. Each bit of this byte has a particular meaning:

bit 0 Sets if transmission ended, resets otherwise.

bit 1 Sets if transmission register is empty, reset otherwise.

bit 2 Ninth bit value when the data frame is nine bit.

bit 3 Not used.

bit 4 Sets in case of Overrun error.

bit 5 Sets if a data has been received.

bit 6 Sets in case of Frame error.

bit 7 Sets in case of Noise error

For further details about SCI functionalities see ST52x301 Data Sheet.

Fuzzy Output 0

Reads the first output calculated by the Fuzzy Unit and assigns to the variable.

Fuzzy Output 1

Reads the second output calculated by the Fuzzy Unit and assigns it to the variable.

A/D Channel 0

Reads the converted value of the A/D Channel 0 of the and assigns it to the variable.

A/D Channel 1

Reads the converted value of the A/D Channel 1 of the and assigns it to the variable.

A/D Channel 2

Reads the converted value of the A/D Channel 2 of the and assigns it to the variable.

A/D Channel 3

Reads the converted value of the A/D Channel 3 of the and assigns it to the variable.

When the settings have not been performed yet, in the two list boxes the writing NONE appears. Clicking OK you confirm the settings carried out.

Note *The change of the peripherals' settings initialization may invalidate the settings previously carried out in the Send or Receive Block. In this case, the settings of the block are reset to NONE. If you carry out a compilation without opening the Send or Receive Block, the corresponding code is not generated and the compilation is carried out all the same. Deleting or changing the name of a Global Variable involves the automatic erasing of the Send or Receive Blocks that involve it, leaving an open node on the flow-chart.*

Chapter 13

Restart Block

The Restart Block can be inserted either from the menu INSERT or by the appropriate toolbar icon.

The insertion of a Restart Block in the flow-chart, determines a Jump of the program to its start; this is equivalent to a link connected under the Start Block. It allows the iterated performing of the program set. Being this the only function of the Restart block, it has no editing environment associated. Then, a double-click on this block causes no effect.

It is not compulsory to insert a Restart Block since a link to any part of the program can be used to iterate the program.

Take note that the last block of the flow-chart has to be linked to another part of the program because the program has to be cyclic. On the contrary an error message appears during compilation.

Chapter 14

Compiler

Compiler button

After the definition of the main project's features, it is possible to compile the code to be loaded in the device's memories, by means of the programming functions and the Programming Board provided with FUZZYSTUDIO™ 3.0.

To launch the compilation of the project, select the item Compiler > Run from the TOOL menu or click on the appropriate toolbar button. The output window will open (or is put in foreground) showing the behaviour of the compilation and the eventual error messages and warnings.

The compilation is divided in three steps: script file generation in WCL language (.WCL), assembler file generation (.ASM), machine code generation (.BIN). Between the first and the second phase, the debugging code is generated to use the FUZZYSTUDIO™ 3.0 Debugger tool.

The generation of the assembler file and code can vary according to the choices taken in the compiler option window.

Compiler Options

The dialog box of the compilation options allow to choose the compilation modes. To open this dialog-box select the item Compiler > Options ... from the TOOLS menu.

- Disabling of the .BIN generation file containing the machine code: uncheck the check-box "Binary".
  Warning: disabling this option it will not be possible to program the device.
- Disabling of the debugging code file: deselect the check box "Debug".
  Warning: disabling this option it will not be possible to use the Debugger.
- In order to choose the generation mode of the assembler files; select one of the following:

1. **Over/underflow control:** allows to generate the code to detect, by using the instructions IsOverflow() and IsUnderflow(), when an arithmetic instruction generates a value out of range allowing to discriminate the overflow from the underflow. The generated code is the least optimized.

2. **With out of range control:** allows to generate the code to detect, by using the instruction IsOutOfRange, when an arithmetic instruction generates a value of the allowed range, without discriminating the overflow from the underflow. The generated code is more optimized than the previous one.
3. **Standard code:** generates a code that is more optimised than the previous ones, but it does not allow the use of the control instructions previously described. We recommend the use of this compiling option if you do not intend to carry out control on the results of the arithmetic instructions.

- Possibility to limit the number of error messages: selecting the check box "Limit messages" and specifying the maximum number of warning and error messages.

Compiler Messages

The following error messages can occur during WCL code generation:

initialisation required for "name"
The Fuzzy Variable "name" has not been initialised. Use Initialise command to initialise the Fuzzy Variable with a Global or Predefined Variable.

pending branch in block "name"
After the block called "name" there is no other block or link connected.

pending "Yes" branch in block "name"
The "Yes" link of the conditional block called "name" is not connected to any block or link.

pending "No" branch in block "name"
The "No" link of the conditional block called "name" is not connected to any block or link.

output link in loop in block "name"
The link in output from the block called "name" is connected to itself: connect this link to a block or to another link.

WARNING

default Store In initialisation used for "name"

The Fuzzy output variables called "name" has not been associated to any Global or Predefined Variable, so the current value is stored in default FUZZY0 and FUZZY1 Variables.

loop in block "name"

The block called "name" is connected to itself generating a loop.

Prescaler value not supported

The specified value for Triac Prescaler is not supported in ST52x301 device. The use of this value may cause undesired behaviour. It is suggested to specify a value higher than 1.

Receive block "name" with no data

The specified Receive block has no data due to missing user's specification or automatic deleting.

Send block "name" with no data

The specified Send block has no data due to missing user's specification or automatic deleting.

unreachable code in block "name"

The block called "name" is not connected to the program flow and it is not compiled.

FATAL ERRORS

The following messages can occur during Assembler code generation:

cannot access output file "name"

The file called "name" containing the data generated by the Compiler cannot be accessed. It may be caused by a disk full error or because the target directory is not accessible for writing operations or due to an internal error. Try rebooting the computer.

cannot copy from temporary file

The temporary file containing the data generated by the Compiler cannot be copied on the .asm file. It may be caused by a disk full error or because the target directory is not accessible for writing operations or due to an internal error. Try rebooting the computer.

cannot create temporary file

A temporary file generated by the Compiler during compilation cannot be created. It may be caused by a disk full error or because the target directory is not accessible for write operations or due to an internal error. Try rebooting the computer.

cannot open input file "name"

The WCL file called "name" containing the WCL generated by the Compiler cannot be open. The file may be corrupted or an internal error occurred. Try rebooting the computer.

cannot open output file "name"

The assembler file called "name" containing the Assembler code generated by the Compiler cannot be open. The file may be corrupted or an internal error occurred. Try rebooting the computer.

cannot read DBI file
The temporary file containing the data for the Debugger cannot be read to be copied in .dbi file. The file may be corrupted or an internal error occurred. Try rebooting the computer.

cannot write on temporary file
A temporary file generated by the Compiler during compilation cannot be written. It may be caused by a disk full error or because the target directory is not accessible for writing operations or due to an internal error. Try rebooting the computer.

cannot write DBI file
The Debugger Information file generated by the Compiler during compilation cannot be written. It may be caused by a disk full error or because the target directory is not accessible for write operations or due to an internal error. Try rebooting the computer.

input stream error
The WCL source file is corrupted or an internal error occurred. Try rebooting the compute.

no more available memory
There is no enough memory to perform the operation. Try again after closing some programs.

unexpected end of source
The WCL source file is corrupted or an internal error occurred. Try rebooting the computer.

wrong chip identifier "name"
The chip name specified as target by the "#define chip_name" instruction is incorrect.

ERRORS

argument is out of range
The argument or operand is out of the range allowed for that function or type.

byte value expected instead of "name"
The item "name" has been found instead of a byte value. Check the syntax of the expression.

cannot define the mbf
The Membership Function cannot be defined because there is no room for the related variable in Antecedent Memory.

cannot use library function
The specified function cannot be used in that context.

cannot use relational operator in condition
The conditional expression contains a relational operator that cannot be used.

conditional expression too long
The conditional operation contains more than one relational operation or operator.

constant "value" is out of range
a specified constant or operation between constant is out of allowed values for destinatio n variable.

function "name" not allowed
The function "name" cannot be used in that context or a syntax error occurred.

identifier '%s' too long
The specified identifier is more than 32 characters.

illegal operation
The specified operation is not valid in context.

impossible to add "name" variable
The number of Global Variables exceeds the allowed number according to the defined Fuzzy Variables.

integer value "value" is too high
An integer value higher than 32767 has been found. Check for syntax errors.

invalid function name "name"
The specified function "name" does not exist. Check for syntax errors.

invalid input register "name"
The input register "name" specified in LDRI instruction in Assembler block is not valid. Check for syntax error or if the predefined variables do not address an input register.

invalid library function "name" parameters
The parameters specified in function "name" are not correct: check for the number or position of parameters or for syntax errors.

invalid operator "name" in expression
The operator "name" in the expression is not allowed: check for semantic or syntax errors.

label "name" is redefined
The label "name" has been already defined before in the same or other Assembler blocks.

label "name" is undefined in procedure "name_blk"
The label "name" used in block "name_blk" has not been defined yet. Check for syntax error or define the label.

missing byte value
In an Assembler instruction a necessary byte value has not been specified.

missing closing comment sequence
The number of open parenthesis is higher than the closed ones.

missing input register
An input register name expected in LDRI instruction in Assembler block is not present. Specify a correct input register in instruction.

missing jump label
In Assembler block a jump instruction is not followed by a label.

missing operand in expression
The expression does not contain an expected operand.

missing token
The expression does not contain an expected operand or statement. This message may occur after previously detected error even if there is not an effective error in the specified place.

missing user defined variable
The expression does not contain an expected variable.

"name" function not allowed with current compiler options
The function "name" cannot be compiled with the currently used compilation modality. Change this in Compiler Option dialog box.

not allowed function "name" in context
The specified function "name" is not a valid name. Check for syntax error.

not readable predefined variable "name"
The write-only predefined variable "name" has been used in read operation.

not writable predefined variable "name"
The read-only predefined variable "name" has been used in write operation.

orphan operand in expression
An expected operator is missing before the operand.

predefined variable "name"' is read only
The read-only predefined variable "name" has been used in writing operation.

predefined variable "name" is write only
The write-only predefined variable "name" has been used in reading operation.

real value "value" not allowed
The specified real value is not in the range of the allowed values.

redefinition for name "name"
The local variable "name" has been redefined inside the same block.

signed value "value" is out of range
In a conditional expression, a constant higher than 127 or lower than -128 has been specified.

too few parameters in "name" function
One or more parameters are missing in the specified function "name".

too many operators in expression
The expression contain more than one operator. Spilt the expression in equivalents containing only one operator.

undefined variable "name"
The variable "name" has not been defined yet. Define the variable or check for syntax errors.

underflow in constant operation
An assignation expression with the difference of two constants have been specified and the result gives a value out of the allowed range of the destination variable.

unexpected binary operator in expression
A unary operator was expected in expression and a binary one was found.

unexpected token "name"
The item "name" is not allowed in that position or is not an allowed identifier. This message may occur after previously detected error even if there is not an effective error in the specified place.

unexpected unary operator in expression
A binary operator was expected in the expression and a unary one was found.

unsupported expression
The specified expression is not supported by the compiler. Check for syntax errors.

user defined variable expected instead of "name"
The item "name" was found instead of a user defined variable in Assembler block instruction.

unrecognised symbol "char"
The not allowed character "char" has been found in the program. Check for syntax errors.

INTERNAL ERRORS

The messages included in the following list normally should not occur. If one of this messages occurs it must be considered as an internal error: contact STMicroelectronics - Fuzzy logic B.U.

cannot define a procedure outside level

empty antecedent list in fuzzy rule

first parameter "name" is not unsigned byte value

first parameter "name" is not constant

invalid configuration register "name"

invalid define keyword "name"

invalid define keyword value

invalid device register "name"

invalid first parameter

invalid fuzzy output mapping "name"

invalid fuzzy output "name"

invalid interrupt number

invalid left vertex distance "value" , assuming 0

invalid mbfxxx index "value"

invalid mbfxxx string "name"

invalid procedure name "name"

invalid right vertex distance "value" , assuming 0

invalid second parameter

invalid type for fuzzy output variable "name"

invalid variable name "name"

invalid vertex "value" , assuming 0

missing chip definition

missing configuration register

missing define keyword

missing define keyword value

missing device register

missing fuzzy membership

missing fuzzy variable

missing interrupt number

missing left vertex distance, assuming 0

missing procedure name

missing procedure type

missing right vertex distance, assuming 0

missing variable name

missing vertex assuming 0

not existing fuzzy variable "name"

not existing membership function "name" in variable "name"

out of permitted range value

redefined procedure "name"

redefinition for variable

too many antecedents in fuzzy rule

undefined membership "name" for variable "name"

unexpected fuzzy var name for end statement

wrong procedure name "name" (expected "name")

WARNINGS

decimal part rounded to "value"

The decimal part specified is not supported by the program precision. The number is rounded to the value "value".

no fuzzy rule defined in fuzzy block "name"

The fuzzy block "name" does not contain any rule. Write rules or delete the block.

type mismatch assuming default type

A local signed byte variables has been defined in the Assembler block and it will be considered as byte.

Messages that can occur during machine code generation

The only message that may occurs during this compilation phase is the following:

out of chip code space

The generated program is longer than the available chip memory space. Try optimising the program.

Other messages that may occur have to be considered as internal errors: contact STMicroelectronics - Fuzzy Logic B.U.

Chapter 15

Debugger

Debugger button

The Debugger tool allows to test the developed program by means of the chip's simulation. The Debugger graphical environment allows to choose and visualize, through the plotting window, the signals to be observed in their time evolution. Then, you can test your program before implementing the application.

Using the Debugger tool the following functionalities are available:

- Simulation of the chip for a user-defined time interval.
- Visualization of the results in a graphical plotting window and in decimal, hexadecimal and binary numeric value.
- WCL tracing program with step-by-step debugging executing a WCL program line for each step.
- Visualization of the generated Assembler with the indication of current line.
- Visualization of the chip's internal registers in decimal, hexadecimal and binary format.
- Trace sequence of the program block in execution.
- Status report of the Debugger.

To execute the Debugger you need to select the item DEBUGGER from the menu TOOL. This operation will activate the Debugger by opening a window named WCL source from which you can open or recall all the windows related with the Debugger.

Note: *If the project has not been compiled after the last change, the Debugger cannot be run and a message inviting you to compile before opening the Debugging session is generated.*

General Description

This section provides an overview of the major elements of the debugger environment such as menus, toolbar and status bar. For additional information refer to index and on-line Help.

These are the main functionalities of the Debugger:

- Opening or switching to the other windows of the Debugger view.
- Setting of the time interval you want to simulate, indicating the time unit.
- Starting of the simulation either in a continuous or step-by-step way.
- Stop or reset the simulation.
- Opening of the WATCH EDIT dialog box to select the variables to observe.

The same operations can be performed selecting the relative item from the menu DEBUGGER, while the menu OPTION allows to change the fonts.

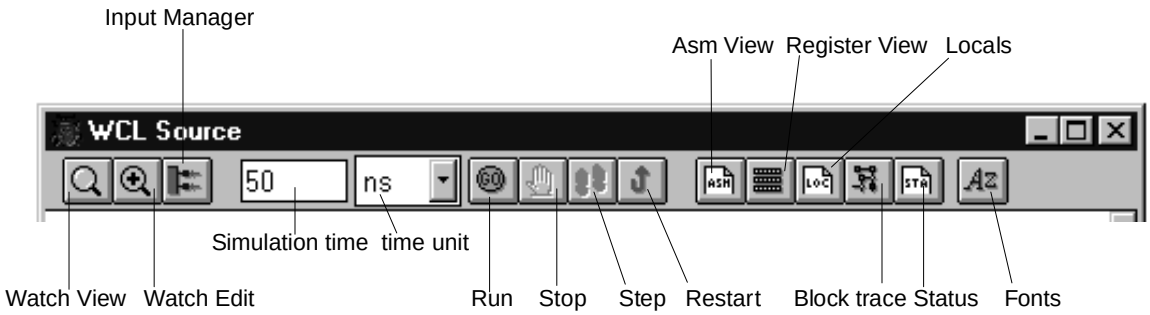
Debugger menus

The Debugger main menu is context-sensitive:when launching the Debugger, the item TOOL disappears and the items DEBUGGER and OPTIONS appear.

File	Contains commands to create, open, close, save and print a project; project info and list of recent files.
Edit	Provides standard editing commands.
View	Allows to toggle on/off Toolbar, Main Toolbar and Status bar.
Debugger	Allows to open the following windows: Watch View, Locals View, Block Trace, ASM View, Register View, Status, Input Editor and Watch Edit. Moreover it allows to run or reset the Debugger.
Options	Allows to open the font selection dialog-box.
Window	Contains commands related to the windows management.
Help	Contains help commands.

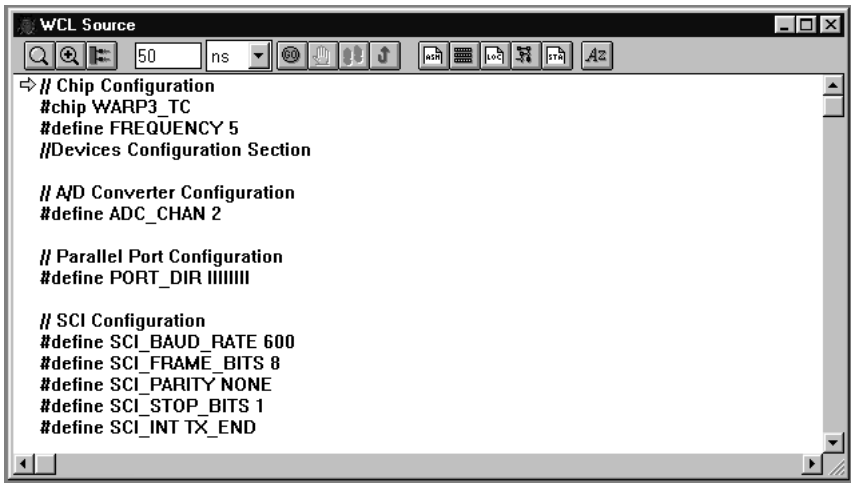
Debugger Toolbar

The window toolbar is common to all the windows related to the Debugger environment.



WCL Source Window

The WCL Source Window appears when executing the debugger. It shows the listing of the program in WCL format so as to allow the tracing of the program during the simulation. An arrow indicates the current line to be executed.

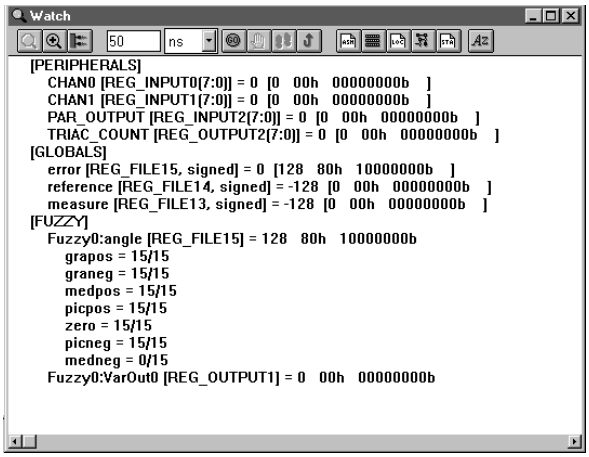


Clicking with the right mouse button over a WCL program line, a Breakpoint is inserted (this is preceded by a bullet at the beginning of the line); in this way, during the program's execution the program stops when encountering the breakpoint line.

To remove the breakpoint, click with the right mouse button over the program's line again.

Watch View Window

The Watch View Window displays in numeric format the values of the signals and of the selected variables. These are then plotted in the graphic window in their time behaviour. The selection is carried out by means of the dialog box WATCH EDIT that allows to add or remove the signals and the variables to be watched. A watch can be displayed as follows:



Global Variables:

NAME [REGISTER, TYPE] UDEC [DEC HEX BIN]

Fuzzy variables:

BLOCK: NAME [REGISTER] UDEC[DEC HEX BIN]

MBF UDEC[DEC HEX BIN]

...

MBF UDEC[DEC HEX BIN]

Signals

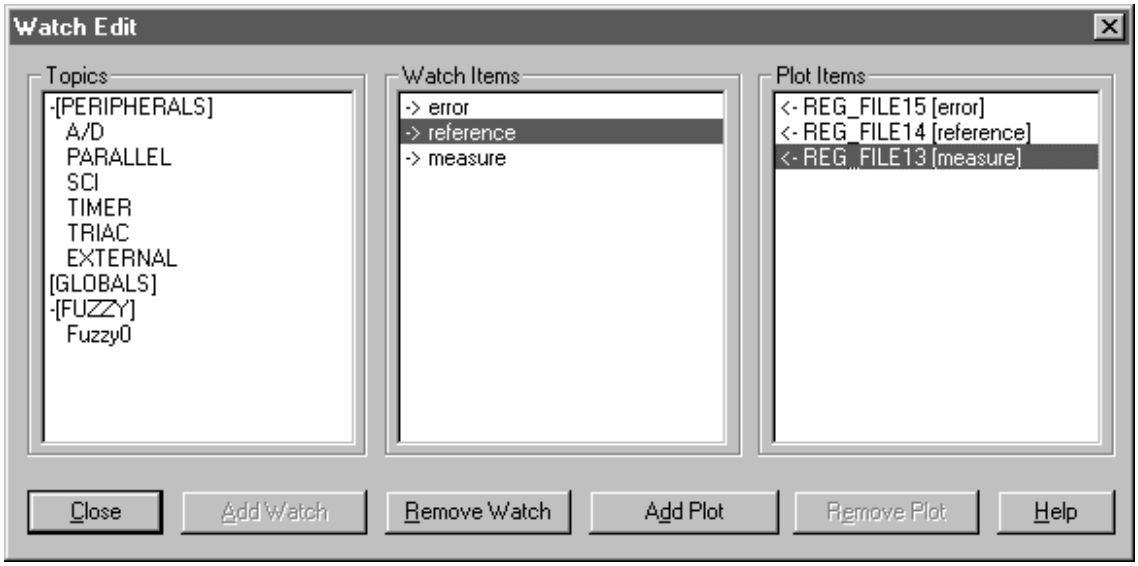
NAME[REGISTER(bit:bit)] UDEC [DEC HEX BIN]

where the following indicate:

NAME	Signal or variable name
REGISTER	Register address corresponding to the variable
TYPE	Variable type: byte or signed byte
UDEC	Current value in user's coordinates
DEC	Current value in decimal format
HEX	Current value in hexadecimal format
BIN	Current value in binary format
BLOCK	Fuzzy block to which the fuzzy variable belongs
MBF	Membership Function name
REGISTER(bit:bit)	Register's address with the bit part considered.

Watch Edit Dialog Box

Allows to add and remove signals and variables from the Watch and Plot windows. To open this dialog-box, select the item WATCH EDIT from the Debugger menu or click on the apposite toolbar button.



The Watch Edit dialog-box consists of three list boxes respectively named Topics, Watch Items and Plot items.

1. Topics

The Topics list box contains the names of the Topics to which variables and signals are associated:

Peripherals: Double clicking on Peripherals, the list of the peripherals available is displayed. Then, double clicking on one of these topics, the items lists display the signals and the variables that refer to that peripheral.

Globals A double click on this topic shows the list of the global variables on the Watch Items list and the list of the corresponding registers on the Plot Items list.

Fuzzy With a double click on the topic Fuzzy, the list of the defined fuzzy blocks is shown. Double clicking on one of these, the list of the variables defined in such a block appears in the Watch Items and Plot Items lists.

2. Watch Items

When one or more items from the Watch Items' list have been selected, click on the ADD WATCH button to add these items to the Watch View items' list or click REMOVE WATCH to delete them from the list. Single or multiple selection are available. Double-clicking on one item in the list you can add or remove it from the Watch View.

3. Plot items

After you have selected one or more items from the Plot Items's list, click on ADD PLOT to add the Plot Items to the list of signals to be plotted graphically, or click REMOVE PLOT to delete them from the list. Single or multiple selection are available. Double-clicking on one item in the list you can add or remove it from the Watch View.

In the Items lists an arrow next to the Item indicates if this has already been inserted (-->) or not inserted (<--) in the Watch View or in the Plot signals list (if not inserted, it can be selected and added to the list).

Note *In signed byte variables, the value of the signed variable is indicated in decimal format and between brackets is indicated the corresponding value of the register in decimal, hexadecimal and binary formats. The value of the register differs of 128 as compared to the variable, according to the convention for the representation of signed byte variables. In the graphical plot window only the signals can be found, then only the value corresponding to the associated register is indicated.*

Watch Edit signals

The peripherals items' names used in the Watch Edit list boxes, correspond to the signals and event of the chip. A detailed explanation of the meaning of this names is given in the following.

A/D

CHAN0	A/D channel 0 converted value (ADC_OUT_0 register)
CHAN1	A/D channel 1 converted value (ADC_OUT_1 register)
CHAN2	A/D channel 2 converted value (ADC_OUT_2 register)
CHAN3	A/D channel 3 converted value (ADC_OUT_3 register)
ADC_IRQ_SET	A/D converter interrupt mask bit (MSKAD bit REG_CONF14)
ADC_IRQ_PRIOR	A/D converter interrupt priority level (REG_CONF15)
ADC_SET	A/D converter Set/Reset status (ADTST REG_CONF2)

PARALLEL

PX	Parallel Port pin No. X (X= 0-8)
PAR_INPUT	Parallel Port register when read (INP_PORT register)
PAR_OUTPUT	Parallel Port register when written (PERIPH_REG_2)
PAR_PIN_SET	Parallel Port pin No. 8 bit (OUT bit REG_CONF1)

SCI

TxD	Serial Port Transmission pin
RxD	Serial Port Reception pin
SCI_INPUT	Serial Port receive register (SCDR_RX register)
SCI_OUTPUT	Serial Port transmitter register (SCRD_TX register)
SCI_IRQ_SET	Serial Port interrupt mask bit (MSKSCI bit REG_CONF14)
SCI-IRQ_PRIOR	Serial Port interrupt priority level (REG_CONF15)
SCI_TXSTART	Serial Transmission Start/Stop bit (TE bit REG_CONF3)
SCI_RXSTART	Serial Reception Start/Stop bit (RE bit REG_CONF3)
SCI_9BIT	Serial Transmission 9th bit data (T8 bit REG_CONF3)

TIMER

TIMEROUT	Timer output pin
TCLK	Timer external clock pin
TRES	Timer external Set/Reset pin
TCTRL	Timer external Start/Stop pin
TIMER_COUNT_INPUT	Timer counter current value (TMR_OUT register)
TIMER_STATUS	Timer Status register (TMR_ST register)
TIMER_COUNT_OUTPUT	Timer counter buffer (PERIPH_REG_0)
TIMER_IRQ_SET	Timer interrupt mask bit (MSKTM bit REG_CONF14)
TIMER_IRQ_PRIOR	Timer interrupt priority level (REG_CONF15)
TIMER_SET	Timer Set/Reset bit (TMRST bit REG_CONF6)
TIMER_START	Timer Start/Stop bit (TMST bit REG_CONF6)

TRIAC

TRIACOUT	Triac output pin
MAIN1	Triac zero crossing input pin
MAIN2	Triac zero crossing input pin / prescaler output pin
TRIAC_COUNT	Triac counter buffer (PERIPH_REG_1)
TRIAC_IRQ_SET	Ttriac interrupt mask bit (MSKTC bit REG_CONF14)
TRIAC_IRQ_PRIOR	Triac interrupt priority level (REG_CONF15)
TRIAC_SET	Triac Set/Reset bit (TCRST bit REG_CONF10)
TRIAC_START	Triac Start/Stop bit (TCST bit REG_CONF10)
TRIAC_TROUT_PIN	Triac output enable/tristate bit (TCTRS bit REG_CONF11

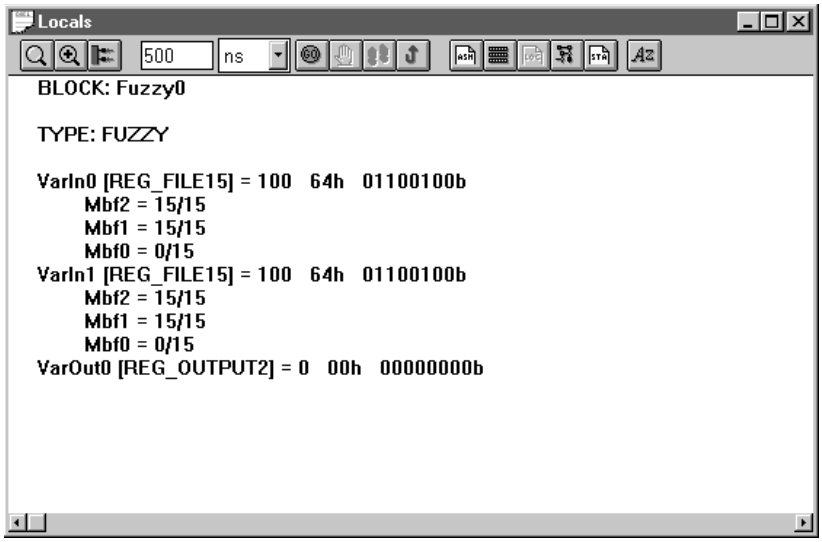
EXTERNAL

INT	External Interrupt pin
EXT_IRQ_SET	External Interrupt mask bit (MSKE bit REG_CONF14)
EXT_IRQ_TRIGGER	Falling/Rising edge selection bit (EXTI bit REG_CONF14)

Locals Window

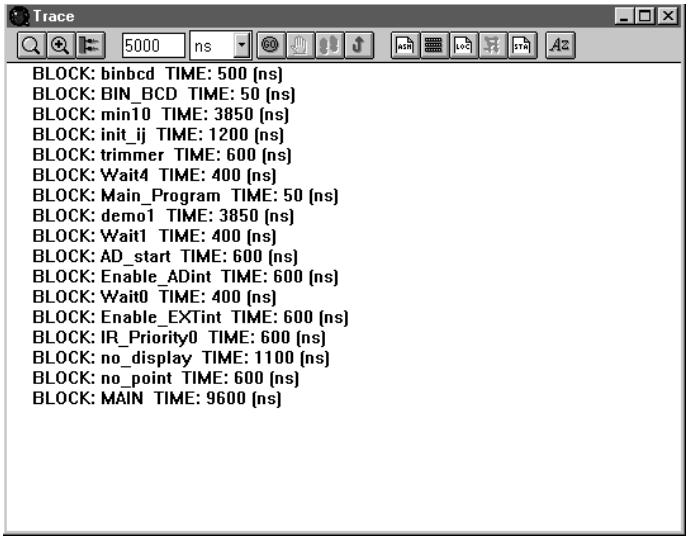
The Locals window is similar to the Watch View window. The difference is due to the fact that only local variables block or the fuzzy variables and the internal registers of the Fuzzy Computational unit (see Appendix C) of the current fuzzy block are listed. The Local Items are not plotted in the plot graphic window.

Note *The Local Items' values are displayed only when the block they belong to is in currently being simulated.*



Trace Window

In this window is provided the sequence of the blocks as these are simulated. The first in the list is the last to be executes. Together with the list of the blocks executed, the execution time of the same block is supplied once processed.



Registers Window

In this window, the list of the internal registers of the chip is given with the decimal, hexadecimal and binary value. The registers can be of different type:

REG_FILExx Register File registers generally associated to the variables; xx indicates the physical address of the register involved.

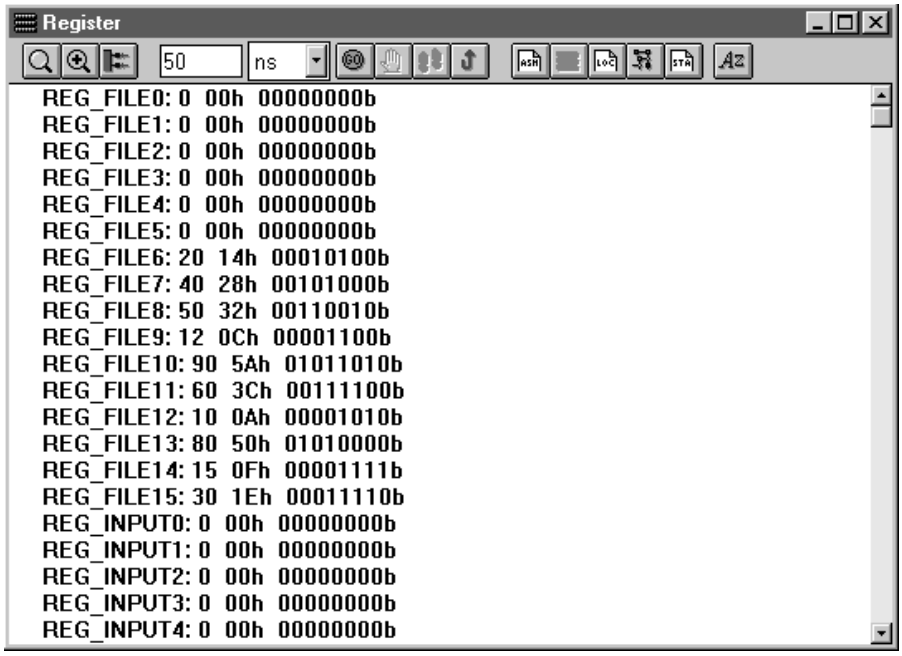
REG_INPUTxx Read-only registers of the peripherals, generally associated to the predefined variables; xx indicates the physical address of the register involved.

REG_OUTPUTxx Write-only registers of the peripherals, generally associated to the predefined variables; xx indicates the physical address of the register involved.

REG_CONFxx Chip's configuration registers generally associated to the predefined variables having the same name; xx indicates the physical address of the register involved.

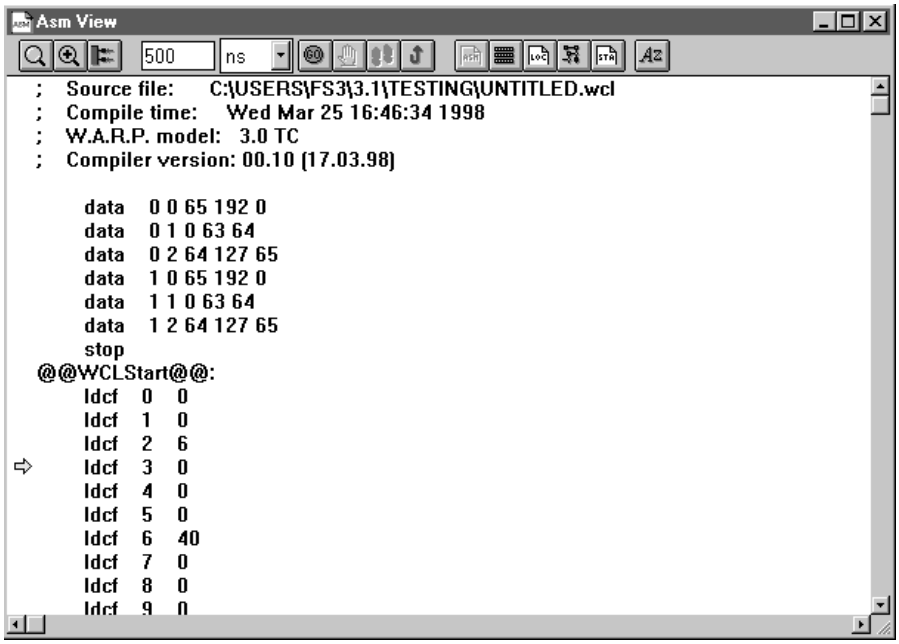
REG_STX Serial transmission register of the SCI, associated to the predefined variable SCI_BUFFER.

REG_SRX Serial reception register of the SCI, associated to the predefined variable SCI_BUFFER.



ASM View Window

This window displays the assembler list generated by the Compiler. An arrow indicates the assembler line in execution in the Debugger.



Status Window

It offers general information on the project and on the status of the current simulation. The information given are the following:

PROJECT Name of the project

DEVICE Target chip of the project

FREQ Frequency used

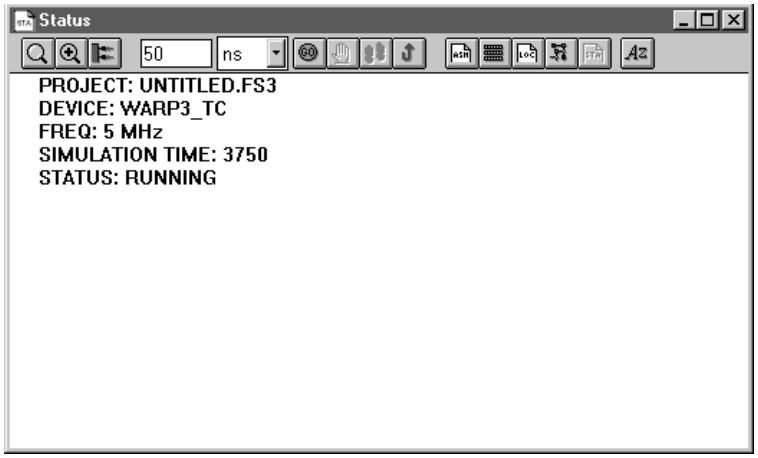
SIMULATION TIME Simulation time carried out

STATUS Status of the simulation

- RESET Reset Status of simulation
- RUNNING Simulation in progress
- DONE Simulation interval performed
- BREAK Simulation stopped by means of the apposite button
- INTERNAL ERROR Contact STMicroelectronics.

INTERRUPT STATUS Enabled and disabled interrupt and priority levels

- MASK Interrupt mask and external interrupt trigger edge
- PRIORITY Priority levels list

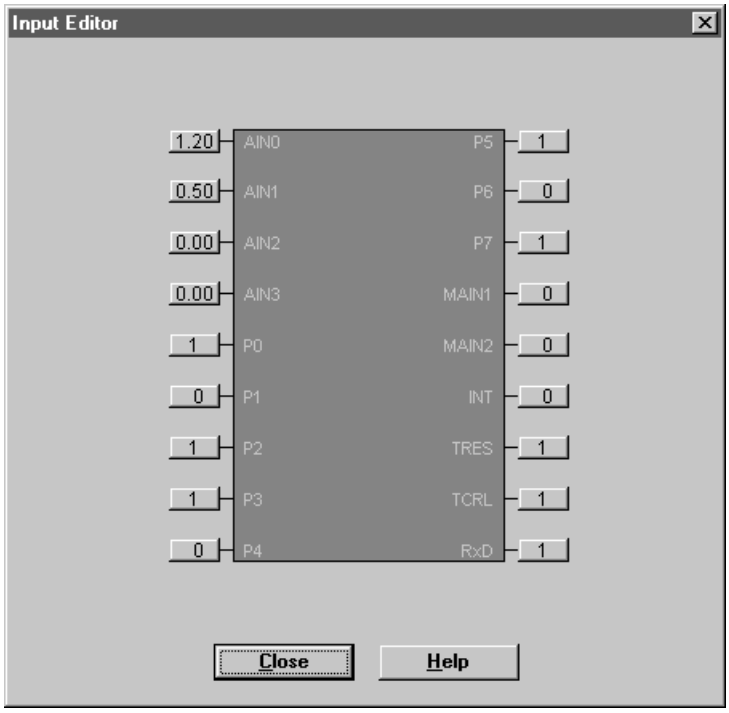


Input Editor Window

Before running the Debugger it is necessary to define the value of the inputs of the chip. This is carried out by means of the Input Editor window. This window is made up of a chip's schematic with its inputs. The inputs can be of two types: Signal or Analog input.

An input of signal type can take only two logical states 0 and 1. To change the input status of a Signal, click on the correspoinding pin.

An Analog input can take a value (in the range of real numbers) included between 0.0 and 2.5 Volt: it represents the voltage of the signal put in input to an A/D convertion channel. To change this value, click on the corresponding pin: a dialog-box opens allowing to set the value; this is displayed in the pin schematics.



How to Use Debugger

To use the debugger the following steps have to ben performed:

1 After a successful compilation of the project select DEBUGGER from TOOLS menu.

Warning: be sure you have checked the option Debug in the Compiler Options dialog-box otherwise the Debugger code file cannot be generated (refer to chapter 15 for more information).

2 Open the Watch View window by clicking on the apposite WCL source toolbar button or by choosing WATCH VIEW from DEBUGGER menu.

3 Open the Watch-Edit dialog box by means of the toolbar or from the DEBUGGER menu and select the variables and signals to be observed and/or plotted in a graphic way.

4 Open the windows containing additional information on the current simulation to observe (ASM View, register View, Locals View, Block Trace, Status View), if you are interested in them.

5 Set the value of the inputs by means of the Input Editor window.

6 Run the Step-by-Step simulation clicking on the STEP toolbar button: the current WCL instruction will be carried out; consequently the items modified are updated in the various Views and the selected signals are plotted in the Plot window. The arrow that indicates the current WCL instruction goes to the next one.

7 Repeat to execute the next lines.

If you want to perform a free running simulation for a certain period of time:

1 Insert in the simulation time in the apposite edit box, indicating the time unit (ns, µs, ms, s).

2 Select RUN from DEBUGGER menu or click on the apposite toolbar button.

3 The simulation can be stopped by inserting a breakpoint or clicking on the STOP button.

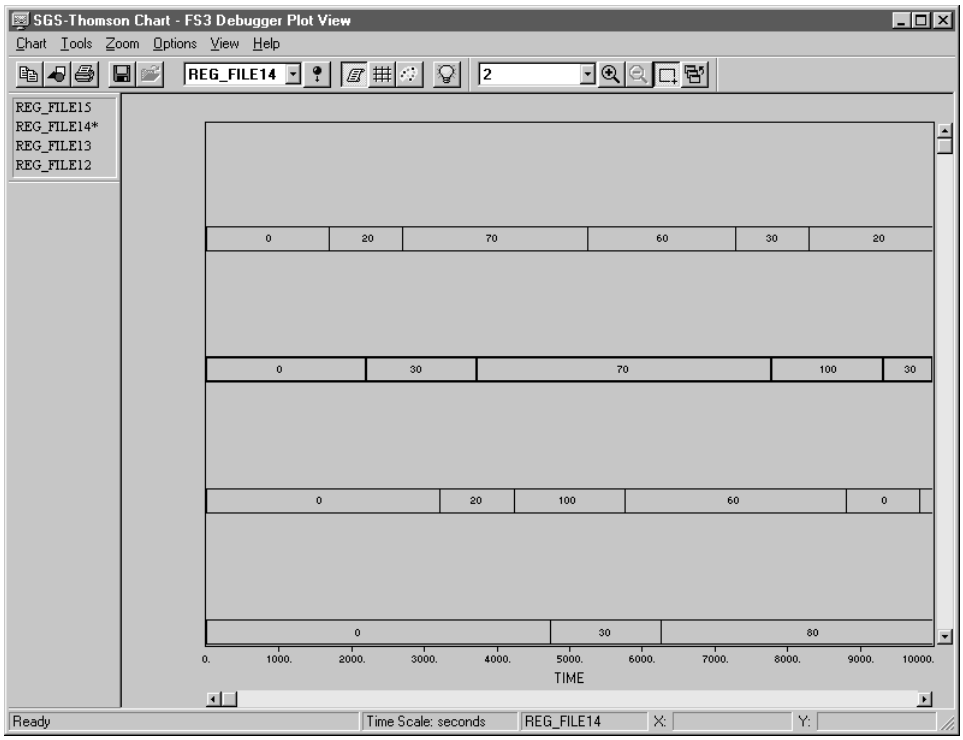
4 The data are updated in the views and the signals can be observed in the plot window.

Plot View Window

The simulation results are represented in their time evolution, in a graphic way in the Plot View window. The signals and buses selected in the Watch-Edit dialog box are traced here.

The signal is a function with two possible values that are the logic states 0 and 1 and is represented with a continuous broken line; the term bus indicates the content of a register or a set of signals that is inserted in a continuous stripe, broken in the points where the value changes.

During the execution of the instructions the plot window is updated with the values produced by the chip's simulator. At the end of the simulation of the time interval defined by the user or after a stop command of the same emulation, the signals can be carefully examined by means of the commands provided by the Plot window.



Plot View menus

Available menus are:

Chart Contains commands to copy the graphic in the clipboard as bitmap or metafile and change the settings of the printer and printing options.

Tools Select the function and toogle the command to move along the selected function.

Zoom These commands give you options for reducing or enlarging the chart display.

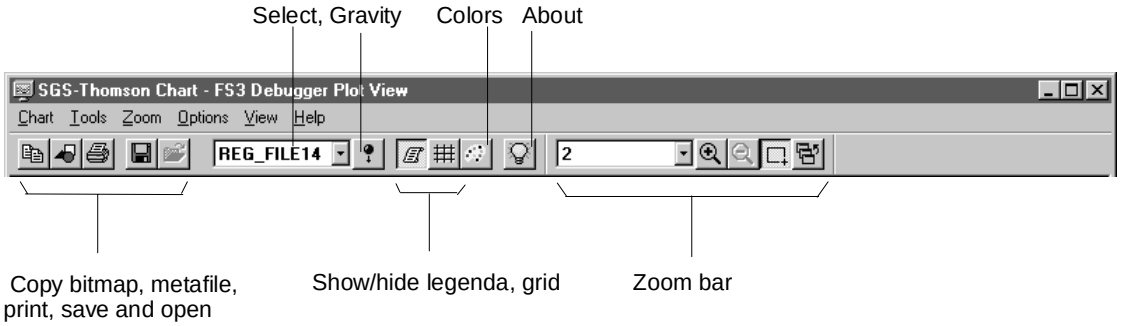
Options Select the time scale and the values' numeric format.

View These commands allow you to change the color settings of the background and foreground, show/hide grid, legenda, toolbar, zoom bar and status bar.

Help Contains help commands.

Plot View Window Toolbar

The Plot View Window includes a toolbar to help you perform the most frequently used commands quickly. To execute a task by means of a button, just click the related button on the Toolbar.



Print and Copy the graphic on the Clipboard

The graphic obtained by the simulation can be printed selecting the command PRINT (CTRL+P) from the menu CHART or use the apposite toolbar button. Before printing, verify the printer's settings are correct by selecting the command PRINT SETUP from the CHART menu.

To copy the graphic on the clipboard select the command COPY from the CHART menu. This command offers two options: COPY AS BITMAP (CTRL+C) or COPY AS METAFILE (CTRL+M) allowing to export on clipbard the graphic in the most suitable format.

Zoom

The Plot View window gives you option to change the screen display in order to better examine the results.

To horizontally enlarge a portion of the selected signal in order to observe a shorter time interval simulation, you can do one of the followings:

- Select the command ZOOM IN from the ZOOM menu: this command will expand the graphic starting from the left extreme currently displayed.
- Use the apposite toolbar button.
- Click with the left-mouse button on the point you want to enlarge; in this case the new interval will be centred in the point indicated. This option is available only if activated by the Zoom Enable toolbar button.

- Click on a specific area and drag a rectangle to define the area you want to zoom: the new extremes will coincide with the extremes of the window. This option is available only if activated by the Zoom Enable toolbar button.

To horizontally shrink the selected signal and observe a longer time interval simulation:

- Select the command ZOOM OUT from the ZOOM menu; this command will determine the shrinking of the graphic starting from the left extreme currently displayed.
- Use the apposite toolbar button.
- Click with the right-mouse button on the point you want to reduce; in this case the new interval will be centred in the point indicated. This option is available only if activated by the Zoom Enable toolbar button.

Note *Using the mouse buttons it might occur that one of the extremes goes outside the defined range. In this case, the extreme is calculated again and located at the extreme of your definition range. However, the centre will be shifted as regards to the point in which you clicked with the mouse.*

To perform the ZOOM OUT, you should have used the ZOOM IN command at least once.
To restore the standard size select RESTORE from ZOOM menu or click on the apposite toolbar button.

The zooming factor ca be set specifying an integer number by means of the command CUSTOM... from the ZOOM menu or from the apposite toolbar drop-down list-box.

Note *It is not possible to perform Zooming operations while the Debugger is running.*

Signals and Bus Selection

To highlight a signal or a bus you want to examine, it is possible to operate in the following modes:

- Select the signal/bus by choosing TOOLS > SELECT FUNCTION

- Select signal/bus from the drop-down list box available in the toolbar.

- Select the signal/bus from the legend, if activated.

To activate the legenda select the item LEGENDA from the VIEW menu or click on the apposite toolbar button.

In the same way it is possible to enable/disable the grid by choosing GRID from the menu VIEW.

Once a signal/bus is highlighted it is shown with thicker lines and can be examined with the function GRAVITY. This consists of a vertical bar moving along the graphic highlighting the time value (X) and the corresponding value of the signal/bus in that point (Y). these values are displayed in the Status Bar. To activate the function GRAVITY work do the following:

- Select the item GRAVITY from the TOOLS menu.

or

- Click on the apposite toolbar button.

Changing Time Base

The time base is by default expressed in nanoseconds. This time base can be changed selecting the desired base (from nanoseconds to seconds) from the menu OPTIONS> TIME.

The bus numeric values can have three types of representation: decimal, hexadecimal and binary. The representation of the numeric values can be modified by the menu OPTIONS > FORMAT.

Changing colors

It is possible to change the color of the background, of any signal or bus traced.

Select the command COLORS ... from the VIEW menu. A dialog-box appears allowing to select the color of Background, foreground and the commands to select each signal/bus and change the color.

Chapter 16

How to Program ST52x301

Once the editing of the project has been completed and after a successful compilation, it is possible to program the device by using the programming board supplied with FUZZYSTUDIO™ 3.0.

To program ST52x301:

1 Connect the programming board to the parallel port LPT1 or click on apposite toolbar menu.

2 Insert a blank ST52x301 device in the apposite socket.

3 Feed the board by means of a 17V DC power supply.
Note: If you use a different power source, it is suggested to use a power supply from 15 DCV up to 18 DCV.

4 Choose PROGRAMMER from TOOLS menu or click on the apposite toolbar button.

Programmer Main Features

The ST52x301 Basic Programmer has been designed to allow a fast programming of the EPROM or OTP version of the fuzzy microcontroller.

- Programs EPROM and OTP versions
- PC driven
- Fast parallel code transfer
- Leds' programming monitoring
- Memory blank-check

Its control software runs under Windows® environment. It can be started by clicking on the apposite toolbar button or menu from FUZZYSTUDIO™ 3.0 software and drives the programmer board through the parallel port LPT1.

After a memory blank-check, the binary program code is loaded into ST52x301 EPROM directly from FUZZYSTUDIO™ 3.0 main environment.

The programmer package is composed of:

- 1 Programming Board
- 1 Communication 25 wire Flat Cable
- 1 sample of ST52x301

Hardware Installation

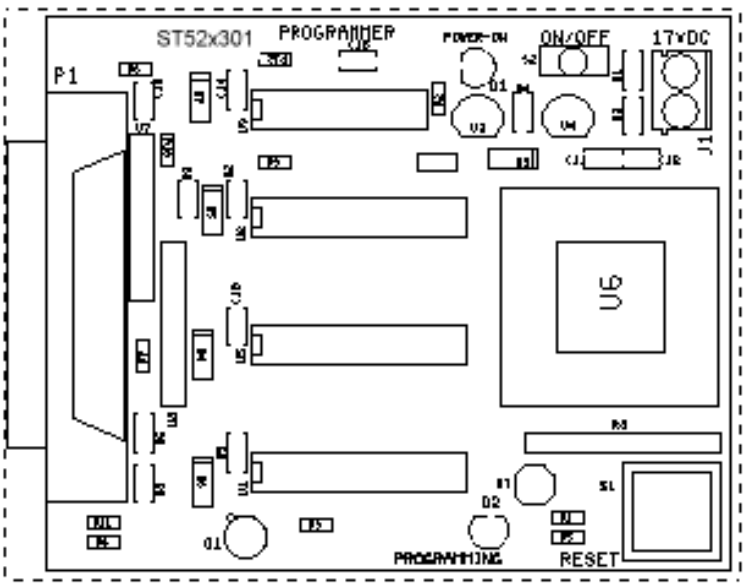
To install the ST52x301 Programmer, you have to:

- Turn off the PC
- Connect the 25 wires cable connector to the PC LPT1 parallel interface
- Turn on the PC
- Run FUZZYSTUDIO™ 3.0
- Insert the device before swtiching on the programmer
- Switch on the ST52x301 Programmer Board (15VDC to 18VDC).

Chip Insertion

When inserting the ST52x301 device into the PLCC socket, it is important to take care of the correct orientation in order to avoid a possible damage to both the board and the device. Pin 1 of chip must to be oriented towards the center of the board The red led (Power on Led) will have a weak light if the device is wrongly fitted into the socket.

Moreover, it is mandatory to switch off the board during the chip insertion and switch it on after the operation in order to have a correct power-on reset of the internal EPROM's chip.



Hardware Description

The Programmer board is a simple interface between the PC and ST52x301 memory banks. This architecture allows a direct control of TFL31 memory signals by means of a PC.

Because of data bus unidirectionality in several parallel ports, the in-board standard logic supplies an hardware comparison between the data word to be write and the written data word. This solution allows an immediate verifying of the programmed code.

The board is designed to work with a wide range of power suppliers (15Vdc to 30Vdc unstabilized) and to work with a low cost 25 wires flat cable (unshielded cable).

A red LED is used to monitor the power and the short circuits during a possible wrong insertion of device. A green LED is used to monitor the programming phase.

A button allows the EPROM's address counter reset. It is suggested to push the button before programming although a power-on reset network is present on reset pin.

Programming Phase

After the compilation phase the main software will generate a binary file with the same fuzzy project name. This binary code will be loaded into ST52x301 memory in the subsequently programming phase.

The programming operations are performed directly by selecting the PROGRAMMER item from the TOOLS menu or by clicking the apposite toolbar button.

The programming phase starts with a default blank-check of ST52x301 EPROM and continues with the data writing. During these phases, a green LED will light to indicate the memory access.

The downloading generates a .log file named download.log. This supplies the following information:

- The name of the binary file bin containing the program code to be loaded onto the device.
- The file contained in file .bin in hexadecimal format.

The result of the device programming indicating the code lenght contained in the file .bin, in hexadecimal and decimal (into brackets) format, and the code lenght effectively transmitted (if an error occurs, this lenght does not correspond with the previous one).

Note: It might occur that the green led is lighting when the PC is on; it will be turned off after FUZZY-STUDIO™ 3.0 has been run.

It is suggested to disconnect the board from the PC parallel port when not is use, becuase it may cause problems to the PC operating system when FUZZYSTUDIO™ 3.0 is not open.

Error Messages

The software can detect some problems related to the programming board. During the programming steps FUZZYSTUDIO™ 3.0 output window could display the following messages:

The Output window will appear and these are the messages it can show :

"Successfully Programming"

The device has been successfully programmed and is ready to be used in the application.

"Data not found; please compile before programming"

The current project has not been compiled yet, or the file obtained is not available anymore. In this case compile the project and restart the programmer.

"Invalid data; please compile before programming"

The file obtained after the compilation is corrupted. Recompile the project and restart the Compiler.

"Data Transmission failure"

An error occurred during the data transmission to the parallel port; check if the device has been correctly inserted in its socket or check if the board is correctly connected to the port and if power supply has been connected.

"Device not blank"

The device inserted is already programmed.

"Not Enough Memory"

The PC memory is not enough to perform the programming of the device. Close some applications and try again.

"Internal Error"

An internal error occurred. Contact STMicroelectronics - Fuzzy Logic B.U.

Appendix A - Fuzzy Logic Introduction. Human language and Indeterminacy

In 1950 Alan Turing proposed a way to test computers for intelligence. He argued that if we have a human (the *interrogator*) talking via keyboard and screen with another human and a computer, and the interrogator is not able to decide which one is the human and which one is the computer just from the analysis of the answers to his/her questions, then we have to admit that the computer (or program) is intelligent. So far, no computer program able to pass this test has been written.

It is very clear that the difficulties involved in designing such a machine (or software) are as complex as the human way of thinking. The interrogator can ask absurd questions such as: "*Yesterday I saw a donkey flying over my house, what do you think of that?*", therefore the program needs to have a huge database of facts concerning animals and things and deduction rules of common-sense reasoning. Moreover, one of the main problems is the communication language. Human natural languages are very imprecise and ambiguous. However they are understood and used by humans in their everyday life. Part of our intelligent behaviour certainly consists of this understanding common language capability. However, it is not easy (and it is indeed a very challenging task) to fully understand how humans accomplish this task and, as a by product, how to teach machines to do it. For instance, we make a heavy use of *adjectives*, that is to say words whose task is to classify objects. The objects classified by the adjectives belong to any universe of discourse which is the set of all possible objects to which the adjective may be applied. Such universes of discourse may depend upon the context. For instance, the adjective *tall* may be applied to humans or to buildings, etc. Computers are able to understand very well adjectives such as *even or odd,* but how can they deal with *big or small* when referred to numbers? Today's machines are based on classical logic. A *logic* comprises a formal language for making statements about (certain) objects and reasoning about properties of these objects. Classical logic has been for many years the only mathematical mean of reasoning and it has been extensively applied in many applications related to artificial intelligence and control. Classical logic is black and white (0 or 1). It relates to a binary world which is the same one on which our computer are (mostly) based on. Even or odd are binary adjectives. A number is either even or odd. No other possibilities are allowed. But when can we say that a number is *big* or *small* ? Likewise, when we say that a certain person is *married* or *single* we are making a statement which is 0-1: a person is either married or single, no other possibilities. However when we say that a person is *young* or *old* we are making an imprecise yet for human beings useful and well understood statement. If we wanted to define the adjective *young* by means of classical logic we would need to find a threshold value, say for instance 30 years, so that if a person is less than thirty then is young, more than thirty is old. However, we would have the unreasonable result that in a matter of seconds one person would change status from young to old.

Fuzzy Logic (and more in general Fuzzy Set Theory) provides a mathematical tool to deal with such a kind of uncertainty and imprecision.

A general overview

Despite the meaning of the word *fuzzy*, Fuzzy Logic is a precise and exact mathematical instrument which can be used for control systems with the advantage of simplifying the development of application of any complexity. The above claimed development simplification is determined by the possibility to express the system knowledge by means of linguistic expressions rather than mathematical equations, as needed by traditional techniques, which in many cases are a quite complicated way to express human experience.

The system development is also simplified because one can use Control Rules which are locally defined. By combining the linguistic and local approaches of the Rules, it is possible to evaluate in a very simple way the effect of a single rule and in turn the way of modifying the rule to improve the results.

To better clarify how to transfer human experience to a fuzzy expert system we will use a simple but complete example regarding the driving of a vehicle in proximity of a road crossing controlled by a light.

The information we can use is:

- Color of the light
- Distance from the crossing
- Vehicle speed

The goal is to control the vehicle speed by means of an action that can be:

- to brake
- to keep the speed
- to accelerate

The linguistic approach.

Fuzzy Logic is a formal instrument that partially closes the gap between human reasoning and computers world. This is possible because we can use reasoning rules which can be expressed in a linguistic way, that is to say the same way that a human being would express them to another human being, to teach him/her how to make a decision and act given certain facts. Both antecedent and consequent part of the rules do not express defined actions but respectively they are reference conditions and behaviours.

Notice that the consequent part of each rule contains only a qualitative expression of the action to be performed, while the quantitative expression of the final decision is determined by the occurrence of all the rules.

Going back to our example we could say:

IF *the light* **is** *red* **AND** *the speed* **is** *High*
THEN *the action* **is** *to brake;*
IF *the light* **is** *red* **AND** *the speed* **is** *Low* **AND** *the crossing* **is** *far away*
THEN *the action* **is** *to keep the speed*
IF *the light* **is** *yellow* **AND** *the speed* **is** *Medium* **AND** *the crossing* **is** *far away*
THEN *the action* **is** *to brake*
.....
IF *the light* **is** *green* **AND** *the speed* **is** *very low* **AND** *the crossing* **is** *very close*
THEN *the action* **is** *to accelerate*

By comparing **THe Observed Data**	light is green, speed is medium, the crossing is not far
with **The Reference Term**	Red, Yellow, Green; Very Low, Low, Medium, High; Very far away, far away, close, very close
we obtnain a degree of truth which determines how much the observed condition is really the **Hypothesis** of the rules and consequently how much the final decision must be similar to the **Reference Conclusion** expressed on the rules	**IF** *the light* **is** *green* **AND** the speed **is** *Medium* **AND** *the crossing* **is** *away* **THEN** *the action* **is** *to keep the speed*

The above action rules are a result of the experience gained in time and to them we refer every time we approach a road light.

Every time we have to make a decision we use rules specifying the correct behaviour under well known conditions, which are used as a reference to be compared with the observed data.

In our example, the observed data are the road light colour, the distance from the road crossing and the vehicle speed, which might be obtained from the speed meter rather than visually approximated with respect to the external world; the decision to make is related to the pressure to be exercised on the brakes or on the accelerator.

It is important to stress once again that the decision is determined by all the rules in a way which is proportional to their degree of truth.

Fuzzy Logic, Fuzzy sets and Membership Function

To allow the computer to make decisions according to the linguistic rules we must make possible for it to evaluate the quantity that we have called Similarity Degree of the observed data and the Reference Terms. This can be done by using Fuzzy Logic.

Fuzzy Logic is an extension of the binary or boolean logic on which is based the mathematical reasoning usually encountered in schools or universities. To briefly introduce fuzzy logic we will start from binary logic and we will sketch their differences.

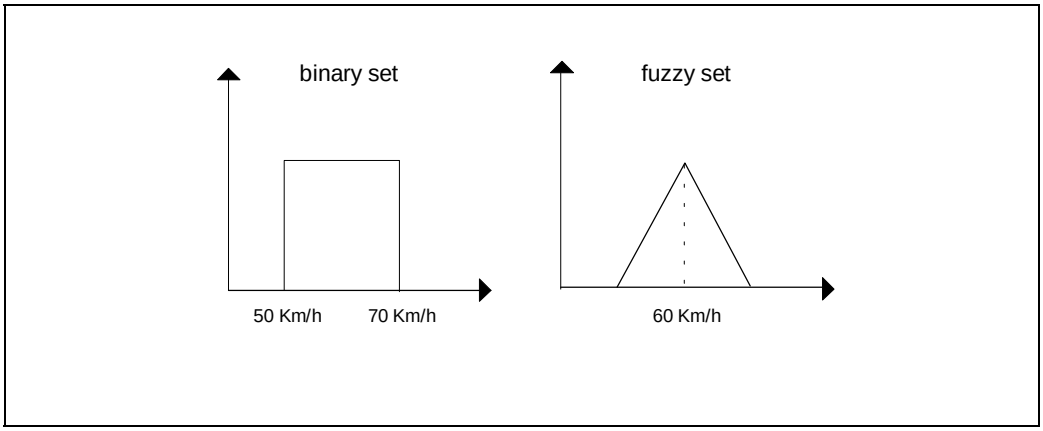
A boolean or binary set is a collection of objects all verifying the same condition (the characteristic property of the set). For instance, we could define the set of Medium Speeds as the collection of all the speed values between 50 e 70 Km/h. In this way we are implicitly saying that all values which are less than 50 Km/h or more than 70 Km/h do not belong to the set of Medium Speeds.

A Fuzzy set is associated with a Reference Term (for example Red, Yellow, Green for the light colour) and is characterized by a collection of objects which are *similar* to it. If, for instance, we decide that a medium speed is 60 Km/h, the closer a certain speed value is to 60 Km/h, the higher is its similarity degree to 60 Km/h. This similarity degree is what is denoted as Membership Degree to the fuzzy set. All the Membership Degree associated to a fuzzy set generate a shape called Membership Function.

The difference between the binary set and the fuzzy set *Medium Speed* is illustrated in fig. 1.

In this way, by using Boolean logic the made decision will be the same for all the values between 50 e 70 Km/h; moreover, we might obtain a completely different action in going from 50 Km/h to 49.999.. Km/h. On the other hand, if we use Fuzzy Logic we would have an answer which will proportionally depend on the membership degree .

Figure 1.



Fuzzy Reasoning

It is important to stress at this point that in order to make a decision it is necessary to have an experience in the field, which in fuzzy terms means that we need to have defined, for each input and output variable, the universe of discourse (i.e. the set of elements where it is defined), the fuzzy sets and the related membership functions, and to have identified the inference rules.

The Fuzzy Reasoning is composed by two computational steps, which permit to infer fuzzy value, for the output variables, starting from fuzzy value, for the input variables.

These steps are:

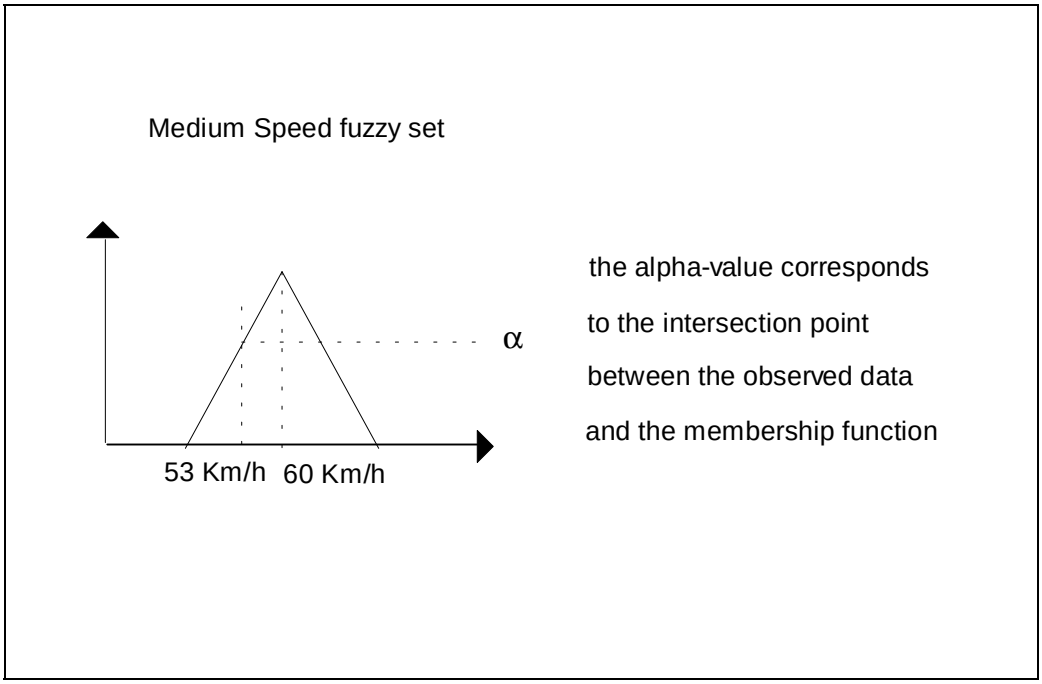
1 Alpha-values computation: given the observed values we compute the membership degree to the fuzzy sets by means of the membership functions. For instance, if we suppose that we have the following data:

- Light colour: green
- Speed:53 Km/h
- Distance from crossing:350 m

the operation of alpha-values computation provides us the degrees of memberships of 53 Km/h to the fuzzy sets *Very Low, Low, Medium, High* etc.; the degrees of membership of 350 m to the fuzzy sets very far away, far away, close, very close, etc. and combines this values in order to compute the strength of activation of each fuzzy rule.

In figure 2 we give a pictorial representation of the computation of the membership degree of 53 Km/h to the fuzzy set *Medium Speed*.

Figure 2.



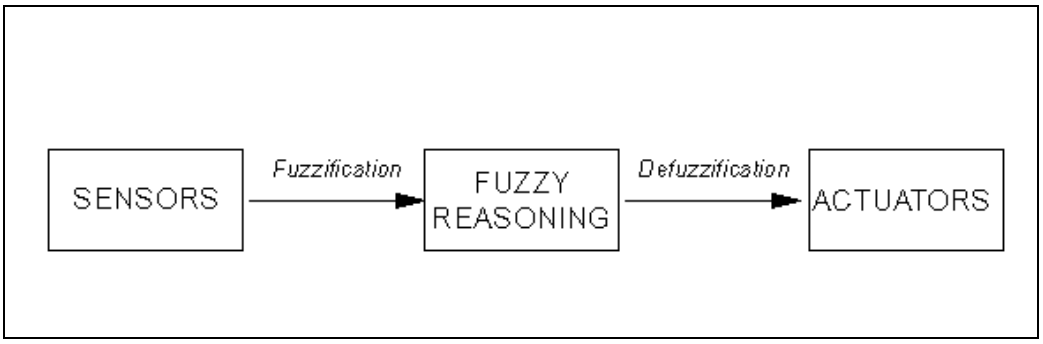
2 Fuzzy Inference: this operation formalizes the fuzzy reasoning by using the fuzzy rules and the alpha-values to deduce the fuzzy output.

In order to use fuzzy logic capability in real applications, involving crisp values which are usually supplied by sensors for the input and provided to the actuators for the output, it is necessary to accomplish the fuzzification and defuzzification operations to interface between the real system and the fuzzy inference engine.

Fuzzification: given an observed data we fuzzify it by means of the association with a corresponding function. Up to now, to speed up the calculus and to simplify the problems, the experts have been used to associate to each observed data a crisp value.

Defuzzification: this is the last operation of the reasoning process. In this case we obtain a precise answer and consequently a precise action to be taken.

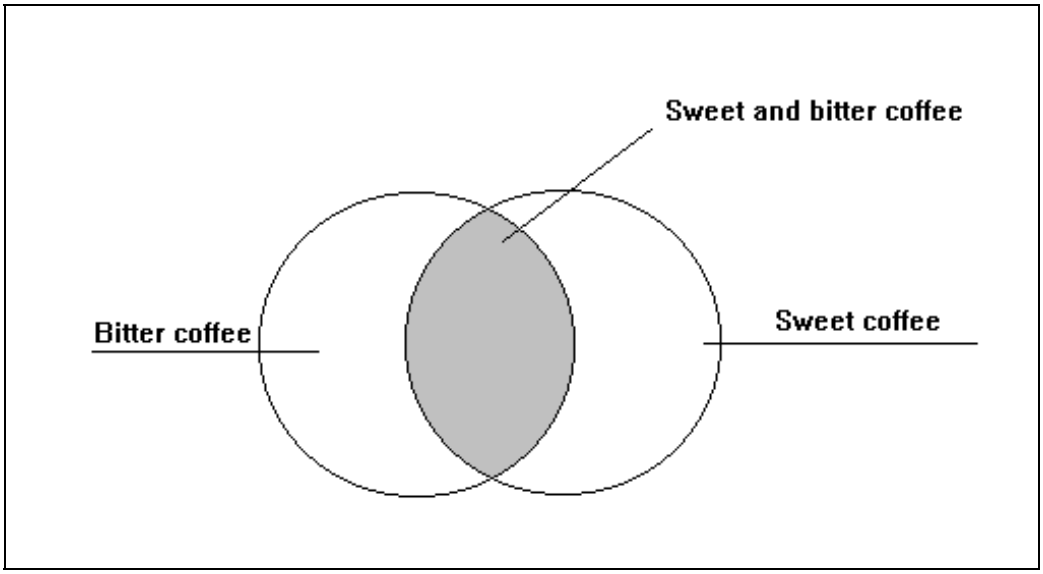
Figure 3.



The mathematical definition of Fuzzy Sets

To better clarify the concept of fuzziness that we are going to mathematically introduce let us briefly mention the *Sugar Paradox*. Suppose that we have a cup of coffee with no sugar. If we add a little grain of sugar certainly we still have a bitter cup of coffee. More in general, given a bitter cup of coffee by adding a little grain of sugar we are still left with a bitter cup of coffee. However, this process of adding little grains of sugar will eventually produce a sweet cup of coffee. From a classical logic point of view we have a paradox. What is happening in this case is that the passage from bitter to sweet is continuous and not abrupt as classical logic would want. If we consider then the sets of sweet cups of coffee and bitter cups of coffee we have the situation described in fig. 4.

Figure 4.



The border between sweet and bitter is not crisp but fuzzy. So for every element of the gray area we have a certain amount of indeterminacy concerning its sweetness. Is this indeterminacy of a statistical nature ? This is a natural question that scientists working in the field of fuzzy logic have already answered many times and in many ways. To answer this question we have to consider two cups of coffee. The first one has a high degree of sweetness. The sweetness concept is not of probabilistic nature. It has a linguistic value and its meaning varies from person to person. This means that the same cup of coffee can be sweet with respect to a person and less sweet with respect to another person. Regarding the second cup, we know that with high probability is sweet and with low probability is bitter. If one wants a sweet cup of coffee or at least with some sugar in it then the choice is the first cup. By choosing the second cup one may end up drinking a completely bitter coffee (not likely, but still possible). Choosing the first cup, one will be sure that the coffee is not bitter even though we do not know exactly the amount of sugar in it.

The Sugar Paradox introduces the notion of sets of a particular nature. These sets are such that some elements belong to them only up to a certain extent. A cup of coffee with just a little sugar cannot be considered bitter but can be considered sweet up to a certain extent only. This kind of membership cannot be formalized with classical (crisp) set membership functions defined, for a set **A** and an element $a \in \mathbf{U}$, as:

$$A(a) = \begin{cases} 1 & \text{if } a \in A \\ 0 & \text{if } a \notin A \end{cases}$$

$$A : U \rightarrow \{0,1\}.$$

which has value 1 (for membership) or 0 (for non membership).

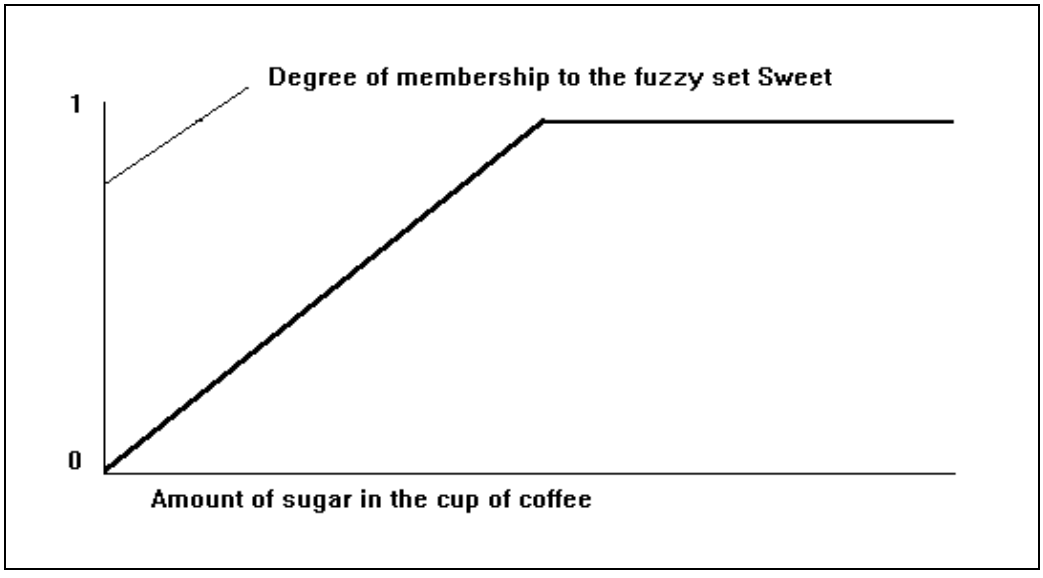
L.A. Zadeh introduced the notion of fuzzy sets in 1965. He defined them as a class of objects with a continuous membership function, valued into the whole interval [0,1]. This way a cup of coffee with just a little sugar will have (for instance) a degree of membership 0.1 to the set of sweet cups of coffee. A possible membership function for the fuzzy set Sweet is shown in fig. 5.

Thus, a fuzzy set **A** can be completely characterized by its membership function A defined as:

$$A : U \rightarrow [0,1]$$

where U (the universe of discourse) is the set of elements where **A** is defined or equivalently the set of parameters to be taken into account to define the degree of membership.

Figure 5.



Membership functions

At this point, a natural question is: where do membership functions come from? The answer is straightforward. They come from people experience and are related to people knowledge and understanding of the problem to be modeled. For instance, a membership function for *Hot Temperature* when referred to a room temperature will be different from the membership function referred to an oven temperature. Moreover, it may (and generally will) change from person to person.

Fuzzy set operators

To build our mathematical formalism of fuzzy sets we need to define the operators that allow us to combine fuzzy sets and obtain new ones, i.e. we need to extend to the fuzzy case the definition of operators such as *set complement*, *union*, *intersection*, and predicates such as *set containment* to the fuzzy sets.

To do it we need to introduce a new operator called "Triangular Norm" and commonly known as *t_norm*.

Stated I = [0, 1], the *t_norm* T function is defined as $T: I \rightarrow I$, and satisfies the following properties:

1 $T(x,1) = x \; \forall \; x \in I$

2 $T(x,y) \leq T(u,v)$ if $x \leq u$ and $y \leq v$

3 $T(x,y) = T(y,x) \; \forall \; x, y \in I$

4 $T(T(x,y),z) = T(x,T(y,z)) \forall \; x, y, z \in I$

where

$\forall$ means: for every value

$\in$ means: belong to a set

$\rightarrow$ means: correspondence between function domain and its support

$\leq$ means: less or equal.

The properties mentioned above are respectively:

1 neutral element existence with respect to T

2 monotony property of T

3 commutative property of T

4 associative property of T

Starting from T it is possible to define a function $S: I \rightarrow I$ as:

$S(x,y) = 1- T(1-x, 1-y)$

known as *t_conorm*, maintaining the associative, commutative and monotony properties and satisfying the condition:

a $S(x,0) = x,\ S(x,1) = 1\ \forall\ x \in I$

b $T(x,y) = 1- S(1-x,1-y)\ \forall\ x, y \in I$

These operators are the basis for the fuzzy set operators definition. In fact, considering two fuzzy sets **A** and **B** the union and intersection operators are defined in terms of T and S as follows:

$(\mathbf{A} \cap \mathbf{B})(x) = T(A(x),B(x))\ \forall\ x \in X$

$(\mathbf{A} \cup \mathbf{B})(x) = S(A(x),B(x))\ \forall\ x \in X$

The following step is the identification of T and S with some algebraic operators. In literature they are used to define:

$T(x,y) = \min(x,y)\ \forall\ x \in I$

$S(x,y) = \max(x,y)\ \forall\ x \in I$

since these two operators satisfy the required properties. It is important to stress that these are the most commonly used association but they are not the only one. Now we are able to formulate the fuzzy set operators. By doing this we will find out that certain laws of the Aristotelian 0-1 logic do not hold any longer.

Set Complement

Let us start with the set complement. The way this operator is classically defined is the following. Given a set **A** subset of a universe U, the complement of **A** is the set whose elements are all and only the elements of U which are not in **A**, as shown by the (Venn) diagram in fig. 6

Thus, denoted by **B** the complement of **A**, for every a in U we have

$$B(a)=1-A(a)$$

These means that, if a is in **A** then its membership degree is 1 (A(a)=1), which implies that B(a)=0, so a is not in **B**.

Conversely if a is in **B** then B(a)=1 and in turn A(a)=0 and so a is not in **A**. Notice that in particular for the classical logic holds:

- for every a in the universe of discourse we have that either A(a)=1 or B(a)=1. Equivalently, we can say that the maximum between A(a) and B(a) is equal to 1. This property is known as the *Law of the excluded middle*: every element of the universe of discourse is either in **A** or in its complement. No other possibilities are allowed. In terms of set operators:

$$\mathbf{A} \cup \mathbf{B} = U$$

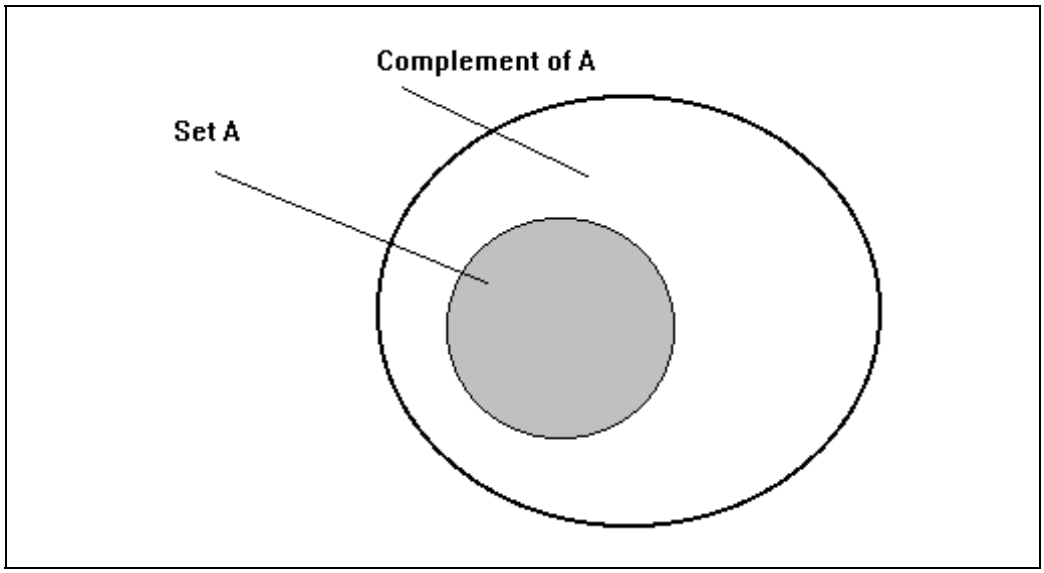
- for every a in the universe of discourse we have that either A(a)=0 or B(a)=0. Equivalently, we can say that the minimum between A(a) and B(a) is equal to 0. This property is known as the *Law of non contradiction* every element of the universe of discourse cannot be both in **A** and in its complement at the same time. In terms of set operators:

$$\mathbf{A} \cap \mathbf{B} = \varnothing$$

What happens when **A** is **fuzzy**, that is to say when the border with its complement is not clearly defined as shown in the fig. 7 ?

In this case we have to define the membership degree to the complement of **A** of the elements in the border of **A**. For consistency with the crisp case we define:

Figure 6.



B(a)=1-A(a)

We notice however that both the *Law of the excluded middle* and the *Law of non contradiction* do not hold any longer. Indeed, if 0<A(a)<1 then 0<B(a)<1, the element a does not belong exactly either to **A** or to its complement but it belongs to any degree to both of them. To clarify the above consider the following examples:

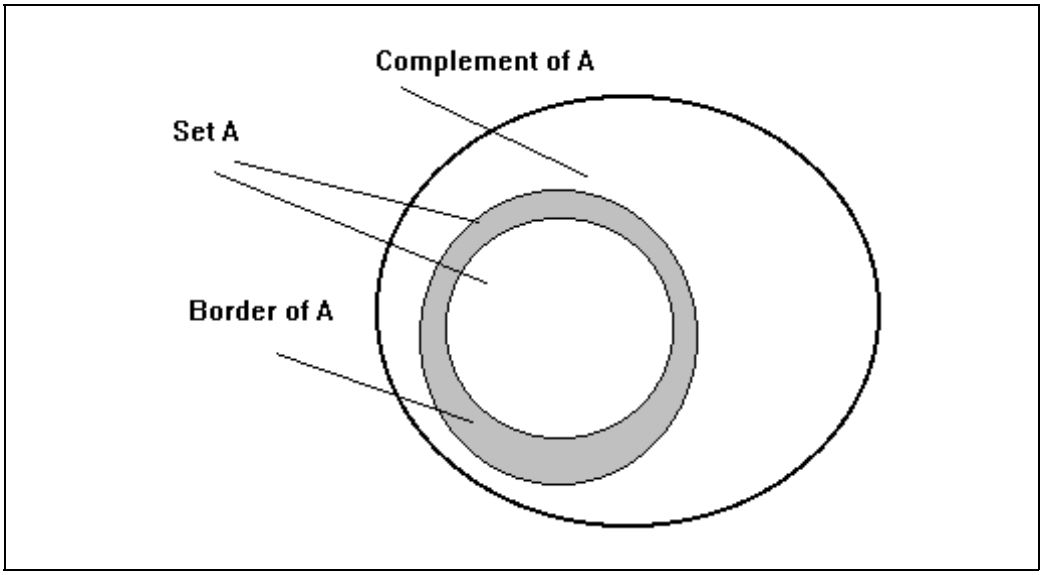
- A cup of coffee with just a spoon of sugar certainly cannot be considered *sweet* ; however it cannot be considered *not-sweet* either. It has a certain degree of sweetness and a certain degree of not-sweetness.
- A man who is 175 cm tall certainly cannot be considered *tall;* on the other hand it cannot be considered *not-tall* either. Again, he will have a degree of tallness and a degree of not-tallness.

Summing up we can say that a fuzzy set does not divide the universe of discourse into two parts: elements and not elements. Instead there is a third part which is characterized by all those elements which cannot be classified exactly either way.

Set Union

The union of two sets is the set whose elements belong to any of the two sets, as shown in fig. 8.

Figure 7.



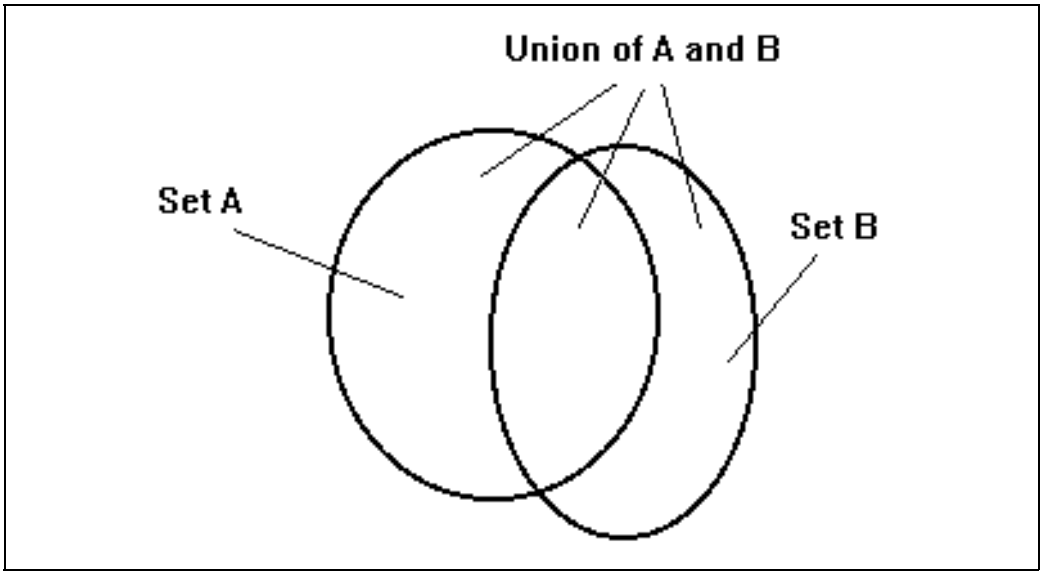
The above definition extends to the case of fuzzy sets by using the maximum rule as stated in the previous paragraph while speaking of *t_norm* and *t_conorm* operator. In details, the degree to which an element a belongs to the union of A and B is given by the maximum of the degree of memberships in **A** and in **B**:

$$A \cup B(a) = \max(A(a), B(a))$$

Set Intersection

The intersection of two sets is the set whose elements belong to both sets, as shown in fig. 9.

Figure 8.

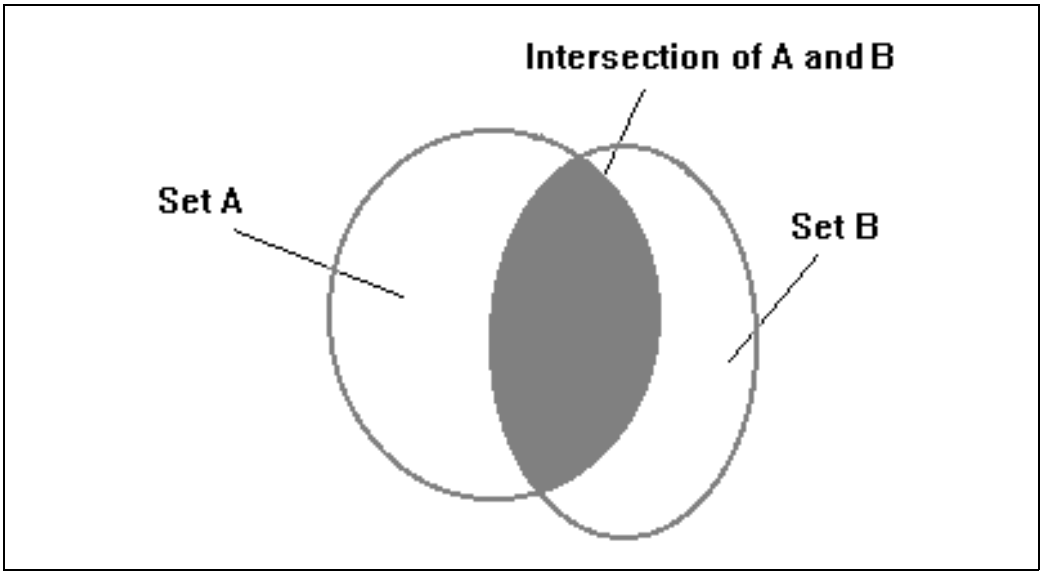


The above definition extends to the case of fuzzy sets by using the minimum rule. In details, the degree to which an element a belongs to the intersection of **A** and **B** is given by the minimum of the degree of memberships in **A** and in **B**:

$$A \cap B(a) = \min(A(a), B(a))$$

This way we have built a mathematical tool which allows us to deal with fuzzy sets in a formal and as we will see useful way.

Figure 9.



The mathematical formalism of Fuzzy Logic

Fuzzy Logic derives from Fuzzy Set Theory. Fuzzy Logic is concerned with statements of type *This coffee is sweet, My brother is tall, The temperature in the room is high.* These statements are characterized by the presence of concepts (such as height, temperature etc.) called *linguistic variables* which are defined over a set called universe of discourse and which are given *linguistic values*. What is a linguistic value ?

Let us consider the example statement: *The temperature in the room is high.* We have no information on the exact value of the temperature but we have instead a good information to decide on the possibility that the temperature has a certain value. For instance we can say that

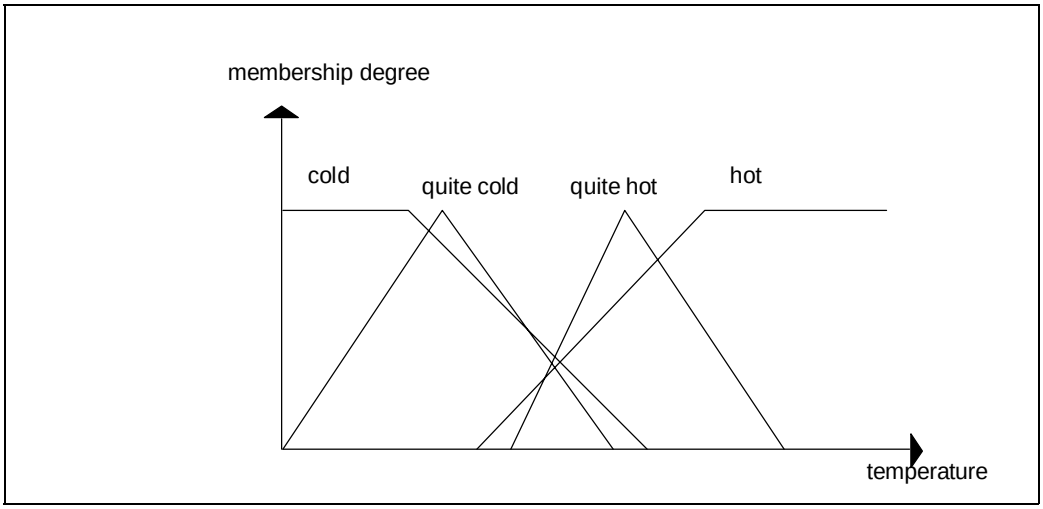
1 it is clearly impossible that the temperature is 10 degrees or less

2 it is very possible that the temperature is 30 degrees or higher.

A linguistic value (applied to a concept) so generates a set of possibilities on the exact value of the linguistic concept. This set of possibilities (or possibility distribution) is the logical counterpart of a fuzzy set: in our examples the fuzzy set of *High Room Temperature.* For any given temperature **t** the higher is the degree of membership to the fuzzy set *High Room Temperature,* the higher the possibility that **t** is the exact temperature in the room. This kind of statements are indicated as *fuzzy predicates*. They are usually denoted by "*x is A*" where x is an element of the universe of discourse and A is a fuzzy term. The possibility value that "x is A" is also the degree to which the proposition "x is A" is true. The set of linguistic values that can be given to a linguistic variables is called *term set*. Term sets are built starting from pairs of antonyms and applying to them logical and linguistic modifications. For instance the term set of the linguistic variable *Temperature* can be described as follows

Linguistic Variable	Temperature
Antonym pair	Hot, Cold
Modifiers	Not, Very, Quite, etc.

Figure 10.



To each of this linguistic values corresponds a specific possibility distribution which is obtained from the basic ones via logical aggregations.

The following picture illustrated a possible term set for the variable temperature.

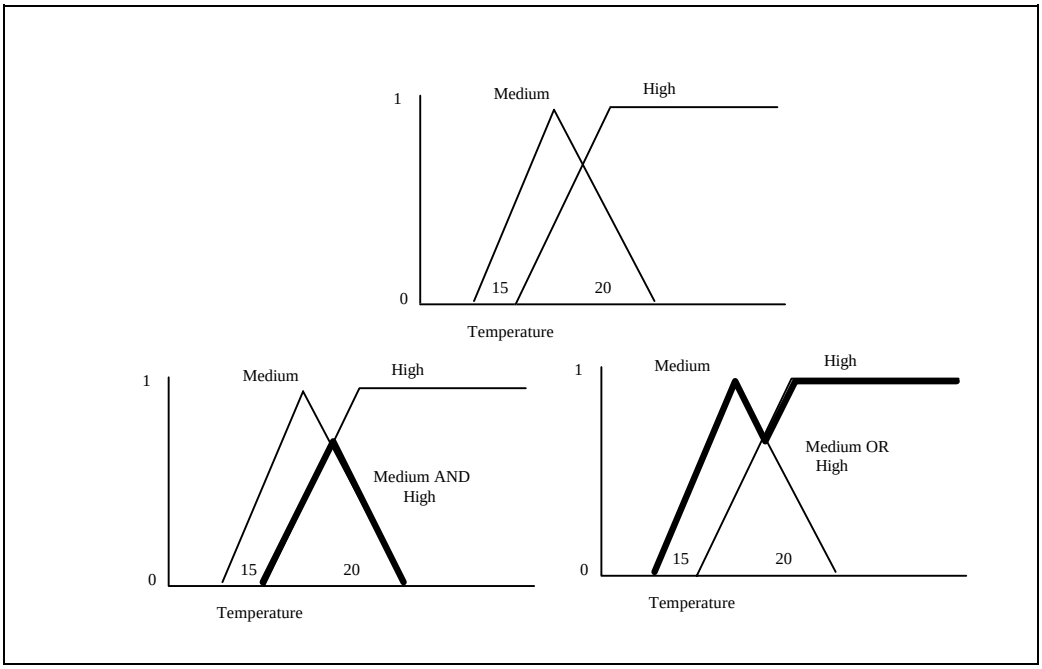
The logical connectives: AND, OR, NOT used to aggregate fuzzy sets correspond to the set operators. It is quite simple to understand how this connectives are defined if one has a clear understanding of the set operations.

- NOT: the logical negation corresponds to the set complement operator. So, given a fuzzy predicate *x is A* with degree of truth *t*, the degree of truth of *NOT (x is A)* will be *1 - t*.
- AND: the logical conjunction corresponds to the set intersection operator. Therefore, given two fuzzy predicates "x is A" and "x is B" the truth value of "x is A AND B" will be obtained by taking the minimum of the two input truth values.
- OR: the logical disjunction corresponds to the set union operator. As a consequence, given two fuzzy predicates "x is A" and "x is B" the truth value of "x is A OR B" will be obtained by taking the maximum of the two input truth values.

Fig. 11 gives a pictorial representation of the two logical connectives in the case we have the fuzzy predicates "x is High" and "x is Medium" where x ranges over the universe of discourse of Temperature.

As we mentioned above, fuzzy statements are evaluated by means of linguistic values. Therefore, along with the above mathematical operators, we can apply to them linguistic transformations (called *hedges*). For instance, given "*x is Young*" we can obtain "*x is Very Young*" or "*x is Quite Young*" etc.

Figure 11.



Fuzzy Reasoning

The main reason for the world-wide popularity gained by Fuzzy Logic is its capability to formalize patterns of human reasoning in a very simple, efficient and useful way. The continuously growing number of applications in fields such as Control Theory, Expert Systems, Robotics, Image recognition, Databases, etc. is an outstanding proof of it. The key idea comes from a simple consideration on classical logic. The fundamental inference rule, that is a rule that allows to obtain new true propositions from given ones, in classical logic is MODUS PONENS:

Premise 1:IF x is A THEN Premise 2:IF x is A	y is B
Conclusion:	yis B

The meaning of Modus Ponens is clear: if we have that *x is A* is true and if it is also true that the fact that *x is A* implies that *y is B* then we can conclude that *y is B* is true. Thus, from the true premises that "*Humans are mortal"* and "*John is human*" we can deduce that "*John is mortal*". Fuzzy Logic gives us a way to deduce useful conclusions either when the premises are not absolutely true or when the antecedent of premise 1 is similar but not equal to premise 2 or when premise 2 is obtained as a modification from the antecedent of premise 1.

Premise 1 above is in the form of a production rule of the kind usually applied in the field of expert systems. The production rule has the meaning: *if the antecedent (x is A) is true then apply the action (y is B).* Production rules of this kind are applied in fuzzy system control.

Premise 1:IF x is A THEN Premise 2:IF x is A’	y is B
Conclusion:	y is B’

Fuzzy Computation

Fuzzy Models are used whenever they can competitively provide better information about any physical process or any system. Fuzzy models are simple and strongly related to the human knowledge. A fuzzy model is given as a collection of (fuzzy) production rules: Fuzzy IF-THEN Rules. The collection of fuzzy IF-THEN rules and the related membership functions represent the knowledge of the system.

In the example below we have defined the following fuzzy sets for the input X_i variables and the Y output variable:

X_1: High, Medium and Low

X_2: High, Medium and Low

Y : A, B, C

and we have the following set of fuzzy IF-THEN rules:

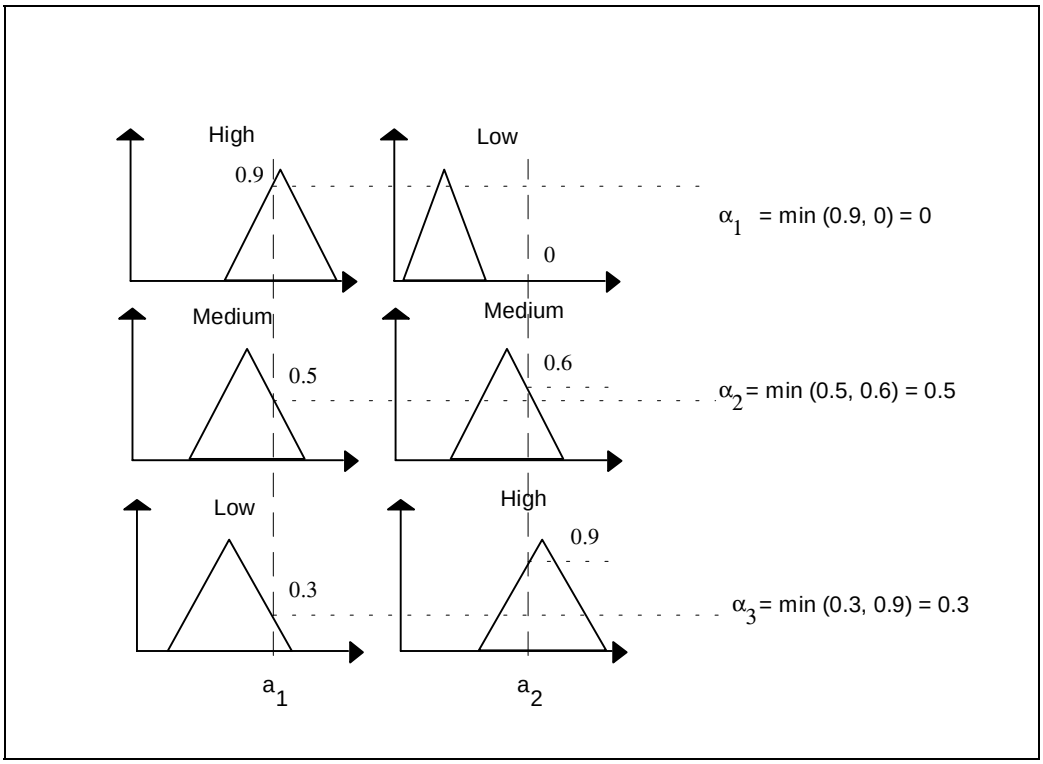
rule 1:If X1 is High and X2 is Low THEN Y is A

rule 2:If X_1 is Medium and X2 is Medium THEN Y is B

rule 3:If X_1 is Low and X2 is High THEN Y is C

As you can see above the correspondence between input values (condition) and output value (action) is expressed in terms of relation on input and output fuzzy sets.

Figure 12.



The rules are used as follows.

Step 1:

fuzzification phase. The input are coded associating to each of them the corresponding crisp value.

Step 2:

alpha-values computation phase. The coded input are compared with the antecedent fuzzy sets in order to evaluate their membership degree to the linguistic values. In our example above, given values a_1 and a_2 for the antecedents, we obtain the membership values:

α_1^1 =High(a_1)α_2^1 =Low(a_2)

α_1^2 =Medium(a_1)α_2^2 =Medium(a_2)

α_1^3 =Low(a_1) α_2^3 =High(a_2)

which are aggregated using the minimum rule in order to compute the strength to which the rules apply:

$\alpha_i = \min(\alpha_1^i, \alpha_2^i) i = 1..3$

The figure 12 gives a pictorial representation of the alpha-values computation.

Step 3:

inference phase. Using the alpha-values obtained from antecedent parts the membership functions of the consequent are modified. The most classical of the inference methods are the *max-min method* and the *max-dot method.*

Using the *max-min method* the membership functions of the consequent are cut at the alpha-value of the antecedent. So, defined U = {y_1, ..y_n} the universe of discourse of the output variable Y, we obtain new fuzzy sets A' B' and C' as follows:

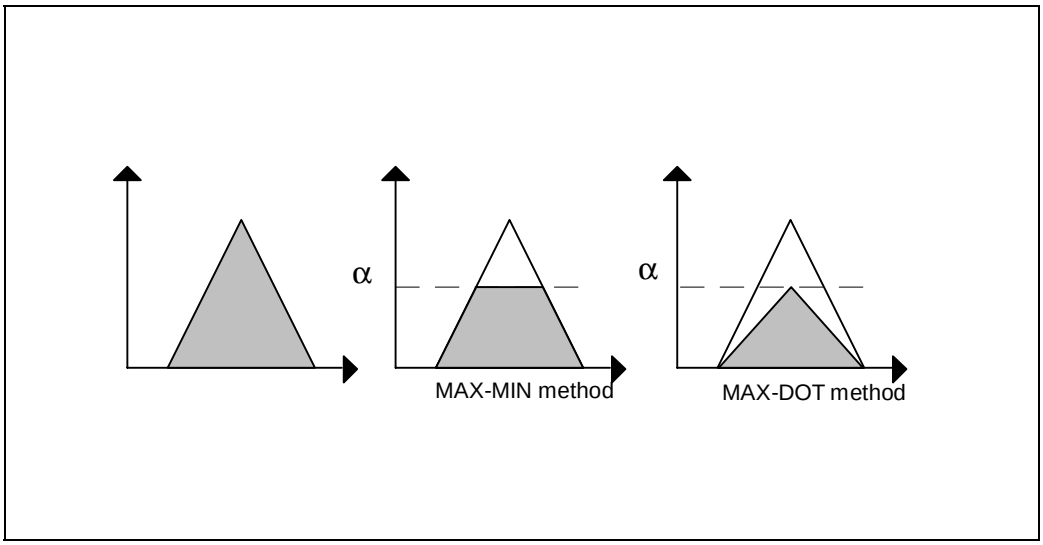
$A'(y_i)=\min(A(y_i),\alpha_1)$, $B'(y_i)=\min(B(y_i),\alpha_2)$, $C'(y_i)=\min(C(y_i),\alpha_3)$.

Using the *max-dot method* the membership functions of the consequent are scaled using the alpha-value of the antecedent. So, we obtain new fuzzy sets A' B' and C' as follows

$A'(y_i)=\min(A(y_i) \bullet a_1)$, $B'(y_i)=\min(B(y_i) \bullet a_2)$, $C'(y_i)=\min(C(y_i) \bullet \alpha_3)$.

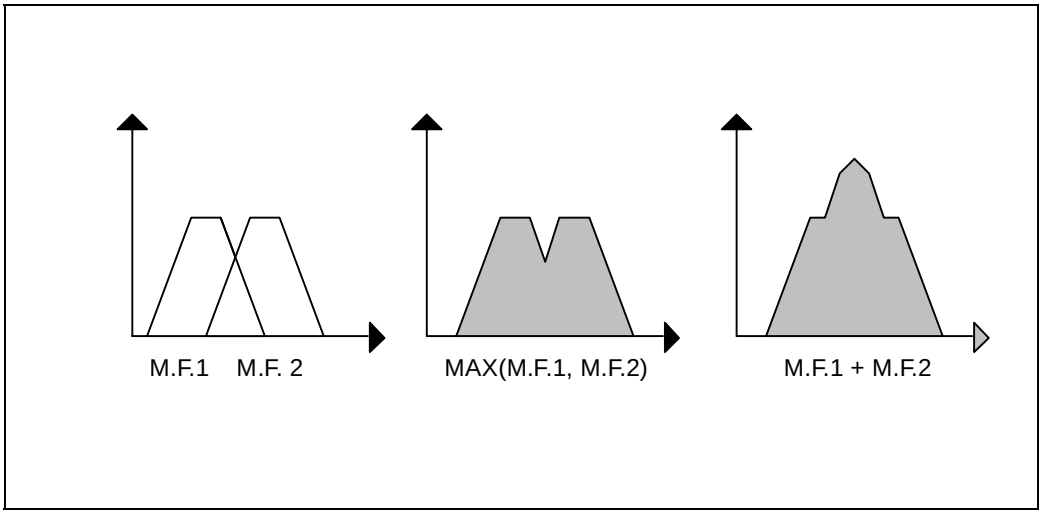
Fig. 13 gives a pictorial representation of the two most classical methods of inference.

Figure 13.



The membership functions of the consequent part, computed following one of the criteria mentioned above, represents the inferred fuzzy set for each rule. The next step is the combination of these fuzzy sets in order to deduce a single value for the output variables. It is realized summing the modified output fuzzy set to obtain a new global fuzzy set G. The sum can be performed in two different ways: either *logical sum* which corresponds to the logical operator **max,** or *arithmetic sum* which corresponds to the **point to point** summation of the membership function values. The difference between them is illustrated in the figure 14.

Figure 14.



The membership function G associated to the consequent is then used in the fourth step as follows.

Step 4:
defuzzification phase. The last step produces a crisp output from the fuzzy set G. In particular notice that this crisp value is an element of the universe of discourse. This crisp output will be the value of the control action. Many defuzzification methods have been proposed. They vary according to the specific application and the designer knowledge and understanding of the system. We will describe below the most commonly used in the hypothesis that G is defined over a finite universe of discourse $U=\{u_1,u_2,...,u_n\}$:

1 *Center of Gravity*: the output value y is given by the formula

$$y=\Sigma\, u_i\, G(u_i) \,/\, \Sigma\, G(u_i)$$

Therefore we obtain the center of gravity of the area belonging to the real plane identified by G.

2 *Centroid Method*: the output value y is given by the formula

$$y=\Sigma\, A_i\, b_i \,/\, \Sigma\, A_i$$

where i varies over the inference rules. A_i is the area of the modified output fuzzy sets (i.e. A'(Y), B'(Y), C'(Y) for the proposed example) and b_i the centroid associated with the fuzzy set.

In the following picture we will illustrate the difference between this two defuzzification methods. The first one start from the global output fuzzy set G and deduces a crisp value as the center of gravity of G. This approach implies a problem in case of a logic sum of the modified output fuzzy sets, since the common areas are taken once only, implying the exclusion of the fuzzy sets covered by the others. The second one, based on the area of the single modified output fuzzy sets, implicitly implies the arithmetic sum, thus the common areas are taken twice.

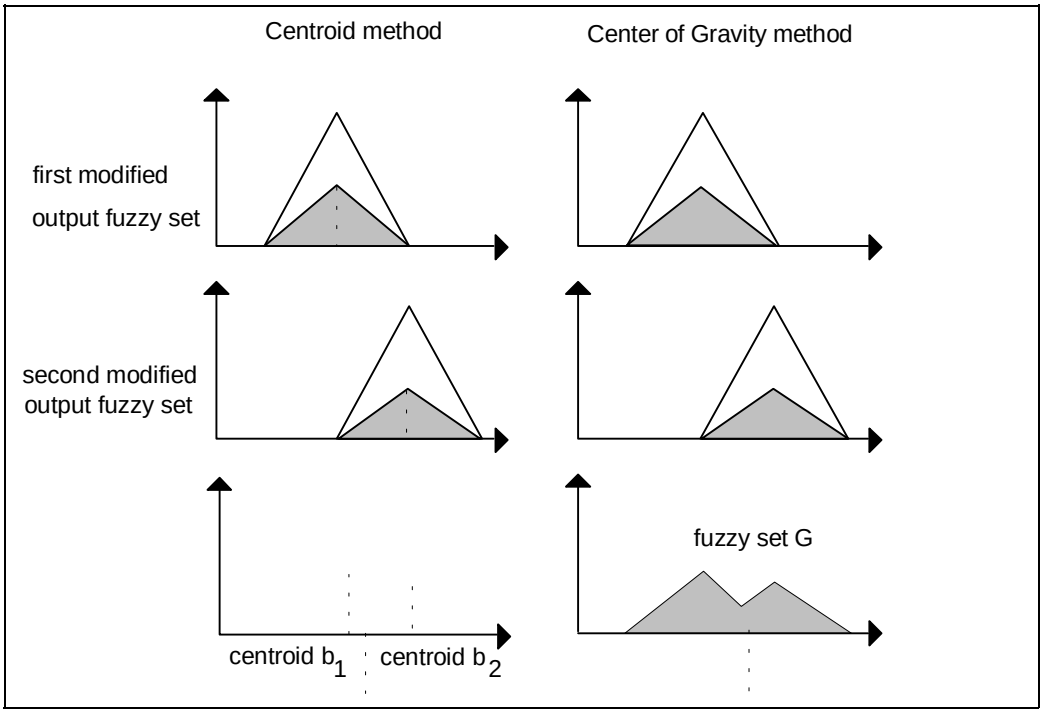
It is interesting to stress that, summing the modified output fuzzy sets using the arithmetic sum in the inference phase and applying the two different defuzzification methods, the result will be the same.

3 *Mean of Maxima*: the output value is given by the formula

$$y=\Sigma\, m_i / h$$

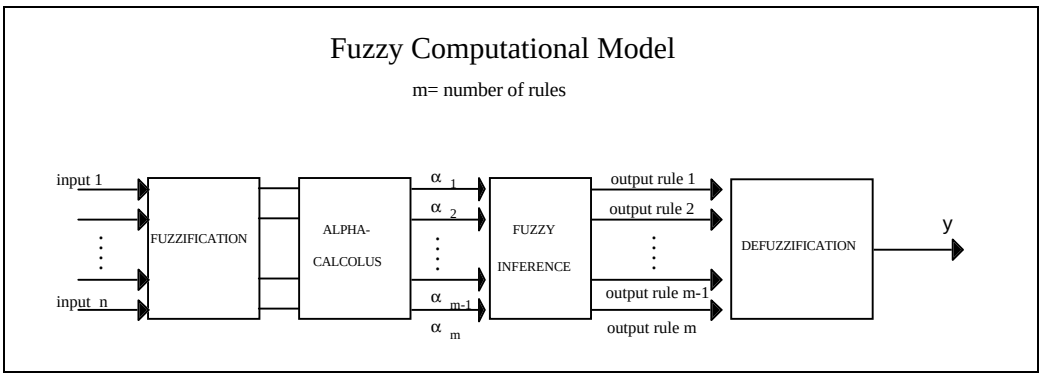
where $m_1, m_2, ..., m_h$ are the h values where of maximum membership degree is G.

Figure 15.



We conclude with a picture that summarizes the structure of a fuzzy computational model.

Figure 15.



Bibliography

[1] G. Klir, T. Folger: *Fuzzy Sets, Uncertainty and Information.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[2] D. Dubois, H. Prade: *Fuzzy Set and System: Theory and Applications.* New York: Academic Press, 1980.

[3] H. Zimmerman: *Fuzzy Set Theory-and its Applications*, 2nd ed. Boston: Kluwer, 1990

[4] A. Kauffmann, M. Gupta: *Introduction to Fuzzy Arithmetic: Theory and Applications*. New York: Van Nostrand Reihold, 1985

[5] T. Terano, K. Asai, M. Sugeno: *Fuzzy Systems Theory and its Applications.* New York: Academic Press, 1987.

APPENDIX B - Quick Reference
W.A.R.P. Control Language (WCL)

General Features of a WCL Program

WCL language is a medium-high level language dedicated for the programming of W.A.R.P. family processors.

Source Code Organization

A generic WCL program is organized into 3 sections, set in the following order:

- Chip target definition and characteristics
- Global variables declaration
- Procedure declaration

For More Information on each section internal structure refer to the following paragraphs.

Characters Conventions

WCL language employs all alphabetical characters, distinguishing between upper and lower cases, numeric digits and the following symbols:

‘#’, ‘_’, ‘-’, ‘+’, ‘=’, ‘<’, ‘>’, ‘&’, ‘^’, ‘|’, ‘!’, ‘(‘, ‘)’

The control characters (tabulation, return, ...) are considered as space characters.

The language is "case-sensitive", i.e. it distinguishes between "high", "HIGH", "High", ...

In particular, the following conventions are used:

- Keywords (‘begin’, ‘end’, ‘if’, ...) are entered in lower case;
- Library functions name have initial capitals;
- Predefined constants are typed in upper cases.

Naming Conventions

All names used within a WCL program are made up by a sequence of characters subject to the following constraints:

- The first character must be alphabetic;
- Only alphabetic, numeric characters and the symbol ‘_’ are admitted;
- Maximum sequence lenght is fixed to 32 characters.

Standard Libraries

WCL language allows to manage all W.A.R.P. family chips in the same way. Due to the different functionalities and peripherals that characterize W.A.R.P. family chips, the information about each chip is contained in a standard library.

This library allows to distinguish the control language for the chip controlled, but it should be continuously updated.

All names defined in the standard library for a specific chip are to be considered as reserved and cannot be used for other purposes (for example, in the global or local variables definition).

Conditional Expressions

A conditional expression is made up by an operand set connected by logic and/or relational operators, and supplies always a "true/false" value.

Operands

The operands admitted within a conditional expression are all the variables (predefined, global and local) visible within the evaluation context of the condition and the numeric values (signed or unsigned).

Operators

The operators admitted within a conditional expression can be of two types: logic or relational operators.

Logic operators have lower priority than relational operators. In case of numeric operands, the value "0" is considered false and true any other value.

The following operators are indicated according to a decreasing priority order:

- ! ("logic not" unary operator)
- && ("logic and" binary operator)
- **||** ("logic or" binary operator)

Relational operators have higher priority than logic operators. These are indicated below according to a decreasing priority order:

- <, >, <=, >= (less than, greater than, less than or equal to, greater than or equal to)
- ==, != (equal to, different by)

Within a conditional expression the use of parenthesis is not allowed. The operators can be associated from left to right except where requested in a different way by the priority of the operators involved. Moreover, it is not possible to use sequences of more operators of the same type. The maximum number of operators admitted in a conditional expression is 3, in order to evaluate the eventual membership to a value given to a given interval.

Library Functions

In a conditional expression it is also possible to use the following standard library language functions.

The available parameters are shown in table 1:

Table 1.

Function	Description	Parameter
`IsBitSet(bit,variable);`	Verifies if a variable bit value is '1'.	bit = 0 ÷ 7 variable *
`IsBitReset(bit,variable);`	Verifies if a bit variable value is '0'.	bit = 0 ÷ 7 variable *
`IsOverflow();`	Verifies if the last arithmetic-logic instruction performed has generated an overflow.	
`IsUnderflow();`	Verifies if the last arithmetic-logic instruction performed has generated an underflow.	
`IsOutOfRange();`	Verifies if the last arithmetic-logic instruction performed has generated an overflow o underflow.	
`TimerStatus(param);`	Verifies if the Timer is in Set or Reset status.	SET
	verify if the Timer is in Start or Stop status.	START
`SciStatus(param);`	Verifies the end of transmission	TX_END
	Verifies if the transmission buffer has been emptied	TX_EMPTY
	Returns true if the ninth bit of the frame is "1"	NINTH_BIT
	Verifies if an Overrun Error has occurred	OVERRUN
	Verifies if the reception buffer is full	RX_FULL
	Verifies if a Frame Error has occurred	FRAME_ERROR
	Verifies if a noise error has occurred	NOISE_ERROR

These library functions can be used instead of one of the generic operands of the conditional expression. Their result will be 'true' in case of positive check, otherwise it is false.

Reference Labels

A reference label is a name, unique inside the whole WCL program, that allows to refer to a particular position in the source code.

The label definition directly occurs in the position in which, through it, you want to refer, by using the following syntax:

```
label_name:
```

The symbol ':' does not belong to the label, it only indicates the end of the characters sequence that define the name. During the definition, such a symbol must not be separeted by the label name.

In a WCL program it is possible to make reference to a label before its definition, but only from another point inside the same procedure in which will be later defined the label.

Predefined Names

Besides the crisp global and local variables, inside a WCL program some predefined variables are also available. These allow to address particular configuration registers or the eventual data registers of the peripherals.

These variables can indicate read-only locations, write-only or read/write and their use is consequently constraint.

Any information relative to the predefined variables depends by the selected chip and it is therefore contained in the standard library language.

It is not possible to redefine the name of a predefined variable, using it for example in the definition of the user variables, either global and local.

Predefined variables are considered as "byte" variables, or unsigned.

Chip Features Definition

The definition of the chip features must be present at the beginning of the WCL program and provides a description of the chip required for the execution of the program, together with its particular configuration.

The instructions present in this section are introduced by the symbol "#", to highlight the definition functionality of the working environment (in parallel with similar pre-processing instructions of the C language).

Within this section there are two types of instructions:

1. *#chip chip_name*

   This instructions allows to identify a specific chip of W.A.R.P. family that will be used for the WCL program.

   The instruction must be the first instruction of the program, unique and it cannot be omitted.

   The argument ‘chip_name’ is a predefined constant, to be chosen among the ones contained for this purpose in the standard library language.

2. *#define environment_variable value*

   This instruction allows to define the particular working configurations of the chip and its internal peripherals, and is used to define the value of all possible functionment parameters of the chip selected, since apposite default values are not forecast.

   The argument ‘environment_variable’ is a predefined variable, contained in the standard library language.

   The argument ‘value’ depends from the particular ’environment_variable’ and must be chosen among the values contained for that purpose within the standard language library.

   All necessary instructions are grouped in a unique sequence located immediately after the instruction ‘#chip’.

Global Variables Declaration

This section immediately follows the one dedicated to the chip selection and its characteristics definition. Two sets of information have to be defined, in the following order: the Membership Functions at first and then the Global Crisp variables.

Membership Function Definition

This sequence of instructions allows to define the various memberships functions that will be later used by the fuzzy variables declared in the fuzzy procedures declared in the program.

The syntax is described below:

```
mbfxxx(distance_left, vertex, distance_right);
```

The name 'mbfxxx()' is an abbreviation for 'mbf000()', 'mbf001()', ..., up to the maximum number of 'membership' allowed by the particular chip selected.

The Membership Functions definition order is not relevant. It is possible to skip one or more numbering positions and it is also possible to place elements of such a list before or after. Only the necessary memberships have to be declared and it is possible to declare them in the desired order, provided that their declaration occurs inside a unique adjacent block of instructions 'mbfxxx()'.

The parameters *left_distance*, *vertex* and *right _distance* allow to define triangular Mbfs, appropriately locating the central vertex position on the x-axis (*vertex*) and the size of the right and left triangle semi-bases (*right_distance* and *left_distance*), as it has been defined for W.A.R.P. family chips.

For More Information Refer to ST52x301 Data Sheet.

Global Crisp Variables Definition

Sequence of instructions that allow to define the crisp global variables are visible within the whole program.

Two types of crisp variables, signed or unsigned, are available according to the following syntax:

- *byte variable_name [= unsigned_value];*

  Allows to declare, and eventually initialize, integer unsigned variables, defined in the range [0,255].

- *signed [byte] variable_name [= signed_value];*

  Allows to declare, and eventually initialize, integer signed variables, defined in the range [-128, 127]

In both cases, the parameter 'variable_name', being visible within the whole program, must be unique. Moreover, you cannot define define global crisp variables having the same name as the predefined variables.

The global crisp variables can be defined in any order provided that it is inside a unique adjacent block of global variables declarations.

The maximum number of crisp variables (global + local) allowed depends by the chip selected.

Procedure Declaration

A procedure is a particular set of instructions of the WCL language characterised by the following syntax:

```
begin type_procedure name_procedure;
instruction;
.....
instruction;
end name_procedure;
```

The parameter ‘name_procedure’ is a unique name within the whole WCL program.

The parameter ‘type_procedure’ identifies the particular type of procedure, allowing to choose among the different elaboration environment and the relative set of instructions.

The procedures are the following ones:

- Arithmetic
- Asm
- Control
- Folder
- Fuzzy

For More Information Refer to the following paragraphs for information on the different characteristics’ procedures.

Arithmetic Procedures

The Arithmetic procedure allows to perform logic-arithmetic computations, organized by means of simple control flow structures.

An arithmetic procedure can be defined within another control or folder procedure and cannot contain any type of procedure.

Local variables declaration

Inside the arithmetic procedure it is possible to define some crisp local variables, that will be added to the existing crisp global variables and will be valid in the single procedure.

The crisp local variables declaration must occur immediately after the beginning of the procedure, in a unique sequence of instructions that follow the same syntax adopted for the declaration of global crisp variables.

The maximum number of crisp variables (global + local) allowed depends by the chip selected.

It is not possible to declare local variables with the same name as predefined or global variables.

Flow control instruction

The 'if' instruction is the only flow control instruction available in an arithmetic procedure that allows to choose between two different sequences of instructions according to the evaluation of a conditional expression and is characterised by the following syntax:

```
if condition
then
instruction;
.....
instruction;
[else
instruction;
.....
instruction;]
endif;
```

The syntax of the condition is the one described in the apposite paragraph about the conditional expression.

Inside the blocks 'then' and 'else' you can use any instruction allowed inside an arithmetic procedure, and it is then possible to define 'if ' nested instructions.

Note: Library Functions described in Table 1 can be also used.

Logic & Arithmetic Operations

The logic & arithmetic operations allowed inside an inside procedure are simple assignment instructions between a variable and an expression that contains an arithmetic-logic operator, according to the following syntax:

variable = term operator term;

where 'term' can be another variable or numeric value.

The operators admitted are the following ones:
Arithmetic operators:

- +
- - (unary and binary);

Logic operators:

- ~,
- &,
- ^,
- | (not, and, xor, or)

The arithmetic operators can work with any type of operands while logic operators only work with 'byte' type operators (unsigned).

Moreover, it is also possible to arrange many of these operators with the assignment operators, obtaining the following assignment operators:

+=, -=, &=, ^=, |=

It is then possible to define expressions with the following syntax:

```
variable composed_assignement_operator term;
```

that is equivalent to the syntax:

```
variable = variable operator term;
```

Library Functions

In an arithmetic procedure it is possible to use the following library functions:

Table 2.

Function	Description	Parameter
`BitSet(bit, variable);`	Allows to set to 1 the value of a variable bit.	bit=0÷7 variable *
`BitReset(bit, variable);`	Allows to set to '0' the value of a variable bit.	bit=0÷7 variable *
`BitNot(bit, variable);`	Allows to negate the value of a variable bit.	bit=0÷7 variable *
`Reset () ;`	Restarts the program	

** The parameter "variable" is a byte or unsigned variable, visible in the procedure that contains the conditional expression.*

ASM Procedures

ASM procedures allow to insert sets of assembler instructions in a WCL program. An ASM procedure can only be defined in control or folder procedures and cannot contain any type of procedures.

Local Variables Declaration

In an ASM procedure it is possible to use any crisp variable defined at global level, besides particular crisp variables defined at local level and visible only inside that particular procedure.

The declaration of the local crisp variables must occur immediately at the beginning of the procedure, according to the sytanx and modality described for the definition of the global crisp variables.

In an ASM procedure it is possible to use and declare only byte crisp or unsigned variables.

However, in and ASM procedure it is possible to use, and then declare, only crisp variables of "byte" type, or unsigned. Eventual global crisp variables of signed type, or signed, will be be treated as unsigned variables.

The maximum number of crisp variables (global +local) allowed depends by the chip selected.

It is not possible to declare local variables with the same name as the predefined or global variables.

Instructions

The set of instructions available in an ASM procedure is a subset of the assembler instructions accepted by the selected chip.

For further information on the set of assembler instructions available in this procedure refer to the standard library language of the chip chosen.

In an ASM procedure it is not possible to directly address the registers of the chip, but only use the crisp variables (predefined, global and local) eventually available. Any reference to hardware addresses in an assembler instruction must be replaced by one of the crisp variables available.

Moreover, it is possible to use jump unstructions, provided that these are used only in the same procedure and they refer only to labels defined according to standard modalities .

Control procedure

It is the highest level type of procedure. All the other procedures can be present only inside a procedure of this kind, while a control procedure can only be defined outside any other procedure.

Each control procedure can allow to define the control flow of one of the procedures that make a WCL program. In fact, a WCL program wiill consists of at least one main control procedure, named 'main', and eventually, by other control procedures that describe the interrupt routines linked to the different peripherals of the chip selected, if employed by the user.

The name of a control prccedure is therefore chosen among a set of predefined names, contained in the standard library language and depending on the particular chip selected. This set will always contain the 'main' name, that indicates the description of the main control flow.

Local Variables Declaration

Currently it is not possible to define any type of local variables inside a 'control' procedure.

Control Flow Instructions

Two instructions for the flow control are supplied:

if instruction
Allows to choose between two different sequences of instructions according to the evalutation of a condition, and is characterised by the following syntax:

```
if condition
then
instruction;
.....
instruction;
[ else
instruction;
.....
instruction;]
endif;
```

The syntax of the condition is the one described in the apposite paragraph on conditional expressions.

Inside 'then' and 'else' blocks it is possible to use any instructions allowed in a control procedure. It is then possible to define if instructions variously nested and also other procedures (even if not of they are not control procedures).

goto instruction
Allows to interrupt the sequential execution of the instructions, jumping directly to the instruction indicated.

The syntax of a generic 'goto' instruction is the following one:

```
goto label;
```

The label is a name, unique inside the whole WCL program, that indicates the memory location. For the definition of a label refer to the relative paragraph.

return instruction
Allows to interrupt the instructions' sequential execution, skipping the procedure that contains it.

The syntax is the following:

```
return;
```

Computational expressions

It is not possible to define logic-arithmetic expressions directly inside a control procedure.

Library Functions

In a control procedure it is also possible to use the following library functions:

Table 3.

Function	Description	Parameter
`DeviceSet(peripheral, operation, ...);`	Allows to set the peripheral configuration status.	The parameter 'peripheral' and "operation" must be chosen among the names available in the standard library language related to the chip selected.
`IrqSetting(external_trigger_interrupt, interrupt_identifier, ...);`	Enables the management of all the interrupt signals indicated in the argument list (automatically disabling any signal not included in the same list.	'trigger_external_interrupt', to be chosen among the values supplied by the standard library language, indicates the signal that triggers the external interrupt.
		The parameters of the type 'interrupt_identifier' are to be chosen among the values available in the standard library language related to the chip selected.
`IrqPriority (interrupt_identifier, ...) ;`	Sets the interrupt signal priority.	The parameters of the type 'interrupt_identifier' are to be chosen among the values available in the standard library language related to the chip selected.
`Receive(variable_name,peripheral_identifier);`	Stores the current data of the peripheral selected in a variable.	The parameter 'variable_name' can refer to any crisp variable (predefined and global) visible in the current context.
		The parameter 'peripheral_identifier' is to be chosen among the values available in the standard library language related to the chip selected.
Send(peripheral_identifier, variable_name);	Sends the values of a variable to the peripheral selected.	The parameter 'peripheral_identifier' is to be chosen among the values available in the standard library language related to the chip selected.
`Wait();`	Interrupts the performing of the program's instructions waiting for one incoming interrupt signals enabled.	

Folder Procedures

It is a particular version of the control type procedure, that can be contained inside other control or folder procedures.

Refer to the paragraph relative to control procedure for more information.

Fuzzy Procedures

It is a procedure that allows to define a single fuzzy system according to the chip selected.

A fuzzy procedure can only be contained inside a control or folder type procedure.

The fuzzy procedure consists of two sections set in the following order:

- fuzzy variable declaration
- fuzzy rules definition

Further information is provided in the relatives paragraphs.

Fuzzy Variables declaration

In a fuzzy procedure the first step to perform is to declare the fuzzy variables to be used by the its fuzzy rules.

The fuzzy variables declaration must occur in a unique block of instructions. These do not follow a particular order but must be necessarily placed at the beginning of the procedure according to the following syntax:

```
begin fuzzyvar variable_name;
input (crisp_variable_name);
output (crisp_variable_name);
storein (number)
domain (lower_bound, upper_bound);
mbf (membership_name, membership_value);
.....
mbf (membership_name, membership_value);
end variable_name
```

The single fields of a ‘fuzzy’ variable must be strictly set in the sequence indicated. It is not necessary to follow any particular order inside the declaration of the fields ’mbs ( )’.

The parameter ’variable_name’ must have a unique name inside the fuzzy procedure.

The parameters ‘lower_bound’ and ‘upper_bound’ of the field ‘domain()’ allow to indicate the Universe of Discourse bounds of the variable. The indicated values are included in the Universe of Discourse.

The parameter ‘membership_name’ must have a unique name inside the current variable declaration.

The parameter ‘membership_value’ allows to associate its value to a membership function. The value of an input variable is chosen among the ones defined in the section related to the Membership Functions definition, included in the section of the Global Variables declaration, then its nameos of the type ’mbfxxx’. The value of an output variable is a crisp value (signed or unsigned) belonging to the Universe of Discourse indicated.

The keyword ‘input()’and ‘output()’ are mutually exclusive. They identify the type of fuzzy variable. Moreover, the field ‘input()’, used only for the input variables, indicates the crisp variable necessary to supply the iniatialisation value of the fuzzy variable. The field ‘output()’, used only for the output variables, indicates the crisp variable to be initialized with the output variable’ value at the end of the fuzzy procedure processing. In both cases, the parameter ‘crisp_variable_name’ can contain onlt the global crisp variable or a predefined variable. When output () keyword, the strein () keyword must also be supplied, the parameter "number" indicates the number of the logic output of the fuzzy computational unit.

Fuzzy Rules definition

The main part of a fuzzy procedure is the set of fuzzy rules declared immediately after the fuzzy variables. These do not have a particular order but follow the syntax:

```
if antecedent_list then consequent_list;
```

where:

```
antecedent_list ::= antecedent [fuzzy_operator antecedents_list]
antecedent ::= input_variable_name is [not] membership_name
```

and

```
consequent_list ::= consequent [and consequent_list]
consequent ::= output_variable_name is output_variable_value
```

The term ‘fuzzy_operator’ can indicate only one of the following operators: ‘and’ , ‘or’.

The term’input_variable_name’ refers to the name of a fuzzy input variable defined in a procedure, and the term membership_name’ refers to the name of a ‘membership’ defined for this variable.

The term ’output_variable_name’ refers to the name of a fuzzy output variable defined in a procedure, but the term ‘output_variable_value’ can refer to the name of one of the ’memberships’ defined for this variable or directly to a crisp value (signed or unsigned) belonging to the Universe of Discourse defined for this variable.

The maximum number of antecedents and consequents that can be used in a fuzzy rule depends on the chip selected.

ST52x301 Standard Library

The following keys refer to the processor's configuration and are used together with the instruction of the type #DEFINE keys [value].

Processor's type:

WARP3_TC	Indicates the ST52x301 processor.

Working frequency key:

FREQUENCY	n = 5/10/20

A/D Converter:

ADC_CHAN	n specifies the number of channel to be converted. n = [1 , 4]

Parallel Port:

PORT_DIR	b	specifies each pin's direction. X=I input pin X=O output pin	b = XXXXXXXX

Timer:

TIMER_PRES	n	specifies the Prescaler's value	n = [0 , 65535]
TIMER_WAVE	n	specifies the type of output	n = SQUARE / PULSE
TIMER_POLAR	n	specifies the output polarity	n = H / L
TIMER_START_SRC	n	specifies the start signal source	n = INT/EXT
TIMER_START_DET	n	specifies the start detection	n = LEV / EDGE
TIMER_LOAD	n	specifies the counter data source	n = FUZZY0 FUZZY1 REGISTER
TIMER_CLK	n	indicates the clock's source	n = INT / EXT
TIMER_INT	n	indicates the interrupt's source	n = STOP / H / L / HL/NONE

Triac Driver:

TRIAC_PRES	n	specifies the Prescaler's value	n = [0 , 65535]
TRIAC_MODE	n	specifies the working modality	n=PWM/BURST/PHASE
TRIAC_POLAR	n	specifies the output polarity	n = H /L
TRIAC_ LOAD	n	specifies the counter data source	n = FUZZY0 FUZZY1 REGISTER
TRIAC_INT	n	indicates the interrupt source	n = H / L / HL /None
TRIAC_CLK_SRC	n	specifies the Clock's source	n=INT/EXT/POW
TRIAC_RETEIO_DIR	n	specifies the RETEIO pin direction	n = I / O

TRIAC_PULSE n	specifies the triac pulse length	n = [0 , 32767]
TRIAC_POW_FREQ n	specifies the power line frequency	n = 50 / 60
TRIAC_MASK_TIME n	specifies the masking time	n = [0 , 6553.7]

Serial Communication Interface

SCI_BAUD_RATE n	Specifies speed transmission	n = 600 1200 2400 4800 9600 19200 38400 EXT
SCI_FRAME_BITS n	Specifies the frame bit number	n = 8/9
SCI_PARITY p	Specifies the use of the parity bit	p = Even/Odd/None
SCI_STOP_BITS s	Specifies the use of the stop bits	s = 1/2
SCI_INT n	Specifies the possible interrupts' source	n = TX_EMPTY TX_END RX_FULL OVERRUN BREAK NONE

Peripherals Identifier for main Procedure Instructions

TIMER	Identifies theTimer peripheral
TRIAC	Identifies the Triac Driver peripheral
ADC	Identifies the A/D converter peripheral
PORT	Identifies the Parallel port peripheral
SCI	Identifies the serial output
PIN	Identifies the Port Direction pin

Predefined Variables

Read-only variables:

CHAN0	Identifies the variable in which the value of the A/D 0 channel is stored.
CHAN1	Identifies the variable in which the value of the A/D 1 channel is stored.
CHAN2	Identifies the variable in which the value of the A/D 2 channel is stored.
CHAN3	Identifies the variable in which the value of the A/D 3 channel is stored.
TIMER_STATUS	Identifies the variable in which the Timer status is stored.
SCI_STATUS	Identifies the variable in which the SCI status is stored.
FUZZY0	Identifies the first Fuzzy output.
FUZZY1	Identifies the second Fuzzy output.
ADC_STATUS	Identifies the variable in which the value of the A/D Converter status is stored.

Write-only variables

TRIAC_COUNT	Identifies the variable in which the Triac Counter value is set.
TRIAC_PRES	Identifies the Triac Driver Prescaler.
TIMER_PRES	Identifies the Timer Prescaler.

Read/Write Variables

TIMER_COUNT	Identifies the variable in which the Timer Counter value is written or read.
PORT	Identifies the variable with the values written/read in the parallel port.
SCI_BUFFER	Identifies the variable with the values written/read in the serial.

Peripherals' Configuration Variables

The peripherals configuration registers can be addressed in the assembler procedures with instructions of the type:

```
LDCF  REG_CONFXX , value
```

where XX is the range [0,15].

Moreover, in the assembler blocks, the instructions that adress the peripherals' registers, can use the predefined variables previously described; more precisely:

The instruction LDRI can use the following predefined variables:

CHAN0, CHAN1, CHAN2, CHAN3, TIMER _STATUS, SCI_STATUS, FUZZY0, FUZZY1, TIMER_COUNT, PORT.

The instruction LDPR can use the following predefined variables:

TRIAC_COUNT, TIMER_COUNT, PORT.

Interrupts Service Routines Keywords

TIMER	Identifies the Timer Interrupt routine.
TRIAC	Identifies theTriac Driver Interrupt routine.
ADC	Identifies the A/D Converter Interrupt routine.
SCI	Identifies the serial port's Interrupt routine.
EXT	Identifies the external interrupt routine.

Timer Status Keywords

SET	Verifies if the Timer is in Set or Reset status.
START	Verifies if the Timer is in Start or Stop status.

SCI Status Keywords

TX_END	Verifies the end of transmission.
TX_EMPTY	Verifies if the transmission bufferhas been emptied.
NINTH_BIT	Returns true if the ninth bit of the frame is 1.
OVERRUN	Verifies if an Overrun Error has occurred.
RX_FULL	Verifies if the reception buffer is full.
FRAME_ERROR	Verifies if a Frame Error has occurred.
NOISE_ERROR	Verifies if an Noise Error has occurred.

Appendix C - Assembler Language

Program Memory and Registers' Architecture

To program in Assembler, it is important to consider the processor's architecture, in particular the address space: Program Memory and registers.

Program Memory

The Program Memory is the EPROM memory where the program is stored. This is made up by three main sections with a fixed space:

Membership Functions (MF) Data Memory	from 0 to 191
Interrupt Vectors	from 192 to 201
Program Space	from 202 to 2047

The MF Data Memory contains the data describing the Membership Functions in a codified form: 3 bytes per MF that represent respectively the left semi-base, the vertex position in the Universe of Discourse and the right semi-base.

The memory locations about the Interrupt Vectors are organised in byte couples where the routine service addresses in the Program Memory are contained. In particular, the locations 192 - 193 contain the start address of the routine service of the interrupt associated to the A/D Converter the locations 194 - 195 to the SCI Interrupt, the locations 196 - 197 to the Timer Interrupt. The locations 198 - 199 to the Triac Driver Interrupt and the location 200-201 to the External Interrupt.

The Program Memory starts at the address 202. It contains all the boolean, arithmetics and fuzzy instructions. The traditional boolean or arithmetic instructions and the ones for the flow control of the program (Jump instructions) are separated from the fuzzy instructions by a STOP instruction and vice versa.

Note *The fuzzy computation cannot be interrupted, then a fuzzy computation block can be considered as a single assembler instruction. The eventual pending Interrupts are served after the STOP instruction.*

Register File

The Register File is a set of 16 registers that can be used either as read and write (addresses 0-15). These are the registers in which you can perform arithmetic and boolean operations. Several assembler instructions can address these locations:

ADD	Additions of the registers.
AND	Logic AND between two registers.
LDRC	Register loading with a constant.
LDRI	Register loading with the content of one input register.
LDPR	Peripheral register loading with a register of the Register File.
LDRR	Register loading with another register.
SUB	Subtraction between registers.
SUBO	Subtraction between registers with Offset.

All 16 registers can be considered GENERAL PURPOSE but some of them have a particular use:

Registers from 0 to 3:
Used as inputs to the fuzzy processing unit.

In particular, the registers 0 to 3 are also used by FUZZYSTUDIO™ 3.0 as working registers for the execution of the instructions at high level. Then these four registers can be used only working directly in assembler, excluding the loading of the inputs to the fuzzy core that is an operation inserted automatically by the Compiler in a transparent way for the user.

In FUZZYSTUDIO ™ 3.0 Assembler Block, these registers must be addressed by using Global variables.

Configuration Registers

The Configuration Registers file consists of 16 write only registers (addresses 0-15). These registers have the task to contain the internal peripherals' configuration to the processor. It is only possible to load some constants on these registers through the following instructions:

LDCF	Loads a constant on the configuration register specified.
IRQM	Determines the interrupts mask: it is equivalent to a LDCF on the register 14.
IRQP	Determines the interrupts priority: it is equivalent to a LDCF on the register 15.

Input Registers

The input registers file is consists of 11 read-only registers (addresses 0 - 10). These registers allow to read the peripherals' values and to check their status. They can be addressable only by means of the instruction **LDRI** that reads the value from the input register specified and loads it on the specified register of the Register File.

The registers have the following functions:

Register 0	A/D Converter Channel 0 converted data.
Register 1	A/D Converter Channel 1 converted data.
Register 2	A/D Converter Channel 2 converted data.
Register 3	A/D Converter Channel 3 converted data.
Register 4	Timer Counter value.
Register 5	Timer Status register: it gives information about the set/reset (bit 0) and start/stop status (bit 1).
Register 6	Parallel Port Input data register.
Register 7	Serial input register.
Register 8	SCI status register; gives information about SCI. (See data sheet for more information.)
Register 9	Fuzzy Output 0
Register 10	Fuzzy Output 1

Note *In addition, the serial Input register must be directly accessed by a SRX instruction. Use this instruction for a correct execution of the data reading. In an assembler block these register are addressed by means of Predefined Variables.*

Peripherals' Data Registers

The Data Registers File of the peripherals consists of three registers write only (addresses 0 - 2). Their function is to write particular data to be used by some of the peripherals. They can be addressed only with the instruction LDPR that transfers the content of the register specified of the register File on the Data Register of the peripherals specified. Their registers are:

Register 0 Timer Data Counter.

Register 1 Triac Data Counter.

Register 2 Parallel Port Output data register.

In addition, Serial Output register can be accessed directly by a STX instruction. The related register is contained inside the peripheral, so it does not belong to this file.

Note *In an Assembler block these registers are addressed by means of Predefined Variables.*

Flags

ST52x301 core owns two flag bits with four stack levels for the interrupts. This means that both the main program and all the interrupt routines have their own flags. A Return from Interrupt restores the flags status at the moment of the interrupt request. The flags bit are two:

S Sign flag: it is set in case of overflow or underflow.

Z Zero flag: it is set when the result of an operation is zero.

The instructions that modify the flags are the following ones:

ADD Sets the zero flag when you add two registers both containing zero; sets the sign flag when the sum of the two registers is higher than 255: in this case the result is always sum-256.

AND Sets the zero flag when the result of the operation is zero

SUB Sets the zero flag when the result of the operation is zero; sets the sign flag when the value of the destination register is lower than the value of the source register that is the result should be negative: in this case the result of the operation is 256+result.

SUBO Like SUB but with the adding that in case of overflow both flags are set. This can occur when the destination register is bigger than the source one of at least 128, in fact the instruction SUBO adds 128 to the result of the subtraction and then the result would be higher than 255.

The flags are taken into consideration by the instructions of conditional jump that are:

JPZ Jumps if zero flag is set.

JPNZ Jumps if zero flag is not set.

JPS Jumps if sign flag is set.

JPNS Jumps if sign flag is not set.

Note *In the Assembler block jumps are allowed only inside the block. Other instructions are described in the following paragraphs.*

Fuzzy Programming in Assembler

The programming of the fuzzy functionalities in assembler is a complex task. For this reason it is more convenient to perform the programming by using the graphic tools provided by FUZZYSTUDIO™ 3.0. Anyway, this complex task can be performed with the arithmetic/logic assembler programming, by using the FS3ASM.EXE tool included in the installation directory of FUZZYSTUDIO ™ 3.0 (see Appendix D for further information).

The fuzzy programming in assembler is divided into two phases: the first to define the Mbfs and the second for the rule inference.

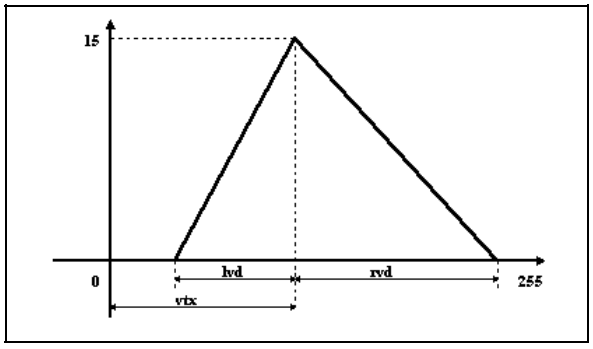
Membership Functions definition

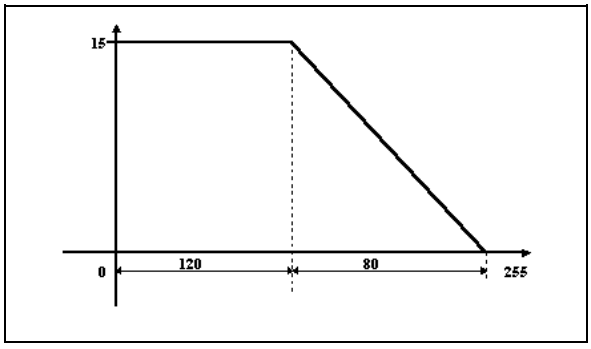
The assembler instruction to define the Membership Functions is DATA. It indicates to the Compiler which data have to be loaded on the Mbf Data memory according to the programmer's specifications. The DATA instruction syntax is the following one (see figure on the left):

DATA var mbf lvd vtx rvd

where:

var	the order number of the variable to which the Mbf is associated.
mbf	the order number of the mbf that you are defining.
lvd	distance of the left vertex Mbf from the central vertex [from 0 to 255].
vtx	position of the central vertex in the Universe of Discourse [from 0 to 255].
rvd	distance of the right vertex Mbf from the central vertex [from 0 to 255].





Note *In case you want to store trapezoidal Mbfs to the extremes of the Universe of Discourse, the value of lvd (if the horizontal side is the left one) or the value of rvd (if the Horizontal side is the right one), is 0. The following Membership Function is stored as third Membership Function of the second variable with the instruction:* **DATA 1 2 0 120 80**

The Membership Functions definition by means of DATA instructions must end with the instruction STOP.

Rule Inference

The instructions for the rules inferencing are described in the Appendix C relative to the structure of the Assembler language and to the fuzzy instructions.

The fuzzy computation unit is made up by:

- Multiplier block for the α calculation and for the consequent inference.
- A two-level stack to contain the two operands of a fuzzy operation.
- Temporary Buffer to store partial results.
- Computational block for AND (min) and OR (max) operations.
- Adder to obtain the partial results of the defuzzification.
- Two registers to hold the partial results of the defuzzification.
- Divider for the defuzzification and the calculation of the output.

Assembler instructions operate on these devices in the following way:

Let us suppose you have previously defined the Mbf with **DATA** instructions, the rule:

IF Inp_0 is NOT MF_{01} AND Inp_2 is MF_{21} OR Inp_3 is MF_{33} THEN $CRISP_1$

is therefore codified as:

LDN 0 1	Loads in the stack the NOT α value relative to the first term of the rule.
LDP 2 1	Loads in the stack the NOT α value relative to the second term of the rule.
FZAND	Calculates the min between two values in the stack.
LDK	Stores the result of the previous operation in the stack.
LDP 3 3	Loads in the stack the value of the α relative to the third term of the rule.
FZOR	Calculates the max between the two values in the stack.
CON 58	Performs the product between the values calculated and the value $CRISP_1 = 58$ (consequent calculus)

The Assembler instructions work on these devices in the following way.

Let us suppose you have previously defined the Membership Functions with the instruction **DATA**, the rule:

IF (Inp2 is MF21 AND Inp3 is NOT MF35) OR (Inp0 is MF03 OR Inp1 is NOT MF16) THEN CRISP2

is codified with the following instructions:

LDP 2 1	Loads in the stack the value of the α relative to the first term of the rule.
LDN 35	Loads in the stack the NOT α value relative to the second term of the rule.
FZAND	Calculates the min between the two values in the stack.
SKM	Stores the calculated value on the temporary register.
LDP 0 3	Loads in the stack the value of the α relative to the third term of the rule.
LDN 1 6	Loads in the stack the NOT α value relative to the fourth term of the rule.

FZOR Calculates the max between the two values in the stack.

LDK Stores the result of the previous operation in the stack.

LDM Copies the content of the temporary register in the stack.

FZOR Calculates the max between the two values in the stack

CON 35 Performs the product between the value calculated and the value CRISP1=35 (Consequent calculus).

After the inference of all the rules relative to an output, you can obtain the output through the instruction:

OUT 0 To calculate the first fuzzy output.

The fuzzy computation instructions always end with a **STOP** instruction and are preceded by another instruction **STOP** to end the Assembler instructions of the ALU.

The rules that can inferenced in assembler must have a max format of eight antecedent terms and one consequent term. More complex rules can be reduced into equivalent ones with an allowed format.

Note *All rules relative to an output have to be consecutive to calculate the output.*

The Structure of a Program

ST52x301 programs in assembler language follow the rigid structure shown below: The easiest program for a chip of W.A.R.P. family consists of a set of arithmetic instructions as the one shown in following list:

To the scheme presented above, formed by a single section of arithmetic instructions, you can add other sections to form a structure as the one shown in the first figure.

```
;
; Arithmetic Instructions Set
;
loop:   jp     loop
        stop
```

The last two sections in the figure must necessarily be present in couple to follow the scheme, but they can be repeated (or not be present) to the user discretion.

Data definition
Interrupt Vector definition
Arithmetic instructions
Fuzzy Instructions
Arithmetic instructions

Each section is made up by an assembler code line sequence, with the eventual insertion of blank lines, and is ended by the instruction STOP.

.....
.....
Arithmetic instructions
Fuzzy instructions
Arithmetic instructions
.....
.....
Fuzzy instructions
Arithmetic instructions

Note *The programmer has to make sure that the last arithmetic instruction is the unconditional jump. On the contrary, even if the assembler program is syntactically correct, the chip would continue to execute and sequentially perform the EPROM content, with malfunctionment.*

Structure of a Generic Code Line

Each code line consists of a single instruction followed by the relative topics. The instruction must not necessarily be at the beginning of the line, but it can be preceded by any number of space character. The instruction is separated by its arguments by at least one space character. The arguments are separated among them by at least one space character and optionally, by a " , " character. The tabs characters are considered as space characters. The use of capital or lower case letters is equivalent.

Comment sequences

It is possible to insert comment sequences in the code by inserting a " ; " character before. The comment sequences can be written along the whole line until its end. It is possible to insert only single comment lines or add a sequence of comment at the end of the instruction.

Line label

A line label is a particular character's sequence that allows to univocally determine a code line and then an assembler instruction. A line label must begin with an alphabetic character and can only contain alphanumeric characters. The definition of a line label occurs only when inserting that label at the beginning of the corresponding code line (the new label can be preceded only by space characters) and making it follow by a " : " character The maximum length of a labe is 32 characters.

The character " : " does not belong to the labels: it is used only to define a new label, to end the corresponding alphanumeric sequence. Furthermore, the character " : " does not carry out as separator and has to be followed by one or more space characters.

ST52x301 assembler language allows to define line labels only within blocks of arithmetic instructions, that is only to refer to arithmetic instructions. However, it is possible to associate line labels to blank lines containing only comment sequence, but those lines have to be followed by at least one line containing an arithmetic instruction.

Data definition section

It is a section of the program that allows to define the Mbfs for the fuzzy variables that will be used in the program. This section is necessary to use the fuzzy part of a chip of W.A.R.P. family. Without it fuzzy instructions sections can not be admitted. The instructions allowed within a data definition section are described in the following table:

Table 1. Data Definition instructions

DATA var mbf lvd vtx rvd var = variable mbf = membership function lvd = left Mbf vertex distance vtx = vertex rvd = right Mbf vertex distance	Stores a fuzzy variable Mbf, defining vtx, lvd and rvd.
STOP	End of Data Definition section

Interrupt Vectors Definition

It is a program's section that allows to define the starting address for the interrupt procedures that will be defined in the program. The interrupt procedures starting addresses are supplied through apposite line labels that identifies the first code line. The programmer is not obliged to define the interrupt procedures for the available signals. It is then possible to omit even the entire interrupt vector definition section. The instructions allowed within the section are described in table 2.

Arithmetic instructions section

It is an executable code section, dedicated to the management of the arithmetic part of the chip (refer to table 3 to 7 for Instructions set). In particular, in this type of sections it is possible to manipulate the various registers of the chip (Table 3), perform arithmetic and logic operations (Table4) and execute the Jump instructions (Table 5).

Table 2. Interrupt Vector section definition

IRQ num label num = interrupt signal label = starting address	Assigns the routine initial address to the corresponding interrupt signal.
STOP	End of Interrupt Vector Definition section

Table 3. Registers management

LDCF reg cost reg = configuration register cost = constant value	Stores a constant value into a configuration register.
LDPR dev reg dev = peripheral register reg = register value	Stores a Register File value into a peripheral register.
LDRC reg cost reg = register cost = constant value	Stores a constant value into a register file element.
LDRI reg input reg = register input = input register	Stores an input register value into a Register File element.
LDRR dest src dest = destination register src = source register	Stores a Register File element into a Register File element.

The arithmetic instructions sections have to be used to define possible interrupt procedures (refer to instructions in table 6).

Table 4. Arithmetic Logic operations

ADD reg1 reg2 reg1 = register reg2 = register	Sum operation reg1 = reg1 + reg2 It sets the Sign Flag (overflow). It sets the Zero Flag (null result)
AND reg1 reg2 reg1 = register reg2 = register	Logic AND operation reg1 = reg1 and reg2 It sets the Zero Flag (null result)
SUB reg1 reg2 reg1 = register reg2 = register	Subtraction operation reg1 = reg1 - reg2 It sets the Sign Flag (negative result) It sets the Zero Flag (null result).
SUBO reg1 reg2 reg1 = register reg2 = register	Subtraction operation with offset reg1 = reg1 - reg2 + 128 It sets the Sign Flag (negative or overflow result) It sets the Zero Flag (null or overflow result).

Table 5. Jump operations

JP lab lab = instruction address label	Jumps to an instruction.
JPNS lab lab = instruction address label	Jumps to an instruction, if the Sign Flag is not set.
JPNZ lab lab = instruction address label	Jumps to an instruction, if the Zero Flag is not set.
JPS lab lab = instruction address label	Jumps to an instruction, if the Sign Flag is set.
JPZ lab lab = instruction address label	Jumps to an instruction, if the Zero Flag is not set.

Table 6. Interrupt signals management

IRQM mask mask enable	Interrupt routines enable/disable.
IRQP priority priority priority code	Interrupt signal priority setting.
MDGI	Temporarily disables the interrupt signals management.
MEGI	Reactivates the management of the interrupt signals.
RETI	End of an interrupt routine.
RINT num num = interrupt signal	Resets an interrupt signals.
UDGI	Temporarily disables the interrupt signals management.
UEGI	Reactivates the interrupt signals management.
WAITI	Stops the program until the overcoming of an interrupt signal.

Table 7. Arithmetic instructions for the serial port.

SRX reg reg = register	Receives data from the serial and download in a register file's register.
STX reg reg = register	Transmitts the register file's register content to the serial.

Table 8.

NOP	Null instruction.
STOP	End of an arithmetic instruction section.

If an arithmetic operation generates an overflow, the destination register is set to the result value equal to the result decreased by 256. if an arithmetic operation generates an overflow, the destination register is set to the value 256 decreased by the result.

Fuzzy Instructions section

It is an executable code section, dedicated to the management of the fuzzy part of the chip. The fuzzy instructions sections are allowed only with the presence of a data definition section. Moreover, each fuzzy section has to be necessarily followed by the arithmetic instructions block (i.e. at least by a jump operation in order to execute properly the program). The instructions allowed within a fuzzy block are listed in Table 8. These are instructions aimed at the definition of fuzzy rules set of whose valuation determines the fuzzy output values.

In W.A.R.P. family, each fuzzy rule is consists of 2 - 8 antecedents and 1 consequent. Each antecedent is the result of the fuzzyfication of the value of one fuzzy variable according to its particular membership, while the consequent is a constant value. The combination of the antecedents, eventually negated, through fuzzy **AND/OR** operations generates a weight to evaluate the consequent.

The two available operations, **FZAND** and **FZOR**, work on a stack with two positions that must be previously loaded by the user through the **LDN** and **LDP** instructions. Being these commutative operations, the loading order of the values on the stack is not relevant. The stack is automatically emptied after the performing of each fuzzy operation.

The instructions **LDN** and **LDP** provide the loading of the stack with the result of the fuzzyfication of the current value of a fuzzy variable according to a given membership (refer to table 8 for the difference of the two instructions). In particular, the first argument of these instructions indicates either the fuzzy variable in question and the element of the register file in which the user has previously transferred the value of the corresponding input register.

Table 9. Fuzzy Instructions

CON consequent consequent constant	It multiplies the result of the last fuzzy operation with the crisp value of the consequent.
FZAND	It implements the fuzzy operation AND between the last two values stored in the data stack.
FZOR	It implements the fuzzy operation OR between the last two values stored in the data stack.
LDK	It stores the result of the last fuzzy operation in the data stack
LDM	It stores the value of temporary buffer in the data stack
LDN var mbf var = fuzzy variable mbf = membership function	It calculates the NOT α value (result of fuzzification) of the fuzzy input with the indicated membership function and stores the result in the data stack.
LDP var mbf var = fuzzy variable mbf = membership function	It calculates the α value (result of fuzzification) of the fuzzy input with the indicated membership function and stores the result in the data stack.
SKM	It stores the result of the last fuzzy operation executed in the temporary buffer.
STOP	End of a Fuzzy Instruction section.
OUT output output = fuzzy output	It performs the defuzzification of a fuzzy output.

After the performing of a fuzzy operation, the programmer has to manage the result by means of one of the instructions **LDK**, **SKM**, **CON**. In particular, the instruction **LDK** allows to load again the result of a fuzzy operation in the stack, to use it in the following operation, while the instruction **SKM** stores that result in a temporary buffer, from which it could be loaded in the stack by means of the instruction **LDM**.

The instructions **SKM** and **LDM** implement the equivalent of a couple of brackets, that cannot be however nested.

The instruction **CON** associates the result of the last fuzzy instruction to a constant value, using it as a weight to evaluate such value.

The instruction **OUT** performs the defuzzification of a fuzzy output using the results of all the previous **CON** instructions.

Appendix D - FS3ASM: Assembler Programming Tool.

The file FS3ASM.EXE, provided in the FUZZYSTUDIO™ 3.0 installation directory, is a tool that allows to program ST52x301 in assembler.

It consists of a text editor for the writing and editing of the assembler program, completed by the Assembler to generate the machine code and the commands for the programming of the devices by means of the board linked to the PC through the parallel cable.

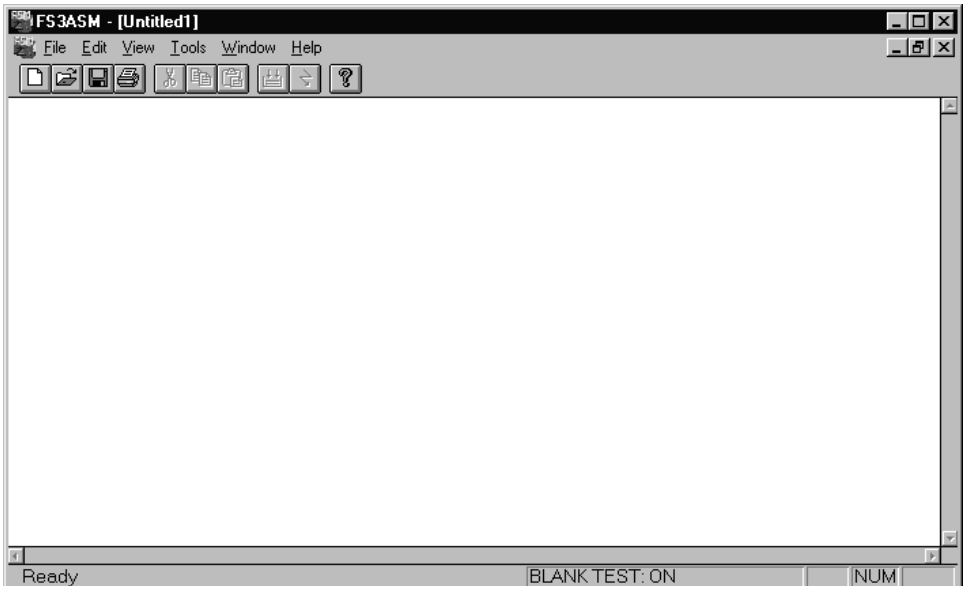
It is possible to generate the Assembler program starting from the beginning or loading a file with the extension .ASM generated by FUZZYSTUDIO ™ 3.0 and modify it. In any case, the original program written with FUZZYSTUDIO™ 3.0 will not be changed and will not correspond to the generated code.

FS3ASM Main Window

This section provides an overview of the major elements of the Assembler programming window, such as menus, toolbar and status bar. For additional information, see the index and online Help.

FS3ASM Menus

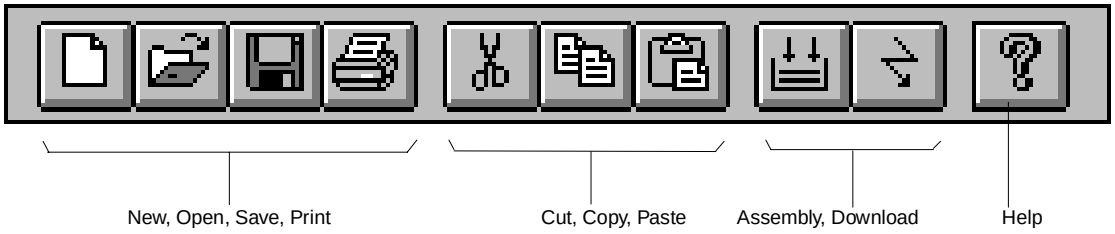
FILE	Provides standard commands for the management of files, printing and a list of the most recently used files.
EDIT	Contains standard commands for the editing of the program.
VIEW	Provides commands to show/hide the toolbar and status bar and font settings commands.
TOOLS	Provides commands for the machine code generation and the device programming by means of the programming board.
WINDOWS	Contains commands related to windows management.
HELP	Contains help commands.



FS3ASM Toolbar

The FS3ASM toolbar allows you to perform the most frequently used commands quickly. To execute a task by means of a button, just click the related button on the toolbar.

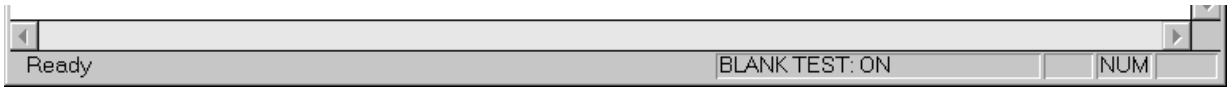
It is possible to display or hide the toolbar by using the apposite command of the VIEW menu.



FS3ASM Status Bar

The status bar contains information about the current line number, the position of the cursor, the blank check status (enabled/disabled), the Caps Lock and Num Lock status.

It is possible to display/hide the status bar by using the apposite command from the VIEW menu.



Managing and Printing Files

A new editing page, and then a new program, can be started with the command NEW (CTRL+N) from the FILE menu or clicking the apposite toolbar button. As default, the file name is Untitledx being x a progressive number according to the already existing files "Untitled".

- The file can be saved by means of the command SAVE (CTRL + S) or, with the possibility to modify its name and location, with the command SAVE AS …
- The file can be closed with the CLOSE command.
- To open an already existing file use the command OPEN… (CTRL + O) or the apposite toolbar button. This command allows to open the dialog box for the selection of the file to open.

To print the file the following commands are available:

- The PRINT… command (CTRL + P) allows you to print the current file.
- The command PRINT SETUP… allows to open a dialog box for the printer setup.
- PRINT PREVIEW allows you to display a program before printing it.

The command EXIT allows you to exit fromFS3ASM tool (ALT + F4).

Editing Commands

FS3ASM provides standard editing commands:

CUT (CTRL + X)	Removes the currently selected text and places it on the clipboard.
COPY (CTRL + C)	Makes a copy of the currently selected text and places the copy on the clipboard.
PASTE (CTRL + V)	Places a copy of the text currently on the clipboard at the currently selected location. The text remains on the clipboard.
DELETE (DEL)	Removes the current selected text.
UNDO (CTRL + Z)	Choose UNDO from the EDIT menu to undo the previous editing action.
FIND…	Searches a selected text.
FIND NEXT (F3)	Repeats the last search performed.
REPLACE…	Allows to search for and replace text items.
SELECT ALL	Selects every text item included in the document.
WORD WRAP	Wrapping text enables you to see all the text on the line, but it doesn't affect the way text appears when it is printed.

It is also possible to set fonts and tabs by selecting the following commands on the VIEW menu:

SET TAB STOPS… allows to specify the number of blank characters which form a single tab stop.

SET FONT… allows you to set the fonts.

SET PRINTER FONTS allows to open the dialog box for the printer fonts setting.

Machine Code Generation

To generate the machine code relative to the Assembler program of the current document, select the command ASSEMBLY from the TOOLS menu or click the apposite toolbar button. This command opens a DOS window in which are shown the program's compilation results. In case of correct compilation a file is generated containing the code in binary format and the following message appears:

```
Compiling assembler code...
0 error(s)
0 warning(0)
Assembler code  compilation done.
```

In case of errors, the list of errors is shown.

Device Programming

Before programming a ST52x301 device, ensure that: the chip is inserted correclty in the socket, the board has been linked to the parallel port and that the power supply is ON. Remind as well that the program has to be compiled first.

It is possible to choose if you want to perform the blank check before the downloading or by selecting BLANK CHECK on the TOOLS menu. The DOWNLOAD command on the TOOLS menu or the equivalent toolbar button launches the programming of the chip. At the end of the downloading the following messages can appear:

Successfully Download
The program has been correctly downloaded on the device's memory.

Error Opening File
The binary code file has not been found, compile again before downloading.

Invalid Bin File
The binary file is corrupt or it is not a file generated by the Compiler.

Error Writing Memory
You tried to unsuccessfully write on the device's memory. The chip could be broken or not well inserted.

Device not blank
In case the blank check is not enabled and the device was not erased before the programming.

Board not present
The board has not been properly connected or the power supply has not been enabled yet.

WinExec error code
Windows cannot start the download module. Try to close some currently active programs.

Internal Error
An internal error occurred, please contact STMicroelectronics - Fuzzy Logic B.U.

W3ASM Error List

argument "xxx" is not integer
The specified argument "xxx" is not an integer value. Only integer values can be used in Assembler commands.

argument "xxx" out of range
The argument "xxx" is out of the allowed range.

bad label string "name"
The label "name" is not a valid label. It may contain not allowed characters.

call to wrong output function
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

cannot access an output file
The code file cannot be accessed. Check the right of the destination directory or if the disk is full.

cannot access input file
The Assembler input file is corrupted or has been deleted or an Internal Error occurred.

cannot access temporary file
The temporary file used during code generation cannot be accessed: the disk may be full or the file is read-only or an Internal error occurred.

cannot open input file "name"
The Assembler input file "name" is corrupted or has been deleted or an Internal Error occurred.

cannot open output file "name"
The code file "name" cannot be opened. Check the right of the destination directory or if the disk is full.

cannot open temporary output file
The temporary file used during code generation cannot be opened: the disk may be full or the file is read only or an Internal error occurred.

error using input file
An error occurred reading Assembler source file: it may be corrupted or has been deleted or an Internal Error occurred.

error using output file
An error occurred writing the code file: the disk may be full or the file is read only or an Internal error occurred.

function "name" returned value "value"
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

fuzzy output not computed
The fuzzy instruction OUT is missing for the computation of the output fuzzy variable.

illegal label "lab_name" before command "com_name"
The label "lab_name" has been specified before IRQ or DATA commands or before a Fuzzy Instruction. Delete the label.

interrupt number "value" out of range
The specified interrupt number is not allowed. Specify a value between 0 and 3.

interrupt redefined for signal "number'
The interrupt "number" vector has been already defined. Check the interrupt vectors indexes in IRQ commands.

left vertex distance "number" out of range
The specified number is not in the range [0 , 255].

line "number" is too long
The line number "number" is more than 256 character

membership "number" out of range
The membership index "number" is not in the range [0 , 15]

membership "num_mbf" redefined for variable "num_var"
The membership "num_mbf" for variable "num_var" has been already defined. Check the MBF indexes in DATA instructions.

misplaced command "name" for ADM block
The command "name" not belonging to the allowed set for Antecedent Data Setting has been found. Check for syntax errors.

misplaced command "name" for ALU block
The command "name" not belonging to the allowed set for ALU operations has been found. Check for syntax errors.

misplaced command "name" for FUZZY block
The command "name" not belonging to the allowed set for Fuzzy computation has been found. Check for syntax errors.

misplaced command "name" for IRQ block
The command "name" not belonging to the allowed set for Interrupt management has been found. Check for syntax errors.

missing ADM definition. Bad Fuzzy block
Fuzzy commands have specified without the definition of Antecedent Memory data.

missing operand(s) for command "name"
One or more operands expected for the command "name" were not found. Complete the instruction with all the correct operands.

missing reference for label "name"
The label "name" has been used but not referenced inside the program. Check for syntax errors.

no more memory available
There is no enough memory to run Assembler. Try closing some open programs.

not enough fuzzy operands
AND/OR operator have been used loading only one value in the stack.

out of chip code space
The generated program is longer than the available chip memory space. Try optimising the program.

pending operands into fuzzy core
A value, previously loaded in the buffer with SKM instruction, has been left without using it.

redefinition for label "name"
The label "name" has been previously defined in the program.

right vertex distance "number" out of range
The specified number is not in the range [0 , 255]

temporary fuzzy core buffer is empty
A LDM instruction has been specified without using SKM instruction before.

temporary fuzzy core buffer is not empty
A SKM instruction has been specified twice without using a LDM instruction between for using the previously loaded value.

temporary fuzzy core stack is empty
CON or LDK or SKM instruction has been specified with the stack empty.

too many fuzzy antecedents
Too many antecedents term (more than 8) have been included in rule processing.

too many fuzzy operands
Too many operands have been specified in instructions for rule processing.

too many operands for command "name"
More than the expected operands were found with command "name". Check the correctness of the instruction deleting unnecessary operands.

variable "number" out of range
The variable index "number" is not in the range [0 , 3]

vertex "number" out of range
The specified number is not in the range [0 , 255]

undefined label "name"

The label "name" is used but not defined. Check for syntax errors.

unexpected end of source

The source code ended in a not correct way. Check if the source file is corrupted or for syntax errors.

unrecognised command "name"

The specified command "name" is not a valid command. Check for syntax errors.

unsupported function "name" for command "command"

Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

Appendix E
FULL - Fuzzy Logic Language

FULL (Fuzzy Logic Language) is a programming language oriented to the definition of Fuzzy control systems. A FULL program is composed by two fundamental parts: the declarations part to define the Fuzzy Variables Term Set, and the procedural part to define Fuzzy control Rules.

< FULL PROGRAM > ::= <DECLARATIONS><RULES>.

In order to define the Term Set, the language allows the following actions:

- associates a label to an Universe of Discourse;
- defines templates for the Membership Functions;
- defines modifiers for the Membership Functions by using expressions;
- defines a Variable specifying the name, the associated Universe and the Membership Functions composing the Term Set.

The set of the rules, having format IF ... THEN ..., defines the knowledge base to determinate the values of output Variables starting from the input Variables values. The antecedent part of the rules consists in a logic expression of fuzzy operators AND, OR and NOT. The expression terms are the logic premise. Each premise is defined by an IS relation between a Variable and one of its Membership Function eventually modified. The consequent part of the rules is a linguistic expression composed by consequence joint by the connective AND. A consequence is defined by an IS relation between a Variable and one of its Membership Functions.

FULL Language Elements

Token

Tokens are elements of source program that are not further reduced by Compiler in its components. In FULL language, tokens are classified in the following categories:

- white space;
- punctuation;
- operators;
- keywords;
- identifiers;
- real values and constants.

White space

White space characters are introduced in the program text in order to improve the readability. During the parsing of the program, the Compiler ignores the white spaces. The recognized white spaces are:

- space
- tab (escape \t)
- carriage-return (escape \r)
- linefeed (escape \n)
- newline (escape \r\n)
- vertical tab (escape \v)
- formfeed (escape \f)

Comments

A comment is a sequence closed between double quote (“) containing whatever combination of characters except the double quote itself. A comment can be inserted anywhere in the source program and the Compiler considers it as a white space.

Punctuation

The punctuation characters in FULL are used mainly to organize the program text. Actually they don’t specify any operation with the language elements. The punctuation characters are the following:

;	.	_	[	]
{	}	(	).	

Some punctuation characters are operator symbols too.

Operators

Operators are symbols that specify the operation to execute with the program objects. In the following, the FULL operators are listed, sorting them according to the priority.

£	(ALT+<0163> character) definition of independent variable;
[]	indexing of Membership Functions;
()	changing of priority in mathematical expressions;
+-	unary sign operator;
%^	module and power operators;
*/	multplicative operators;
+-	additive operators;
@	entry point;
,	sequencer;
=	definition operator.

All the operators are left associative.

Keywords

Keywords assume a particular meaning in FULL language and, for this reason, the Compiler manages them differently from the other words. The list of reserved FULL Keywords is the following:

AND	IF	OR	TIMES
AT	IS	POINTS	UNIVERSES
BEGIN	LAMBDA	POLYLINE	VARIABLES
CONTINUE	LESS	RENAME	VERY
END	MODIFIERS	SHAPES	WITH
FOR	NOT	THEN	

Identifiers

Identifiers are names assigned to universes, modifiers, forms, variables and terms in a FULLPROGRAM. It is not possible to use reserved keywords as identifiers. After being declared, the identifier can be used in the program text as the object that it represents.

The FULL language puts some limitations on the words used as identifiers. An identifier must start with a letter (upper or lower case) and it can be composed by letters, digits and underscores (_). Upper and lower case letters are considered different.

In the following, the identifiers' grammar is shown:

```
<IDENTIFIER>     ::= <LETTER><DIGITLETTER> |
                     <LETTER>.
<DIGITLETTER>    ::= <LETTER><DIGITLETTER> |
                     <DIGIT><DIGITLETTER> |
                     <LETTER>|
                     <DIGIT>.
<LETTER>         ::= <LETTER> |
                     _.
```

With <LETTER> we intend one of the following:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

With <DIGIT> we intend one of following:

0 1 2 3 4 5 6 7 8 9.

Constants

A constant in FULL is a decimal number with a sign. The constant is composed by an integer part, a decimal part and an exponent. The limits of the constant values depends on implementation. In any case, the FULL Compiler considers as equal constant values having the same first 9 significative digits.

In the following, the constants grammar is shown:

```
<CONSTANT>        ::= <INTEGERPART><SECONDPART> |
                      <INTEGERPART>.
<INTEGERPART>     ::= <SIGN><DIGITSEQUENCE> |
                      <DIGITSEQUENCE>.
<SIGN>            ::= +   |   -.
<DIGITSEQUENCE>   ::= <DIGIT><DIGITSEQUENCE> |
                      <DIGIT>.
<SECONDPART>      ::= . <DECIMALPART> |
                      <EXPONENT>.
<DECIMALPART>     ::= <DIGITSEQUENCE><EXPONENT> |
                      <DIGITSEQUENCE>.
<EXPONENT>        ::= e <INTEGERPART> |
                      E <INTEGERPART>.
```

Without explicit indication of sign, the positive sign is assumed by default.

Expressions

An expression is a sequence of operands and operators. An “operand” is the object managed by the operator. Operands in FULL are identifiers, constants, mathematical functions or expressions between parenthesis. Expressions are used in FULL to define modifiers and terms (Membership Functions) with continuous functions. Both modifiers and terms are real functions of a real variable. The FULL language asks the definition of an identifier for the real variable of the function dominion (independent variable). The identifier for the independent variable is declared with the operator £ or LAMBDA. The identifier is valid only in the expression defining the function and it is no longer valid after the end of expression.

In the following, the function grammar is shown:

```
<FUNCDEF>         ::= <INDEPENDENTVAR> . <EXPRESSION>.
<INDEPENDENTVAR>  ::= £ <IDINDEPVAR>|
                      LAMBDA <IDINDEPVAR>.
<IDINDEPVAR>      ::= <IDENTIFIER>.
```

The operators priority rules (see "Operators" paragraph) are used for the evaluations of the expression together to the left associativity.

```
<EXPRESSION>          ::= <EXPRESSION><OPADD><ADDENDUM>|
                          <ADDENDUM>.
<ADDENDUM>            ::= <ADDENDUM><OPMUL><OPERANDWITHSIGN> |
                          <OPERANDWITHSIGN>.
<OPERANDWITHSIGN>::= <CONSTANT>|
                          + <OPERAND>|
                          - <OPERAND>|
                          <OPERAND>.
<OPERAND>             ::= <IDINDEPVAR>|
                          <MATHFUNC>|
                          ( <EXPRESSION> ).
<MATHFUNC>            ::= <FUNCTOR> ( <EXPRESSION> ).
```

With <OPADD> we intend one of following operations:

+	sum operation;
-	subtraction operation.

With <OPMUL> we intend one of following operations:

*	multiplication operation ;
/	division operation;
%	module operation;
^	power operation.

<FUNCTOR> is one of the following symbols of mathematical function:

abs	absolute value of a real number;
acos	arcos of a real number in the interval [-1,1];
asin	arcsin of a real number in the interval [-1,1];
atan	arctan of a real number;
cos	cosine of a real number;
cosh	hyperbolic cosine of a real number;
exp	exponential function;
log	natural logarithm of a positive real number;
log10	decimal logarithm of a positive real number;
round	real number rounding;
sin	sine of a real number;
sinh	hyperbolic sine of a real number;
sqrt	square root of a not negative real number;
tan	tangent of a real number;
tanh	hyperbolic tangent of a real number;

The function symbols are not reserved words of the language.

Declarations

A “declaration” specifies the interpretation to be given to an identifier. The objects in FULL that can have a name are the universes of discourse, modifiers, forms, variables and membership functions. For this reason the declarations are divided in sessions called paragraph. Each paragraph starts with a keyword that indicates the kind of objects to be defined and ends with the start keyword of another paragraph or with the BEGIN keyword that indicates the start of the rule list. Paragraphs can have any order. A paragraph can appear more than once in the paragraph list. After the declaration, the introduced identifier is valid in the whole source program but the use is restricted to the syntactical and semantical rules of FULL language.

```
<DECLARATIONS>    ::= <PARAGRAPH><DECLARATIONS>  |
                      <PARAGRAPH>.

<PARAGRAPH>       ::= <UNIVERSES> |
                      <MODIFIERS> |
                      <FORMS> |
                      <VARIABLES>.
```

Universes

The “Universes” paragraph starts with the keyword UNIVERSES. It is possible to define whatever number of universes inside the paragraph. A universes declaration consists in the definition of a label for a real number interval.

```
<UNIVERSES>       ::= UNIVERSES <UNIVERSESLIST>.
<UNIVERSESLIST>   ::= <UNIVERSE><UNIVERSESLIST>  |
                      <UNIVERSE>.
<UNIVERSE>        ::= <IDENTIFIER> = <INTERVAL> ;.
<INTERVAL>        ::= [ <CONSTANT> , <CONSTANT> ].
```

The first constant in the interval definition must be lower than the second one. The universe identifiers can be used anywhere a universe of discourse specification is requested.

Modifiers

The “Modifiers “ paragraph is introduced by MODIFIER keyword. It is possible to define whatever number of modifiers inside the paragraph. A modifier declaration defines an association between an identifier and a real function of a real variable. This variable, called independent variable, assumes values in the interval [0,1]. The modifier function must assume values in interval [0,1]. During compilation, values external to the interval [0,1] will be cut to the interval extremes. The independent variable identifier is introduced with the notation £ or with the keyword LAMBDA (see "Expressions" paragraph).

```
<MODIFIERS>     ::= MODIFIERS <MODLIST>.
<MODLIST>       ::= <MODIFIER><MODLIST> |
                    <MODIFIER>.
<MODIFIER>      ::= <IDENTIFIER> = <FUNCDEF> ;.
```

FULL supplies three already defined modifiers: NOT, VERY e LESS defined as:

```
NOT = £ x . 1-x;
VERY = £ x . x^2;
LESS = £ x . sqrt(x);
```

It is possible to define again each modifier inside the program. A modifier can be applied to Variable terms specifying the modifier name. A modifier works in different ways if it is used during a Variable definition (see "Variables" paragraph) or if it is applied in a rule (see "Rules" paragraph).

Shapes

The “Shapes” paragraph is introduced by the keyword SHAPES. It is possible to define inside the paragraph any number of shapes. A shape declaration defines a link between an identifier and a normalized shape for a membership function. The shape is defined in a normalized Universe of Discourse, that is in the interval [0,1]. Each normalized shape has an “entry point” associated, that is a point in the interval [0,1] where the normalized shape is defined that represents the shape itself. When the shape is fixed into the Universe of Discourse of the Variable and it becomes a membership function, the entry point is used to give a position to the membership function so created (see "Variables" paragraph). As a default the entry point value is 0.

<FORMS> ::= SHAPES <FORMSLIST>.

<FORMSLIST> ::= <FORM><FORMSLIST> |
<FORM>.

<FORM> ::= <IDENTIFIER> = <SHAPEDEF> ; |
<IDENTIFIER>=<SHAPEDEF><ENTRYPOINT>;.

<ENTRYPOINT> ::= @ <CONSTANT>.

FULL supplies three different ways to define shapes (membership function in the normalized universe of discourse): by using points, multi-line, and continue.

<SHAPEDEF> ::= <POINTS> |
<MULTILINE> |
<CONTINUE>.

Using the definition by points (keyword POINTS), the Membership Function is defined by a list of couples of values. The first value represents a value in the Universe of Discourse. The second value represents the belief value of the Membership Function in the point of the Universe of Discourse specified with the first value of the couple. A “belief value” is a value in the interval [0,1]. The couples’ list must be sorted considering first the lower values of the universe of Discourse.

<POINTS> ::= POINTS { <COUPLESLIST> }.

<COUPLESLIST> ::= <COUPLE> , <COUPLESLIST> |
<COUPLE>.

<COUPLE> ::= <CONSTANT> / <BELIEF>.

<BELIEF> ::= <CONSTANT>.

Using the multi-line definition (keyword POLYLINE), the Membership Functions defined by a list of couples of values that specifies the segments extremes of the polyline representing the Membership Function. Each couple, with the exception of the first and the last one, defines the end of the previous segment and the start of the following one. The couples’ list must be sorted considering first the lower values of the universe of Discourse.

<MULTILINE> ::= POLYLINE { <COUPLESLIST> }.

In order to define continuous shapes (keyword CONTINUE), the Membership Function must be defined using a list of stroke. A “stroke” is a couple composed by an interval of the Universe of Discourse and a function of the independent variable. The identifier of the independent is specified just after the keyword CONTINUE and it is valid just for the definition of the Membership Function. The rules of this kind of definition are the following:

- the function is considered zero where it is not specified;
- negative function values are converted to zero;
- function values higher than 1 are converted to 1;
- if a is the first interval bound and b is the second one, then it must be b 3 a;
- if Ii and Ii+1 are two consecutive intervals, then it must be sup{Ii} £ inf{Ii+1 };
- if Ii and Ii+1 are two consecutive intervals and max{Ii} = min{Ii+1 } then it must be verified that Fi (max{Ii}) = Fi (min{Ii+1 }). The Compiler does not compute the function and issues a warning message.

```
<CONTINUE>      ::= CONTINUE <IDINDEPVAR> { <STROKESLIST> }.
<STROKESLIST>   ::= <STROKE> , <STROKESLIST> |
                    <STROKE>.
<STROKE>        ::= <RANGE> . <EXPRESSION>.
<RANGE>         ::= <OPEN>|
                    <OPENLEFT>|
                    <OPENRIGHT>|
                    <INTERVAL>.
<OPEN>          ::= ( <CONSTANT> , <CONSTANT> ).
<OPENLEFT>      ::= ( <CONSTANT> , <CONSTANT> ].
<OPENRIGHT>     ::= [ <CONSTANT> , <CONSTANT> ).
```

Variables

The variables paragraph is introduced by the keyword VARIABLES. Inside the paragraph, it is possible to define whatever number of Variables. A variable declaration defines a link between a variable identifier and a term set. A “term set” is a set of membership functions defined on a Universe of Discourse. If not specified, the Universe is the normalized one (that is an universe in [0,1] interval).

```
<VARIABLES>       ::= VARIABLES <VARIABLESLIST>.
<VARIABLESLIST>   ::= <VARIABLE><VARIABLESLIST>  |
                      <VARIABLE>.
<VARIABLE>        ::= <IDENTIFIER> = <UDD> . <TERMSET>; |
                      <IDENTIFIER> = <TERMSET>;.
<UDD>             ::= <IDENTIFIER> |
                      <INTERVAL>.
<TERMSET>         ::= { <TERMSLIST> }.
<TERMSLIST>       ::= <TERM> , <TERMSLIST> |
                      <TERM>.
```

An identifier and its definition must be supplied for each membership function belonging to the term set. The membership functions identifiers are valid only inside the term set definition. In other words, a membership function belonging to a term set can be accessed only by the variable identifier. A membership function in a term set can be defined directly inside the term set. In this case, the same rules of shape definition (see "Shapes" paragraph) are valid. Otherwise, it is possible to fix shapes or to modify already defined membership functions.

```
<TERM>            ::= <IDENTIFIER> = <TERMDEF> ;.
<TERMDEF>         ::= <SHAPEDEF> |
                      <FIXING> |
                      <MODIFIED> |
                      <REPEATFIXING>.
```

To fix a shape it is necessary to supply three parameters: the shape name, the position, in the Universe of Discourse of the Variable, of the shape’s entry point and the shape’s width. The shape’s width indicates a scale factor that allows to map the normalized universe to the Universe of the Discourse of the Variable.

```
<FIXING>          ::= <IDENTIFIER> ( <ENTRYPOS>,<WIDTH> ).
<ENTRYPOS>        ::= <CONSTANT>.
<WIDTH>           ::= <CONSTANT>.
```

It is possible to define membership functions using modifier with fixed shapes or with already defined membership functions in the term set. The modifiers can be used in cascade.

```
<MODIFIED>          ::= <MODIFIERSLIST><TOBEMODIFIED>|
                        <TOBEMODIFIED>.
<MODIFIERSLIST>::=  <MODIFY><MODIFIERSLIST>|
                        <MODIFY>.
<TOBEMODIFIED>      ::= <IDENTIFIER>|
                        <ELEMENT>|
                        <FIXING>.
<ELEMENT>           ::= <IDENTIFIER> [ <INTEGER> ].
<INTEGER>           ::= <DIGITSEQUENCE>.
<MODIFY>            ::= <IDENTIFIER>|
                        NOT  | VERY | LESS.
```

The repetition of shape fixing can be realized using FOR ... TIMES ... AT. In this way a membership function vector can be declared defining: the vector dimension, that is the number of repetitions of shape fixing; the fixing of the first element of the vector; the distance between the entry points of the following shapes. The compiler issues an error message if the entry point positions so computed are out of the Universe of Discourse boundaries. Using the RENAME construct it is possible to rename the membership functions of the vector. This construct accepts a list of identifiers. The association between the membership functions of the vector and the identifiers is done by the position of the identifier in the list. If a membership function has not to be renamed, the character "_" (underscore) can be used. It is not necessary that the identifier list has the same length of the vector: the exceeding identifiers are ignored. The not renamed membership functions can be accessed indexing the vector name. The vector option base is 1.

```
<REPEATFIXING>      ::= FOR <NUMBTIMES> TIMES
                        <FIXINGMODIFIED>
                        AT <DISTANCE>
                        [ RENAME <MBSIDLIST> ].
<NUMBTIMES>         ::= <INTEGER>.
<FIXINGMODIFIED>    ::= <MODIFIERSLIST><FIXING> |
                        <FIXING>.
<DISTANCE>          ::= <CONSTANT>.
<MBSIDLIST>         ::= <IDMBS> , <MBSIDLIST> |
                        <IDMBS>.
<IDMBS>             ::= <IDENTIFIER> |
                        _.
```

Rules

The procedural part of FULL language starts with the keyword BEGIN and ends with the keyword END. The keyword END closes also the source FULLPROGRAM. The procedural part is composed by a set of rules. A “weight” can be associated to each rule using the WITH construct. As a default, the weight associated is 1. To increase the importance of a rule, the weight associated to the rule must be higher than 1; otherwise, to decrease the importance of the rule, the weight must be lower than 1.

<RULES> ::= BEGIN <RULESLIST> END.

<RULESLIST> ::= <RULE><RULESLIST>|
<RULE>.

<RULE> ::= <RULEFORM> [WITH <WEIGHT>] ; |
<RULEFORM>.

<WEIGHT> ::= <CONSTANT>.

A rule is an inference of the kind IF ... THEN ... composed by an antecedent part and a consequent part.

<RULEFORM> ::= IF <ANTECEDENT> THEN <CONSEQUENT>.

The antecedent part is a logic expression that uses the fuzzy logic operators AND & OR. In fuzzy logic expressions, AND operator has higher priority than OR operator. The use of parenthesis can alter the order in which the operators are evaluated.

<ANTECEDENT> ::= <ANTECEDENT> OR <ALTERNATIVE> |
<ALTERNATIVE>.

<ALTERNATIVE> ::= <ALTERNATIVE> AND <CONTRIBUTE> |
<CONTRIBUTE>.

<CONTRIBUTE> ::= (<ANTECEDENT>)|
<PREMISE>.

The consequent part contains a list of consequences of the inference, grouped with the AND connector. The AND connector has only a syntactical role and a totally different meaning of the AND fuzzy logic operator.

<CONSEQUENT> ::= <CONSEQUENCE> AND <CONSEQUENCE> |
<CONSEQUENCE>.

“Premises” and “Consequences” are unary predicates of the kind M(x) where M is a Membership Function defined in the Universe of Discourse of the variable x. In order to define this kind of predicates, FULL supplies the syntactical connective IS. So the unary predicate can be syntactically described as “x IS M”.

As a consequence, M can be substituted by a membership function name, by an element of a membership function’s vector, or by a crisp value in the universe of Discourse of Variables x.

<CONSEQUENCE> ::= <IDENTIFIER> IS <OUTPUT>.
<OUTPUT> ::= <IDENTIFIER>|
<ELEMENT> |
<CONSTANT>.

In a premise, M can be substituted by a membership function name, by an element of a membership functions vector, or a modified membership function of the Variable x.

<PREMISE> ::= <IDENTIFIER> IS <MODIFIEDMBS>.
<MODIFIEDMBS> ::= <MODIFY><MODIFIEDMBS>|
<IDENTIFIER>|
<ELEMENT>.

In FULL a “consequence” can be considered as a “premise” of another rule. In such a case, the rule interpretation assigns to the “premise” p an a value equal to the max J value of the rules having p as “consequence”.

FULL program example

In this paragraph, all the functionalities given by FULL language are described by means of an example program. This program has not a particular semantic, except the necessary one to explain the language.

The first three shapes are defined: the building of variables term sets is based on them. The first shape is a triangle having the vertex with max belief in the center of normalized universe; the vertex is also the entry point of the shape. The second shape is a crisp value in the middle of the normalized universe. The third is a parabola equation having vertex with max belief in the middle of the normalized universe a minimum belief in the universe extremes.

```
SHAPES
triangle = POLYLINE {0/0, 0.5/1, 1/0} @ 0.5;
crisp = POINTS {0.5/1} @ 0.5;
parabola = CONTINUE x { [0,1]. -4*(x^2) + 4*x };
```

In addition, a modifier that transforms the triangular membership functions in gaussian membership functions is defined. The equation for the modifier is obtained making repeated modifications to the triangle corresponding to “NOT VERY NOT VERY triangle”.

```
MODIFIERS
gauss = LAMBDA y . 1-(1-y^2)(1-y^2);
```

The control to be defined uses three variables: temperature, pressure and out. After defining the universes, the term set declaration must be done.

The variable “temperature”, defined in the universe “degrees”, has a term set composed by 5 membership functions. The membership functions “verylow” and “veryhigh” are defined directly inside the term set definition as polyline. They are triangles in the extremes of the universe of discourse. The other membership functions are defined using a multiple fixing of the shape triangle. the first element of the vector “middle” has the vertex, that is the entry point of the “triangle” shape, in the position 25 of the universe “degrees” and the base of triangle width equal to 50. The second and the third element of the vector have vertex respectively in 50 and 75.

The variable “pressure”, defined in the universe “atmosphere”, has a term set having three membership functions. The membership function ”low” and “high” are gaussians obtained with the modifier “gauss” of the shape “triangle”. Notice that the vertex of “low” and “high” are on the extreme of the universe. The membership functions “medium” is a fixing of the shape “parabola” with the vertex in the position 10 in the universe of discourse “atmosphere”. The variable “out” has a term set composed by a vector of 10 fixing of the shape “crisp”.

```
UNIVERSES
degrees= [0,100];
atmosphere = [1, 20];
VARIABLES
temperature = degrees .
        {
       verylow = POLYLINE { 0/1, 25/0 };
       veryhigh = POLYLINE { 75/0, 100/1 };
       middle = FOR 3 TIMES triangle(25,50)
                       AT 25 RENAME low, medium, high;
       }
pressure = atmosphere .
       {
       low = gauss triangle(0,20);
       medium = parabola(0,20);
       high = gauss triangle(20,20);
       }
out = [1,10] .
       {
       out = FOR 10 TIMES crisp(1,2) AT 1 RENAME off;
       }
```

In the following 3 rules, defined on the declared variables, are showed.

```
BEGIN
IF temperature IS verylow THEN out IS off WITH 2;
IF temperature IS VERY low AND (pressure IS low OR pressure IS medium)
       THEN out IS out[1] AND out IS out[2];
IF temperature IS NOT verylow AND out IS off THEN out IS 3.5;
END
```

FULL Language Grammar

In the following the whole FULL grammar is showed.

```
<FULLPROGRAM>      ::= <DECLARATIONS><RULES>.
<DECLARATIONS>     ::= <PARAGRAPH><DECLARATIONS>  |
                       <PARAGRAPH>.
<PARAGRAPH>        ::= <UNIVERSES>|
                       <MODIFIERS>|
                       <FORMS>|
                       <VARIABLES>.
<UNIVERSES>        ::= UNIVERSES <UNIVERSESLIST>.
<UNIVERSESLIST>    ::= <UNIVERSE><UNIVERSESLIST>  |
                       <UNIVERSE>.
<UNIVERSE>         ::= <IDENTIFIER> = <INTERVAL> ;.
<INTERVAL>         ::= [ <CONSTANT> , <CONSTANT> ].
<MODIFIERS>        ::= MODIFIERS <MODLIST>.
<MODLIST>          ::= <MODIFIER><MODLIST>  |
                       <MODIFIER>.
<MODIFIER>         ::= <IDENTIFIER> = <FUNCDEF> ;.
<FORMS>            ::= SHAPES <FORMSLIST>.
<FORMSLIST>        ::= <FORM><FORMSLIST>  |
                       <FORM>.
<FORM>             ::= <IDENTIFIER> = <SHAPEDEF> ;  |
                       <IDENTIFIER> = <SHAPEDEF><ENTRYPOINT>;.
<ENTRYPOINT>       ::= @ <CONSTANT>.
<SHAPEDEF>         ::= <POINTS>|
                       <MULTILINE>|
                       <CONTINUE>.
<POINTS>           ::= POINTS { <COUPLESLIST> }.
<COUPLESLIST>      ::= <COUPLE> , <COUPLESLIST> |
                       <COUPLE>.
<COUPLE>           ::= <CONSTANT> / <BELIEF>.
<BELIEF>           ::= <CONSTANT>.
<MULTILINE>        ::= POLYLINE { <COUPLESLIST> }.
<CONTINUE>         ::= CONTINUE <IDINDEPVAR> {<STROKESLIST>}.
<STROKESLIST>      ::= <STROKE> , <STROKESLIST> |
                       <STROKE>.
<STROKE>           ::= <RANGE> . <EXPRESSION>.
```

```
<RANGE>          ::= <OPEN>|
                     <OPENLEFT>|
                     <OPENRIGHT>|
                     <INTERVAL>.
<OPEN>           ::= ( <CONSTANT> , <CONSTANT> ).
<OPENLEFT>       ::= ( <CONSTANT> , <CONSTANT> ].
<OPENRIGHT>      ::= [ <CONSTANT> , <CONSTANT> ).
<VARIABLES>      ::= VARIABLES <VARIABLESLIST>.
<VARIABLESLIST>  ::= <VARIABLE><VARIABLESLIST>  |
                     <VARIABLE>.
<VARIABLE>       ::= <IDENTIFIER> = <UDD> . <TERMSET>;|
                     <IDENTIFIER> = <TERMSET>;.
<UDD>            ::= <IDENTIFIER>|
                     <UNIVERSE>.
<TERMSET>        ::= { <TERMSLIST> }.
<TERMSLIST>      ::= <TERM> , <TERMSLIST> |
                     <TERM>.
<TERM>           ::= <IDENTIFIER> = <TERMDEF> ;.
<TERMDEF>        ::= <SHAPEDEF>|
                     <FIXING>|
                     <MODIFIED>|
                     <REPEATFIXING>.
<FIXING>         ::=<IDENTIFIER>(<ENTRYPOS>,<WIDTH>).
<ENTRYPOS>       ::= <CONSTANT>.
<WIDTH>          ::= <CONSTANT>.
<MODIFIED>       ::= <MODIFIERSLIST><TOBEMODIFIED>.
<MODIFIERSLIST>  ::= <MODIFY><MODIFIERSLIST>|
                     <MODIFY>.
<TOBEMODIFIED>   ::= <IDENTIFIER>|
                     <FIXING>.
<MODIFY>         ::= <IDENTIFIER>|
                     <ELEMENT>|
                     NOT  | VERY | LESS.
<ELEMENT>        ::= <IDENTIFIER> [ <INTEGER> ].
<INTEGER>        ::= <DIGITSEQUENCE>.
<REPEATFIXING>   ::= FOR <NUMBTIMES> TIMES
                     <FIXINGMODIFIED>
                     AT <DISTANCE>
                     [ RENAME <MBSIDLIST> ].
<NUMBTIMES>      ::= <INTEGER>.
```

```
<FIXINGMODIFIED>   ::= <MODIFIERSLIST><FIXING>|
                       <FIXING>.
<DISTANCE>         ::= <CONSTANT>.
<MBSIDLIST>        ::= <IDMBS> , <MBSIDLIST>|
                       <IDMBS>.
<IDMBS>            ::= <IDENTIFIER>|
                       _.
<RULES>            ::= BEGIN <RULESLIST> END.
<RULESLIST>        ::= <RULE><RULESLIST>|
                       <RULE>.
<RULE>             ::= <RULEFORM> [ WITH <WEIGHT> ] ;|
                       <RULEFORM>.
<WEIGHT>           ::= <CONSTANT>.
<RULEFORM>         ::= IF <ANTECEDENT> THEN <CONSEQUENT>.
<CONSEQUENT>       ::= <CONSEQUENCE> AND <CONSEQUENCE>  |
                       <CONSEQUENCE>.
<CONSEQUENCE>      ::= <IDENTIFIER> IS <OUTPUT>.
<OUTPUT>           ::= <IDENTIFIER>|
                       <ELEMENT>|
                       <CONSTANT>.
<ANTECEDENT>       ::= <ANTECEDENT> OR <ALTERNATIVE>  |
                       <ALTERNATIVE>.
<ALTERNATIVE>      ::= <ALTERNATIVE> AND <CONTRIBUTE>  |
                       <CONTRIBUTE>.
<CONTRIBUTE>       ::= (<ANTECEDENT>)|
                       <PREMISE>.
<PREMISE>          ::= <IDENTIFIER> IS <MODIFIEDMBS>.
<MODIFIEDMBS>      ::= <MODIFY><MODIFIEDMBS>|
                       <IDENTIFIER>|
                       <ELEMENT>.
<FUNCDEF>          ::= <INDEPENDENTVAR> . <EXPRESSION>.
<INDEPENDENTVAR>   ::= £ <IDINDEPVAR>|
                       LAMBDA <IDINDEPVAR>.
<IDINDEPVAR>       ::= <IDENTIFIER>.
<EXPRESSION>       ::= <EXPRESSION><OPADD><ADDENDUM>|
                       <ADDENDUM>.
<ADDENDUM>         ::= <ADDENDUM><OPMUL><OPERANDWITHSIGN>|
                       <OPERANDWITHSIGN>.
```

```
<OPERANDWITHSIGN>::= <CONSTANT>|
                     + <OPERAND>|
                     - <OPERAND>|
                     <OPERAND>.
<OPERAND>        ::= <IDINDEPVAR>|
                     <MATHFUNC>|
                     ( <EXPRESSION> ).
<MATHFUNC>       ::= <FUNCTOR> ( <EXPRESSION> ).
<IDENTIFIER>     ::= <LETTER><DIGITLETTER>|
                     <LETTER>.
<DIGITLETTER>    ::= <LETTERS><DIGITLETTER>|
                     <DIGIT><DIGITLETTER>|
                     <LETTERS>|
                     <DIGIT>.
<LETTERS>        ::= <LETTER>|
                     _.
<CONSTANT>       ::= <INTEGERPART><SECONDPART>|
                     <INTEGERPART>.
<INTEGERPART>    ::= <SIGN><DIGITSEQUENCE>|
                     <DIGITSEQUENCE>.
<SIGN>           ::= +   |   -.
<DIGITSEQUENCE>  ::= <DIGIT><DIGITSEQUENCE>|
                     <DIGIT>.
<SECONDPART>     ::= . <DECIMALPART>|
                     <EXPONENT>.
<DECIMALPART>    ::= <DIGITSEQUENCE><EXPONENT>|
                     <DIGITSEQUENCE>.
<EXPONENT>       ::= e <INTEGERPART>|
                     E <INTEGERPART>.
```

```
" FULL source from 'SAMPLE' project by Fuzzy Studio 2.0 "
VARIABLES
distance = [0,100] .{
       medium = POLYLINE {24.7058824/0, 49.8039216/1, 75.2941176/0};
       low = POLYLINE {0/1, 24.7058824/1, 49.8039216/0};
       high = POLYLINE {49.8039216/0, 75.2941176/1, 100/1};
};
speed = [0,250] .{
       medium = POLYLINE {61.7647059/0, 124.509804/1, 188.235294/0};
       low = POLYLINE {0/1, 0.980392157/1, 123.529412/0};
       high = POLYLINE {124.509804/0, 249.019608/1, 250/1};
};
brakes_power = [0,8] .{
       one = POINTS {1.03529412/1};
       two = POINTS {2.00784314/1};
       five = POINTS {5.05098039/1};
       four = POINTS {4.01568627/1};
       three = POINTS {3.01176471/1};
       six = POINTS {6.02352941/1};
       seven = POINTS {7.09019608/1};
       eight = POINTS {8/1};
       zero = POINTS {0/1};
};
BEGIN "9 rules defined"
IF distance IS low AND speed IS high THEN brakes_power IS eight ;
IF distance IS low AND speed IS medium THEN brakes_power IS four ;
IF distance IS low AND speed IS low THEN brakes_power IS two ;
IF distance IS medium AND speed IS low THEN brakes_power IS 1 ;
IF distance IS medium AND speed IS medium THEN brakes_power IS 4 ;
IF distance IS medium AND speed IS high THEN brakes_power IS six ;
IF distance IS high AND speed IS low THEN brakes_power IS zero ;
IF distance IS high AND speed IS medium THEN brakes_power IS two ;
IF distance IS high AND speed IS high THEN brakes_power IS four;
END
```

Full Product Information at http://www.st.com