FUZZYSTUDIO™ 4.1

USER MANUAL

NOVEMBER 2000

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 STMicroeletrconics or as otherwise by law is an infringement 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) sublicence, 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 DEVICE OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF STMicroelectronics.

As used herein:

1. Life support devices or systems are those which (a) are intended 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 reasonably expected 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 responsably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

Table of Contents

ABOUT THIS MANUAL

This manual is designed to help you to get familiar with FUZZYSTUDIO™4.1 Software Development Environment for the ST52 family of Fuzzy Microcontrollers.

It provides an overview of the ST52 programming tools included in FUZZYSTUDIO™4.1 environment and of the additional tools provided with FUZZYSTUDIO™4.1 Kit. It introduces you in the new “Visual” programming approach used in FUZZYSTUDIO™4.1 to get quick and smart results in applications development.

You can find the contents of this manual also in the Online help included in FUZZYSTUDIO™4.1, accessible from the main menu.

Manual Contents

The FUZZYSTUDIO™4.1 Manual is organized as follows:

1: Getting Started
The Getting Started chapter provides you with general information about FUZZYSTUDIO™4.1 including the key features and what’s new in this release. It also provides you with a guide to the installation of the program and on the user interface. This chapter also includes reference to available documents and literature.

2: FUZZYSTUDIO™4.1 Overview
This part gives an overview of FUZZYSTUDIO™4.1 and an introduction to the programming approach for the development of your applications.

3: Project Management
This section provides an overview of the major elements of the FUZZYSTUDIO™4.1 Main Window, such as menus, toolbar and status bar. The Project Window allows to access any part of the project and define user program procedures.

4: Initial Settings
This chapter describes how to initialize the device. Variable initialization and peripheral configuration are described.

5: Blocks Editor
The Blocks Editor is the main tool used in the environment to design the block diagram of the program parts. In this chapter you will learn the standard editing commands related to blocks and links.

6: Fuzzy Block
The environment to define a Fuzzy System is described in this chapter. In this chapter you will learn how to edit the fuzzy system, define fuzzy variables, draw membership functions and how to write fuzzy rules.

7: Arithmetic Block
The Arithmetic Block allows to carry out the arithmetic and logic instructions of the device. In this chapter you will learn how to use the Arithmetic Block editor, the instructions set syntax and how to write the program lines by using ‘C’ code.

8: Assembler Block
The Assembler Block allows to program routines at low level. In this section you will learn how to use the Assembler Block editor, the instructions set syntax and how to write the program lines.

9: Conditional Block
The Conditional Block allows to modify the logic flow of the program according to a specified condition operating on the Global and Predefined Variables. In this chapter you will learn how to use the Conditional Block editor and how to write the conditions.

10: Blocks for Peripherals Management
In this chapter you will learn how to use the Send, Receive and Peripheral blocks for the management of peripherals' operations.

Chapter 11: Interrupts Related Blocks
This section explains how to use the Interrupts related blocks supplied with FUZZYSTUDIO™4.1 to manage the interrupts.

Chapter 12: Other Blocks
In this chapter you are provided with information on other blocks used for special functions. They are used to implement some important features of the microcontroller or to improve the readability of the block diagram.

Chapter 13: Compiler
This chapter describes how to compile the project and get the code to be loaded in the device.

Chapter 14: Debugger
This chapter describes the use of the Debugger, the tool allowing to test the developed program by means of the chip's simulation.

Chapter 15: Device Programming
In this chapter the device programming procedure and the Programming Board are pointed out.

Appendix A: Features dependent on the target device
In this appendix the development environment's features are described taking into account the differences among the various devices of the ST52 family.

Appendix B: ST52x420 Programmer Board
This appendix supplies the description of the ST52x420 Programmer Board.

Appendix C: FSASM Assembler programming Tool
This part describes the FSAsmtool, supplied with FUZZYSTUDIO™4.1 kit, to program the de-vice in Assembler language.

Appendix D: Fuzzy Logic Introduction
This part is useful to learn Fuzzy Logic basic concepts.

Appendix E: FULL (FUzzy Logic Language)
This Appendix describes the FULL description language syntax. FULL is used to export/im-port fuzzy system data among STMicroelectronics proprietary Fuzzy Logic S/W tools.

BEFORE YOU BEGIN

General Conventions

Before you start using FUZZYSTUDIO™4.1 , it is important to understand the terms and notational conventions used in this documentation.

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 indicates the names of commands and menus commands of FUZZYSTUDIO™4.1.

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™4.1 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

Aplus sign (+) used between two keys’ 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 Oand release it. This is the shortcut to choose the File OPEN command.

1 - WELCOME TO FUZZYSTUDIO™4.1

FUZZYSTUDIO™4.1 represents the new way to program low-end microcontrollers. In addition it allows the design of Fuzzy Logic Control Systems. The Windows-based user interface supplies a very simple to understand and easy-to-use environment to quickly develop applications with all the ST52 family of fuzzy microcontrollers.

Key Features

The FUZZYSTUDIO™4.1 environment is context sensitive according to the selected target device. The New Project dialog-box, appearing at the beginning, allows to choose one of the ST52 family microcontrollers from the list of the available ones.

The FUZZYSTUDIO™4.1 *Visual* approach consists of a simple and intuitive graphical support, composed by some *wizards* and editing environments. It is based on *Blocks*, each representing a microcontroller's function to be programmed. The blocks are interconnected by means of links in order to establish the block diagram of the program. Each block type has an associated editor to write the instructions or set the operations to be performed by the microcontroller. The *Blocks Editor* is the tool to establish the block diagram and it is used in most of the parts of the environment.

The whole project is organized in the *Project Window* as a tree-view, that allows to access to the whole part of the project. From the tree-nodes it is possible to open the editors for the device configuration or variables definition and the block editors to establish the program parts like *Main Program*, procedures and interrupt routines. Each single block can also be reached from the Project Window.

The *Peripherals Configuration Editor* is composed by some dialog-boxes allowing the specification of the peripherals' functionality without writing any code line but just clicking with the mouse on guided options. Taking into consideration that usually, to configure the peripherals, it is necessary to program each single bit of several registers, keeping continuously the data-sheet beneath the eyes, with the new methodology it is possible not only to get the configuration faster, but it is also possible to avoid errors programming not admissible configurations.

FUZZYSTUDIO™4.1 supplies the editors for defining *variables* and initializing *data tables*. The first allows to associate symbolic names to memory locations and define the variable types. The variable types available in the actual version are *Byte*, *Signed Byte*, *Word* and *Signed Word* . According to the variable type, the *Compiler* translates the instructions in such a way to manage automatically the different variable types. The second editor allows to fill data tables with constant values; this can be used inside the program as look-up tables.

From the *Project Window* it is possible to define *new user procedures*. The standard procedures already available are the *Main Program* and the *Interrupt Service Routines*. Accessing to these procedures (both standard and user defined) the associated Blocks Editor opens, allowing the definition of the program.

The Compiler generates the object code files and the machine code to be loaded in the device target. The first object code generated is expressed in a representation language, called *FSCODE*, that reports, in a listing file easy to be analyzed by the programmer, the instructions and settings fixed with the Blocks Editors. The FSCODE language syntax is very close to the 'C' language one: some restrictions have been fixed because not all 'C' functionalities can be supported by ST52 devices. From the FSCODE list file, the Assembler code is generated and from this, the final machine code is obtained.

The machine code is finally loaded in the device by the Programmer tool that allows to transfer data through the parallel port of the Personal Computer to the *Programming board* supplied with FUZZYSTUDIO™4.1 kit.

Before physical implementation, it is possible to test the program by using the *Debugger* tool. This allows to simulate the device in all its parts, peripherals and interrupts included. *Program source* can be examined step-by-step, *variables* and *signals* can be observed in text format or plotted in a graphical window working like an oscilloscope.

Another tool is supplied with FUZZYSTUDIO™4.1 kit: *FSAsm*. It allows to develop programs for ST52 family in Assembler, to generate the machine code and load the code in the device.

INSTALLATION AND CONFIGURATION

System Requirements

Before you install FUZZYSTUDIO™4, make sure you have all the hardware and software you need to run the program:

- Intel type 80486 processor or higher.
- 32 MBytes RAM memory, 64 Mbytes are recommended
- Hard Disk with at least 10 MBytes of free space.
- VGA or higher graphics card. 1024x768 resolution is recommended.
- Mouse.
- Windows 95/ 98/ NT.

Installing FUZZYSTUDIO™4

Your first step is to use the Setup program to install FUZZYSTUDIO™4 on your hard disk. Be sure to start Microsoft Windows before to install FUZZYSTUDIO™4.

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 Start menu.

3. On the Run text box, type: **A:\SETUP** then choose OK or press Enter.

4. Follow the instructions on the screen.

5. You can choose the installation directory, if different from the default one or you can also choose the program folder where to add the program icons.

7. After copying the necessary files, the setup program automatically generates the Program Folder and the icon to start the program.

Starting to Use

After you have installed FUZZYSTUDIO™4, you can use the application.

To start FUZZYSTUDIO™4:

1. Open the FUZZYSTUDIO™4 Program Folder in the Start menu.

2. Click over the FUZZYSTUDIO™4 icon to run the program.

User Interface

This chapter provides basic skills on the use of FUZZYSTUDIO™4 and explains the items you see on the screen. You can learn how to choose commands and how to use FUZZYSTUDIO™4 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™4 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 command that you can use in 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™4 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:

- To display a menu that contains the command you want, click the menu name in the menu bar. Click the command name. You can also point to the menu, drag to the command you want, and then release the mouse button.
- If the command displays a dialog box, specify the information you need.
- 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:

- To activate the menu bar, press ALT or F10. The FILE menu appears selected, indicating that the menu bar is active.
- 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. 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™4 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™4 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 dialog box.

When you are working with FUZZYSTUDIO™4, 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™4 displays the Help topic for the selected command or window item in the HELP window.

Available Documents

You can find documents that contain answers to many of your technical questions or problems on the use of ST52 family products from STMicroeletronics Web site at the following address:

http://www.st.com/stoline/products/support

You can read, print, or download datasheets, application notes, product presentations and information on other Fuzzy Logic Software products, such as the updated versions of FUZZYSTUDIO™4 and the patches.

2 - FUZZYSTUDIO™ 4.1 OVERVIEW

FUZZYSTUDIO™4.1 is the development system that allows to program ST52 family of fuzzy microcontrollers. This high level tool helps you to:

- Develop applications without Assembler programming
- Verify a developed project with the Debugger tool
- Program the microcontroller through the programming board supplied with FUZZYSTUDIO™4.1 kit

FUZZYSTUDIO™4.1 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 block-diagram of your project by inserting the appropriate blocks. Each block you insert is already designed for a definite type of functionality that can be programmed either in a graphic way or with high level instructions. 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™4.1 is equipped with tools that allow the machine code generation and the debugging of the program. It is characterized by the following main functionalities:

- Fuzzy programming
- Arithmetic programming
- Program's block-diagram definition
- Peripheral configuration and activation
- Programming and activation of Interrupts and associated routines
- Procedures management
- Machine code generation
- Full chip emulation in graphic environment
- Device EPROM memory programming
- Possibility to program at low level using ST52 Assembler.

All the functionalities related to the program development are grouped in the Project Window, which allows easy access to all the program editors. So we have:

Main Program: to access all the blocks and folders in the main routine.

Interrupts: to access each interrupt routine and the blocks contained in it.

Procedures: to define new user procedures and access to the blocks contained in them.

Peripherals: to access the peripherals configuration wizards.

Variables: to define the Global Variables name and type.

Tables: to initialize the name and the associated constants and vectors values.

The editors are contained in the blocks. FUZZYSTUDIO™4.1 main blocks are:

Icon	Block Name	Description
	Start Block	Starting point of the program or procedure
	Fuzzy Block	Allows to performs the fuzzy functions
	Arithmetic Block	Carries out arithmetic and logic operations
	Assembler Block	Allows to insert arithmetic instructions in Assembler
	Conditional Block	Modifies the program flow in relation to user-defined conditions
	Interrupts Disable Block	Disables globally the interrupts
	Interrupts Enable Block	Enables globally the interrupts
	Interrupts Reset	Resets all pending interrupts
	Interrupts Mask Block	Enables selectively interrupts
	Interrupts Priority Block	Manages interrupts priority
	Peripherals Blocks	Group of blocks that 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
	Call Block	Calls user-defined procedures
	Wait Block	Stops the program until the first interrupt signal
	Halt Block	Puts the device in Halt mode
	Restart Block	Restarts the program
	Return Block	Ends the user-defined procedures
	IRQ Block	Starts an interrupt request service routine
	RETI Block	Ends an interrupt request service routine
	Folder Block	Allows to group blocks
	Links	Connect two different parts of the program

Thanks to these blocks it is possible to realize the desired project by simply drawing the program block-diagram and fixing each block function.Many of the blocks listed above have not an associated editor, because they represent an action to be performed.

Programming Approach

The FUZZYSTUDIO™4.1 Visual programming approach has been designed in order to help you in the development of a project. These are the main phases that you have to follow:

- **Configure the Peripherals and device's functionalities**

  The peripherals' configuration is obtained by means of the property-sheet recalled from the Project window. You can choose the peripherals' working mode by clicking on the relative check-boxes. The configurations that 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 use the relative Peripherals Block in the point of the program where you need the peripheral working.

- **Define the Global Variables**

  You can define the Global Variables' name and type through the apposite window recalled from the Project Window.

- **Create the Data Tables**

  If look-up tables or constants are needed in the program, use the Tables editor by opening it from the Project Window.

- **Create the Block-Diagram**

  It is possible to design a program, composed by a main program and by procedures directly defining the program's block-diagram. After opening the main Program window, or a procedure's window, to draw the program's block-diagram, you must choose the type of block you want to insert by clicking on the relative the Blocks Editor toolbar button, 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 the associated editor of each block to program the action to be performed**

  Each block you insert in the flow-chart, corresponds to a suited editing environment that can be opened by double-clicking on it (except for the blocks that have not an associated editor). For example:

  The editing tool of the *Arithmetic Block* is a text editor that allows to write the arithmetic operations to be performed by the processor. It is possible to insert conditional statements among the arithmetic instructions inside an Arithmetic Block. Refer to further paragraphs for the description of the available operations and their grammar.

  A double-click on the *Fuzzy Block* allows to open the *Fuzzy System Editor*, the environment to define Fuzzy Variables, the Membership Functions and 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 graphical 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 block-diagram routine.

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

- **Define the interrupts service routines and procedures.**

  To be able to define the service routines or procedures, in terms of instructions or commands, you work as in the main program, but inside the client area related to each interrupt and procedure, 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, the Output Window opens 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 the FSCODE language; an Assembler code file; a file containing information used by the Debugger; and the code to be loaded in the target device memory. In case of errors in the compilation, you have to correct and compile your program again. Code lines that have generated errors can be easily reached by double-clicking over the error message in the Output Window.

- **Use the Debugger to validate the program.**

  After the compilation has been successfully 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 half clock cycle then the graphics that it is able to produce are in function of the time. You can select the signals and/or the registers to visualize and choose 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 behavior of the device according to the program. You can also supply the input signals set to the chip to simulate external connections to the input pins defining them easily by using the Stimulus Editor. The registers values and the status of the selected signals can be visualized in a text format in the Watch window and others or graphically plotted in the Plot window. If the results of the simulation are not satisfying, appropriate changes have to be performed to the current program stored in memory.

- **Program the device with the board supplied with FUZZYSTUDIO™4 Kit.**

  The final step is to program the target device memory by means of the programming functions and the appropriate electronic board connected to the PC through the parallel port.

- **Test the real application.**

  Now you can insert the programmed chip in your own application and test it.

3 - PROJECT MANAGEMENT

This chapter describes the following topics:

- FUZZYSTUDIO™4.1 Main Windows
- FS4 project files management
- FS4 project management by means of the Project Window

The FUZZYSTUDIO™4.1 Main Window

This section provides an overview of the major elements of the FUZZYSTUDIO™4.1 Main Window, such as menus, toolbar and status bar.

The Main Window appears as soon as the program is started. The available menu commands and the enabled toolbar buttons are useful to create a new project or to load an existing project file. Commands to show/hide toolbar and status bar are also available. The Help commands are always available.

The FUZZYSTUDIO™4.1 Main Window application menus

The following menu items are available in the Main Window:

File Contains commands to create a new project, to open an existing one or to change the printer and printing options. The list of recently used files is also provided to access quickly to the projects.

View Contains commands to show or hide the Main Window toolbar and Status bar.

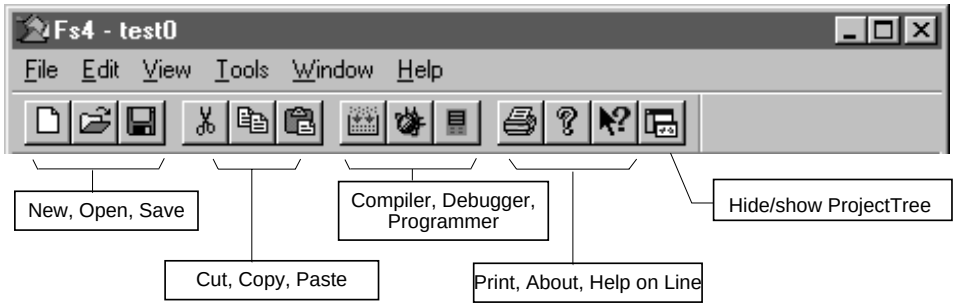
Help Contains commands to access and use the help.

Fig. 3.1 - Main Window

The FUZZYSTUDIO™4.1 Main Window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar.

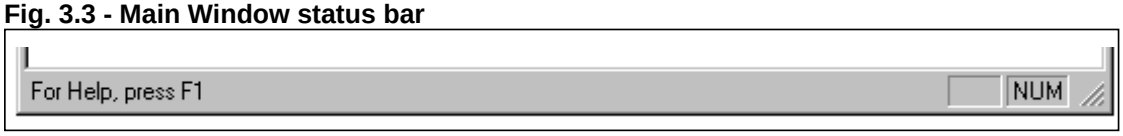
Fig. 3.2 - Main Window toolbar



The FUZZYSTUDIO™4.1 Main Window status bar

The status bar displayed at the bottom of the Main Window provides a brief description of the toolbar command currently pointed by the mouse cursor.

Fig. 3.3 - Main Window status bar



Project Files Management

FUZZYSTUDIO™4.1 stores the project data into a single file with the extension .FS4. The commands for the project file management (such as SAVE or OPEN) can be selected from the FILE menu. The enabled commands of the FILE menu are different according if there is an opened project or not. If the project has not been opened yet, the commands available for the project file management are the NEW and the OPEN commands.

The workspace status and some user settings are saved automatically, when closing the project, in a file with .FSW extension.

Starting a New Project

The NEW command allows to start a new project. When you choose this command, FUZZYSTUDIO™4.1 displays the New Project dialog box to define the project name, specify the target device and describe the project.

The project name must be compatible with a standard file name format because the project name will be used as project file name with the extension .FS4. 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 select quickly 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.*

The target device list contains the names of the devices installed in the current version of the development environment. If the target device of your application is not included in the list, please contact STMicroelectronics or open the Fuzzy Logic Web pages to get an updated version.

The New Project dialog-box also contains a text box where it is possible to write notes about the project or any information you need to recall. This information can be read or modified later by choosing the PROJECT INFO item from the FILE menu.

Working with an existing Project

The OPEN command allows to open a project previously saved. On the OPEN command the standard Open dialog box pops-up, allowing you to select the project file to be loaded.

The CLOSE PROJECT command allows to quit the current FUZZYSTUDIO™4.1 project. Then, you are asked to save the project if you have carried out some modifications not saved yet.

The SAVE command directly updates 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. If no modification has been carried out after the last saving, the SAVE command is disabled.

On the SAVE AS command the standard Save As dialog box pops-up, allowing you to change current project name. The Save As dialog box pops-up also performing the SAVE command when the project name has non been specified yet. SAVE commands will prompt for confirmation before overwriting an existing file.

The PROJECT INFO item allows to read the information relative to the project (Target Chip, name, working directory) and to read/write the user's project description.

To print the active window of the project use the PRINT command in FILE menu. PRINTER SETUP and PRINT PREVIEW commands are also available.

Project Window

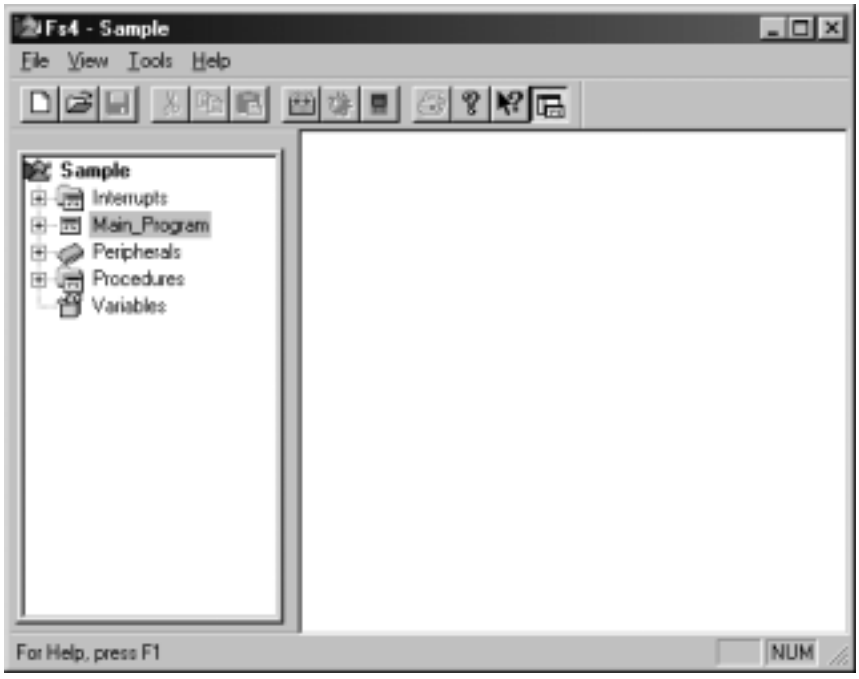
After creating a new project or opening an existing one, the Project Window is displayed on the left side of the Main Window. This is a “docking window” i.e. it is fixed in a dock along any of the four borders of the Main Window or it can be put outside. The Project window can be shown/hide by using the PROJECT TREE VIEW command from the VIEW menu or using the apposite toolbar button.

The Project Window allows you to access any part of the project and define user program procedures. It is organized in tree-view format, so you can access the parts of the program working as with the directory browser of the Windows 95 Explorer. The items (nodes) on the top of the tree are the following:

- Main Program
- Interrupts
- Peripherals
- Procedures
- Variables
- Tables

Click a plus or minus sign to expand or contract a node, in order to show or hide the items contained inside the node. Variables and Tables nodes cannot be expanded because they have not sub-nodes.

Fig. 3.4 - Project window



Main Program

Double clicking the Main Program item, the Blocks Editor window opens, allowing the editing of the Main Program block-diagram. All the program starts with the Start Block in the Main Program, after completing the configuration of the device as stated in the Peripherals Configuration tool.

Expanding the Main Program node clicking on the plus sign, the already inserted blocks are shown: double-clicking on them the relative contents is opened. Sub-nodes relative to the Folder Blocks can be further expanded so that they show the blocks contained in them. Right-clicking the mouse on the nodes representing a block, the pop-up menu opens displaying the blocks' related commands:

Open opens the editor associated to the block.

Close closes the editor associated to the block if open.

Rename opens the dialog-box to change the block's name.

Delete deletes the block.

Some block types support only part of the previous command. If the block is a Call Block, two additional commands are available:

Open Procedure opens the called user procedure.

Hide the procedure label hides the name of the procedure next to the call block.

If the Block is a Send or Receive Block already set, you can hide/show the related active Label. Refer to the Blocks Editor description for further information on these commands.

Interrupts

Clicking on the plus sign of the Interrupts node, the sub-nodes related to the interrupt service routines are shown. Double-clicking on one of them, the Blocks Editor to implement the interrupt routine opens.

Because the interrupt is an asynchronous event, the interrupt service routines are separated by the Main Program. The interrupt service routines editing windows are the same as the Main Program, with the exception that they start with the IRQ Block and end with one or more RETI Blocks.

Expanding the interrupt node, clicking on the plus sign, the already inserted blocks are shown: double-clicking on them the relative contents is opened. Sub-nodes relative to Folder Blocks can be further expanded so that they show the blocks contained in them. Right-clicking the mouse on the nodes representing a block, the pop-up menu opens displaying the blocks' related commands, as described before for the Main Program.

Peripherals

Double-clicking the Peripherals item, the Peripherals Configuration property-sheets opens. Each sheet refers to the configuration of one peripheral or device functionality and it is composed by some controls that allow to specify the configuration (see chapter 4).

Expanding the Peripherals node, clicking on the plus sign, the peripherals list is shown. Double-clicking on one of them causes the opening of the property-sheet with the sheet related to the peripheral put on top. The nodes are not further expandable and the only command available by right-clicking the mouse is the OPEN command.

Procedures

From this node, user procedures can be created by right-clicking the mouse. The commands available from the pop-up menu are:

Create Procedure:
creates a new procedure node with default name “Procedurex”, being x a progressive number.

Create Procedure and Open:
creates the procedure node and opens the editor window to define the procedure’s block-diagram.

After you create the procedures, they appear as sub-nodes of the tree-view. Right-clicking the mouse on them, the pop-up menu opens displaying the blocks’ related command, as described before for the Main Program.

The created procedure names are added in the list shown in the Call Block editor. Procedures start with the Start Block and end with one or more Return Blocks.

Deleting a procedure, you are requested for confirmation twice: the second time you are informed about the Call Blocks that are using the procedure, if any.

Variables

The item Variables has not sub-nodes and the only available commands are OPEN and CLOSE, accessed by right-clicking the mouse. Double-clicking on the item also opens the Variables window.

The Variables window shows the already defined and Predefined Variables (see Chapter 4), their types and properties, and allows the definition of new user variables to be used in the program.

See Chapter 4 to learn more about Variables and how to define them.

Tables

The item Tables has not sub-nodes and the only available commands are OPEN and CLOSE, accessed by right-clicking the mouse. Double-clicking on the item also opens the Tables definition editor.

The Tables definition editor shows the already defined constants and tables (see Chapter 4), their types and values, and allows the definition of new tables and constants to be used in the program.

See Chapter 4 to learn more about Tables and constants and how to define them.

4 - INITIAL SETTINGS

In this chapter you will learn how to configure the target device peripherals and to define the variables and data tables. The topics described are the following:

- Variables window and *Global Variables*.
- *Data Tables* definition Windows.
- Peripherals configuration property-sheet.

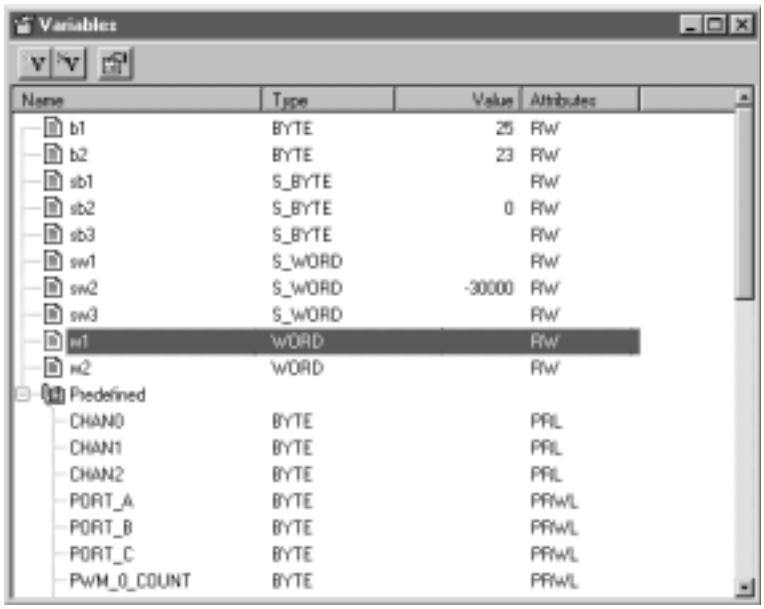
Variables Window

User-defined variables are called *Global Variables*. They can be used in the program inside the blocks that affect the RAM memory locations (such as Arithmetic Block). Each Global Variable is characterized by name, type and memory location: the user establishes the first and the second one and the Compiler automatically fixes the last.

The *Predefined Variables* are variables with a predefined name, supplied by the environment. They address to particular registers of the device target (typically the peripheral's ones). These variables cannot be changed by the user but can be used as the Global Variables to interact with the peripherals or the I/Os. Some of the Predefined Variables are read-only or write-only, when both readable and writable they refer to different memory locations at the same time.

The Variables window is the tool to define and initialize the Global Variables and show the Predefined Variables. To open the Variables window double-click the VARIABLES node in the Project Window tree-view.

Fig. 4.1 - Variables Window



The Variables window is composed by four fields:

Name: shows the variables names (user-defined or predefined)

Type: shows the variables type (byte, signed byte, word, signed word)

Value: To edit and show the initialization value

Attributes: shows the variables attributes (read, write, predefined, locked)

The Predefined Variables cannot be modified. The available Predefined Variable list depends on the chosen target device. See Appendix A to see the Predefined Variables for each device.

To insert a new Variable:

1. Right-click an empty point on the Variables window client-area;

2. Select the NEW VAR command and the variable type from the pop-up menu;

3. Change the default name **NewVarX**, being X a progressive number, with the name you like.

A new Global Variable can be inserted by pushing the INSERT key or using the toolbar button as well.

To initialize a Variable:

1. Click on the value field of the variable;

2. Edit the initialization value and push the ENTER key;

Act in the same way to change a previously set value.

To modify an existing Global Variable:

- Right-click the variable to be changed then:
  - select the command RENAME from the pop-up menu if you want to change the variable name;
  - select the command SET TYPE and choose the type if you want to change it;
  - select the command DELETE or push the Delete key or click the toolbar button if you want to delete the Global Variables;
  - select the commang EDIT VALUE to edit the initialization value of the variable;
  - select the command REMOVE ALL to delete all the Global Variables;

The same commands are available in the menu EDIT.

The variables have attributes that indicate the actions that can be performed on them. The attributes belong to the variables and cannot be changed. They are indicated with the following letters:

R indicates that the variable is readable

W indicates that the variable is writable

P indicates the predefined variables

L indicates that the variables cannot be modified (Locked) such as the predefined ones.

Note: *The maximum number of Global Variables that can be defined depends on the target device RAM space and on the dept of the calls to subroutines and interrupts. Actually the system stack uses the last RAM memory location, filling them from the last, two bytes for each call level.*

Filter dialog-box

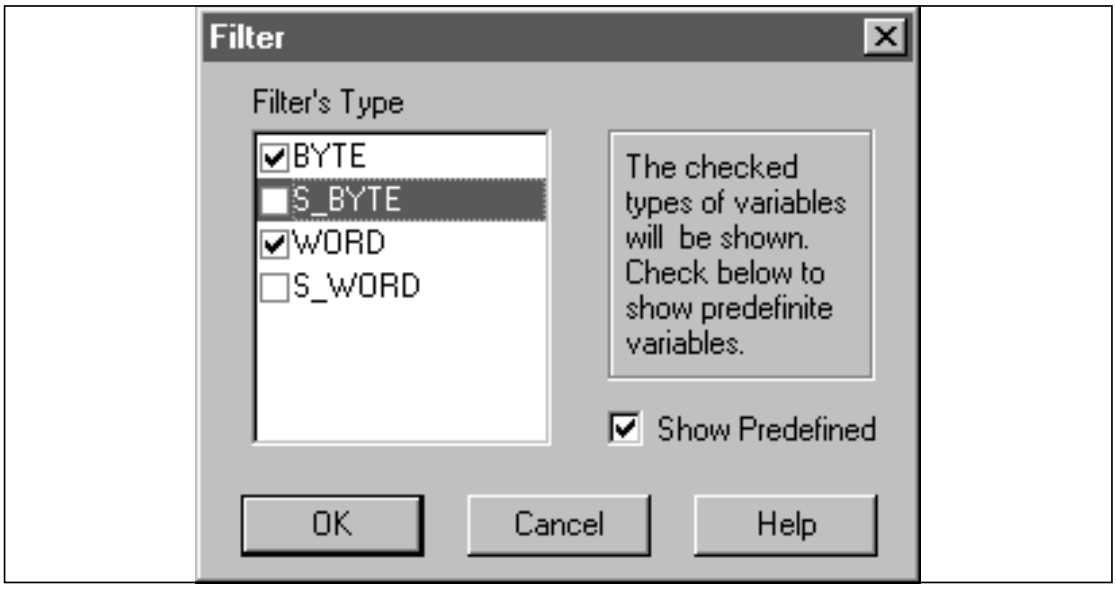
You can decide the variable types to be shown by using the Filter dialog-box opened by selecting the command SET FILTERS from the EDIT menu or by clicking the apposite button in the variables window toolbar.

Select the variables type to be shown in the check-list box by checking them;

Check the “Show predefined” check-box to show the Predefined variables folder.

Note: You cannot change the type of an existing variable with a filtered type: select the relative filter check-box and then change the type.

Fig. 4.2 Filter dialog-box



Tables Window

The Tables window is the tool to define data tables (i.e. vector of constants) and constants. It is useful to implement look-up tables and the constant definition allows the use of parameters improving the readability of the program. Tables can be used in the program inside the blocks that affect the RAM memory locations (such as Arithmetic Block). Data are stored in the device's Program Memory.

To open the Tables window double-click the TABLES node in the Project Window tree-view.

Fig. 4.3 - Tables window

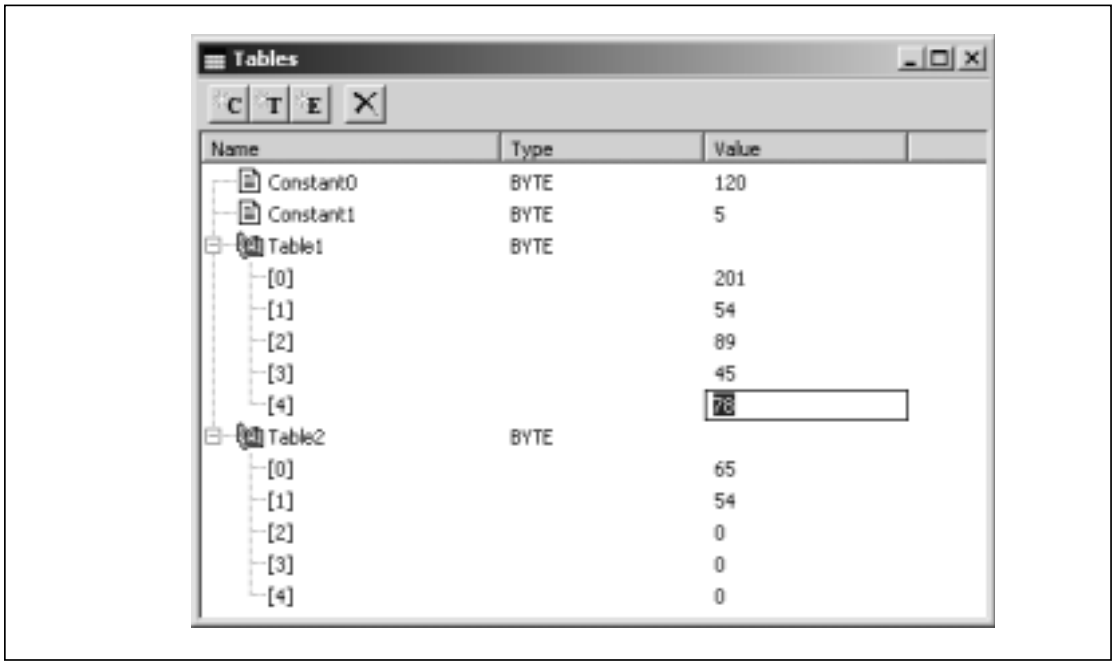
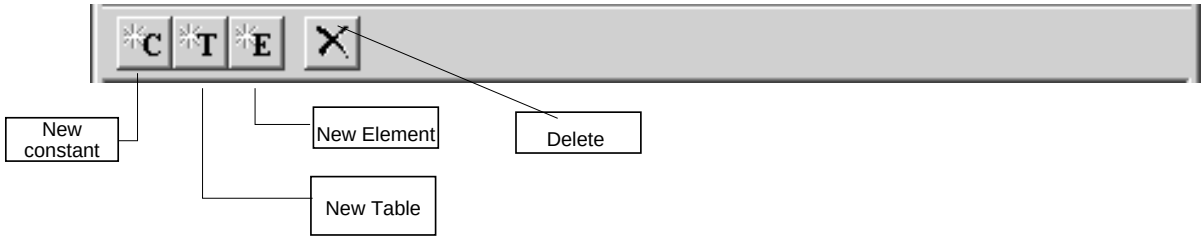


Fig. 4.4 - Tables window toolbar



The Tables window is composed by three fields:

Name: shows the tables and constants names

Type: shows the data type (byte, signed byte, word, signed word)

Value: shows the tables data values.

To define a new table:

1. Right-click an empty point on the Table window client-area;

2. Select the NEW TABLE command and the data type from the pop-up menu;

3. Change the default name “TableX”, being X a progressive number, with the name you like;

4. Set the value of the first element of the table, according with the type range, and push ENTER;

5. Click the apposite toolbar button to insert the next value.

To define a new constant:

1. Right-click an empty point on the Table window client-area or an existing table name;

2. Select the NEW CONST command and the data type from the pop-up menu;

3. Change the default name “ConstX”, being X a progressive number, with the name you like;

4. Set the value of the constant, according with the type range, and push ENTER.

To modify an existing table or constant:

1. Right-click the table or constant to be changed;

2. Select the command RENAME from the pop-up menu if you want to change the name;you can do the same by clicking over the selected name and editing the new name;

3. Select the command SET TYPE and choose the type if you want to change it;

4. Select the command DELETE or push the Delete key or the apposite toolbar button if you want to delete the table or the constant.

To modify a table or a constant value:

1. Click on the value you want to change;

2. Edit the new value and push ENTER key.

To add a new element in a table:

1. Select the element of the table where you want to insert the new one;

2. Push the INSERT key or the apposite toolbar button to add the element in the position of the selected one;

3. Edit the value and push ENTER.

To append a new element in a table:

1. Select the table where you want to insert the value;

2. Click the apposite toolbar button to add the element;

3. Edit the value and push ENTER.

To delete a table element:

1. Select the element to be deleted;

2. Push DELETE key or select the command DELETE form the pop-up menu or click the apposite toolbar button.

Note: *To execute the INSERT or MODIFY commands you can use the pop-up menu, the menu EDIT of the main window, the toolbar or the keyboard as well. Executing insert or modify commands on an empty point of the client-area or when a table or constant name is selected you will insert a new table or constant. In addition you can move along the values using the arrows keys.*

Peripherals Configuration

The device configuration is usually a hard task for programmers. By using the Peripherals Configuration property-sheet it becomes easy, because you have to select the configuration features instead of programming them.

The Peripherals Configuration property-sheet contents depends on the selected target device. You can find a complete description of each peripheral type configuration sheet in Appendix A.

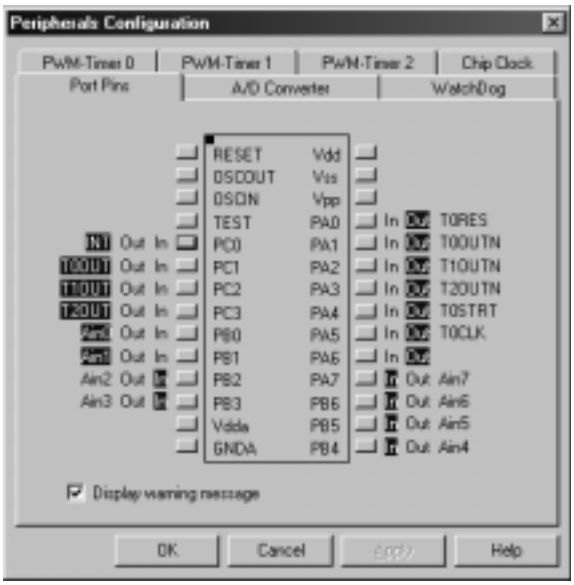
To access to the Peripheral Configuration property-sheet double-click the PERIPHERALS node in the Project Window tree-view or the sub-node related to the peripheral to be configured.

The property-sheet is a special kind of dialog box that has three main parts: the containing dialog box, one or more property pages (sheets) shown one at a time, and a tab at the top of each page that you can click to select that page. Each page refers to the setting of a single peripheral, which can be performed selecting the required features by means of check-boxes or radiobuttons and specifying data by means of text-boxes.

If you do not intend to use a peripheral, default configuration is assumed, that corresponds to the settings you can find before modifying the property-page.

- Click OK or APPLY button to confirm the settings.
- Click CANCEL button to delete the last modifications.

Fig. 4.5 - Peripheral Configuration property-sheet



5 - BLOCKS EDITOR

The Blocks Editor is the tool to design the block diagram of the program parts. The Block Editor is available in the following environments:

- Main Program
- Interrupt Service Routines
- Procedures
- Folders

The use of the Block Editor is the same for all the environments listed above, with some exceptions that will be described later in this chapter.

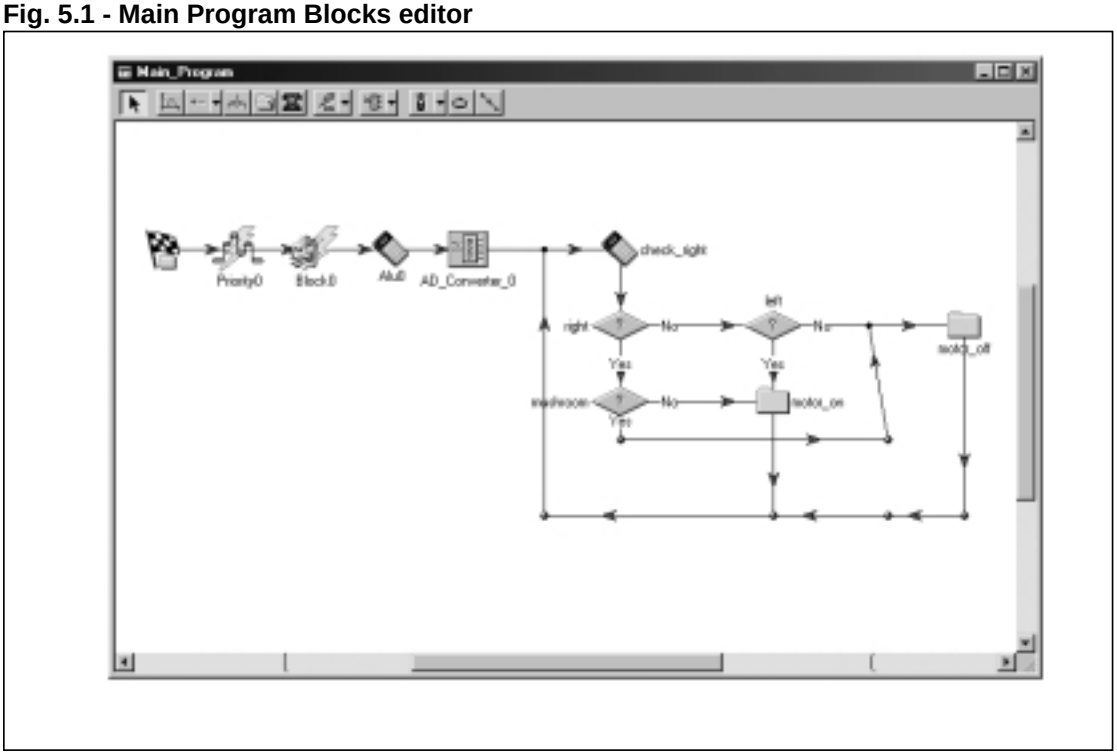
In this chapter you will learn to:

- Edit a block diagram
- Insert blocks and links
- Use standard editing commands with blocks and links

Blocks Editor Window

This section provides an overview of the major elements of the Blocks Editor such as menus, toolbar and status bar. These are the same in all environments based on Blocks Editor tool, with some exceptions as described later in this chapter.

Fig. 5.1 - Main Program Blocks editor



Blocks Editor menus

The following menu items are available when the Blocks Editor is open in foreground:

File Contains commands to create, open, close, save and print window contents and to visualize the project's information.

Edit Provides standard editing commands.

View Contains commands to hide/show toolbars, status bar, Project and Output windows.

Tools Contains commands to run Debugger, Compiler or Programmer tools.

Window Contains commands related to window management.

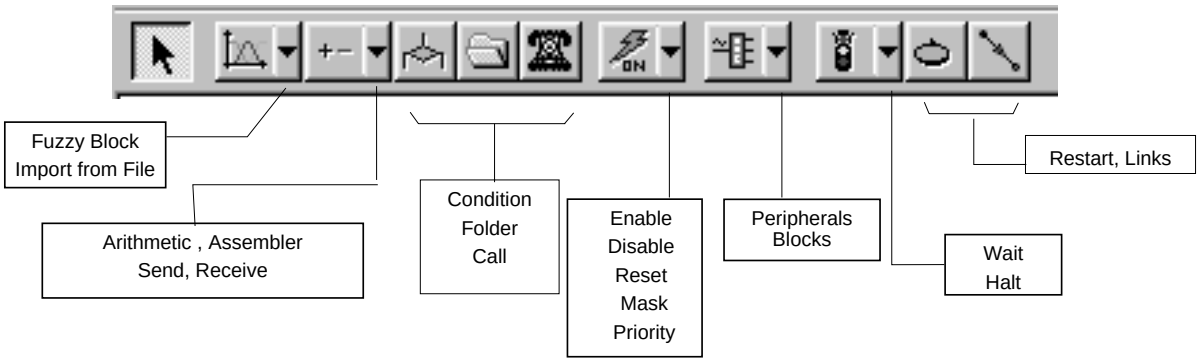
Help Contains help commands.

Blocks Editor window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar of the following environments:

- Main Program
- Procedures
- Interrupt routines
- Folders

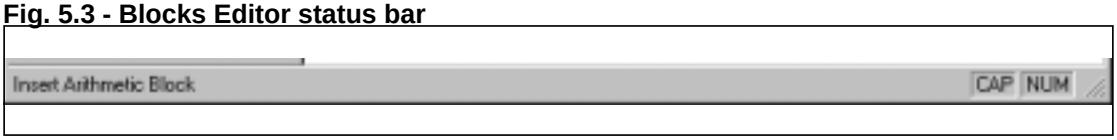
Fig. 5.2 - Blocks Editor toolbar



Blocks Editor window status bar

The Status Bar displayed at the bottom of the Blocks Editor provides a brief description of the toolbar command currently pointed by the mouse cursor.

Fig. 5.3 - Blocks Editor status bar



FUZZYSTUDIO™4.1 Blocks

The blocks available in the FUZZYSTUDIO™4.1 Blocks Editor are the following ones:

Icon	Block name	Description
	Start	Starting point of the program or procedure
	Fuzzy	Allows to perform the fuzzy functions
	Arithmetic	Carries out arithmetic and logic operations
	Assembler	Allows to insert arithmetic instructions in Assembler
	Conditional	Modifies the program flow in relation to user-defined conditions.
	Interrupts Disable	Disables globally the interrupts
	Interrupts Enable	Disables globally the interrupts
	Interrupts Reset	Resets all pending interrupts
	Interrupts Mask	Enables selectively interrupts
	Interrupts Priority	Manages interrupts priority
	Peripherals	Group of blocks that enables/disables and sets/resets the peripherals.
	Send	Sends the value to a peripheral coming from a register
	Receive	Receives a value from a peripheral and stores it in a RAM location
	Call	Calls user-defined procedures
	Wait	Stops the program until the first interrupt signal
	Halt	Puts the device in Halt mode
	Restart	Restarts the program
	Return	Ends the user-defined procedures
	IRQ	Starts a interrupt request service routine
	RETI	Ends a interrupt request service routine
	Folder Block	Allows to group blocks
	Links	Connect two different parts of the program

In a Block Editor window not all the blocks are available at the same time. Some of them can be used only in particular windows. For example, the RETI block is available only in Interrupts Routines Blocks Editor windows.

The Links are not proper blocks, but some functionalities are similar to the ones of the "object" block.

Block Diagram Starting Point

Each block diagram edited with the Blocks Editor has a starting point, represented by a particular block with no associated editor. This block is inserted automatically by the editor and cannot be removed. In the following, the blocks used as block diagram starting points are listed:

Block diagrams have also one or more exit points, except the Main Program. They are the following:

Type of block	Description
Return block	Return point of the procedures
RETI block	Return from interrupt service routine
Folder’s return block	Folder's exit point

Labels

A label, i.e. a symbolic name, is associated to each block. Such label represents the program address where the first instruction contained in that block starts.

Default names are inserted automatically when inserting the block, but they can be changed anytime by selecting the item RENAME from the pop-up menu that opens by right-clicking on the block.

You can move a label around the referring block by click & drag operations to the desired position. Labels can be snapped on the cardinal points or positioned anywhere around the block according to the activation of the SNAP ON or SNAP OFF item you can choose from the pop-up menu which opens by right-clicking on the label.

Note: *Labels can contain up to 32 characters including spaces.*

Working with Blocks

This paragraph supplies you information on how to work with the Blocks Editor, in order to create the Block Diagram of your program.

Inserting blocks

The Block Diagram is built by inserting the appropriate blocks, by using the mouse and the toolbar.

To insert a block:

1. From the Block Editor window toolbar select the block you want to insert in your diagram.

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. Click & drag the mouse on the block and then release it to move the block in the desired position.

After inserted, a Block can be freely moved inside the client-area by using the click & drag. Moving a single link determinates the moving of the connected blocks. Moving a block determinates the modification of the link because the connected edge moves with the block.

Linking blocks

Links connect two blocks and an arrow indicating the sequence of the program execution characterizes them. A link cannot be connected to a block already linked with the same direction: each block can have only one input and one output link. The Conditional Block has two output links tagged with "Yes" and "No".

To link two blocks:

1. From the Block Editor 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, double-click on the client area, the open link is indicated with a red node. The links can be connected to other links allowing to carry out cyclical programs; a black node on the link indicates the connection. Blue nodes indicate a link edge connected to another link edge. After you connect the links these can be moved freely in the client area.

Disconnect blocks and links

Links and Blocks can be disconnected by using the pop-up menu commands, accessed by right-clicking the item with the mouse.

To disconnect a Block from the diagram:

- select the Disconnect command from the pop-up menu.

To disconnect a link, select one of the following commands:

- **Disconnect from Start Block:** disconnects the link from the source block
- **Disconnect from the End Block:** disconnects the link from the destination block
- **Disconnect from both:** disconnects the link from both blocks
- **Split**: inserts a blue node in the link spitting it in two connected links. You can do that double-clicking on the line as well.

Single and multiple selection of blocks

You can select a single block or a link just clicking on it. Take note that this operation cannot be performed in link insertion modality. In this case, you will have to activate the cursor before selecting 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 several 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 color: the blocks become gray and the links become green.

Deleting blocks and links

To delete blocks or links:

- select the block(s) or link(s) to be deleted (single or multiple selection)
- push the DEL key or select the command DELETE from the pop-up menu or from the EDIT menu.

You are always asked for confirmation before deleting a block then:

- click YES button to confirm the deleting of the single block
- click YES TO ALL to confirm the deleting of all the selected blocks
- click NO to avoid the deleting of the single block
- click QUIT to end the deleting procedure

Opening and closing blocks

To open the block means to open the editor window associated to the block, in order to specify the commands and settings to be performed by the block.

To open a block:

- double click the block

  or

- select the OPEN command from the pop-up menu, accessed by right-clicking the mouse on the block.

The opened blocks are identified in the block diagram by a little folder located on the lower-right side of the icon representing the block. Double-clicking on such icon determines the editor window to become active in foreground.

In the same way, to close the block means to close the associated editor.

To close a block:

- close the editor's window

  or

- select the CLOSE command from the pop-up menu, accessed by right-clicking the mouse on the block. This command is active only when the block is opened.

Copying blocks

Standard editing commands such as COPY, CUT and PASTE are provided to duplicate blocks and links into other parts of the project or in other projects. Blocks are copied with their contents, i.e. with the commands and settings specified in the associated editor.

To copy, cut or paste blocks and links:

- select the block(s) or link(s) (single or multiple selection)
- select the command related to the action to perform from the EDIT menu or click the apposite toolbar button in the Main Window.

The COPY and PASTE commands allow to copy blocks from a project to another one. When pasting one or more blocks, the consistency of the operation is checked:

1. If the block already exists with the same name, the pasted block's name is modified appending to it a progressive number.

2. If the block settings specify a not yet defined variable, a warning message is issued and you are asked to paste the block empty or to keep the setting and define later the variable. This control cannot be performed when pasting Arithmetic or Assembler blocks.

3. If there is no room for Global of Fuzzy Variables or Membership Functions added with the paste operation, a message is issued and the pasting of the block is aborted.

If you want to copy the block(s) as part of the diagram drawing, select the command COPY ON CLIPBOARD AS BITMAP from the EDIT menu. This option allows you to include the block diagram in documents and similar.

Note: *Paste command is activated only after performing a copy or cut operation.*

Other commands

Right-clicking on the Block Editor client-area when in cursor modality, a pop-up menu appears containing the following items:

Grid: Enables or disables the grid

Line up: Aligns the Block Editor objects on the grid

Moreover, double-clicking the client area when in cursor modality, the mouse cursor is modified allowing the moving of the entire diagram.

6 - FUZZY BLOCK

FUZZY
BLOCK

This chapter is dedicated to the Fuzzy Block description and the use of the associated editor. This is the most complex block, due to the several options available. The Fuzzy Block allows to carry out the fuzzy functions to be performed by the ST52 fuzzy family microcontrollers.

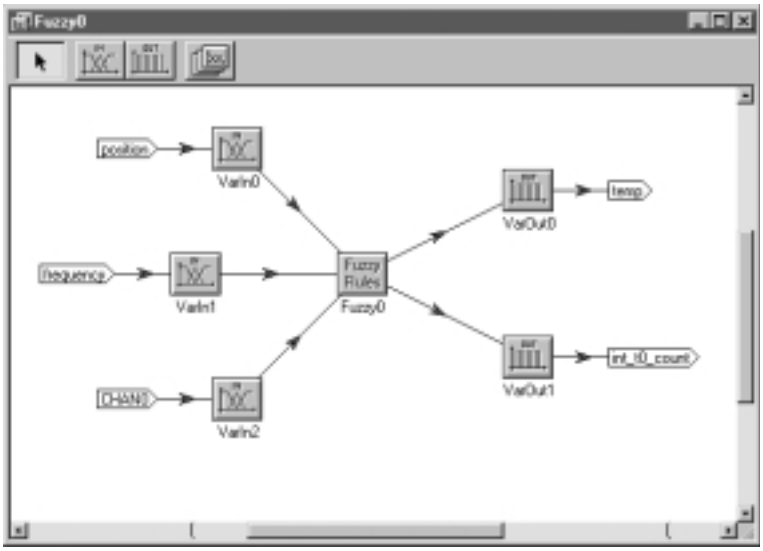
The possibility to define more than one Fuzzy Block in the block diagram, allows to use the fuzzy system which is more appropriate to the general conditions of the system and then to realize an adaptive fuzzy control system.

Double-clicking on the Fuzzy Block icon, the fuzzy system editor opens, allowing you to graphically define the fuzzy system structure, fuzzy variables, Membership Functions and fuzzy rules. The environment is composed by some editors, each of them suited for its functions. The first editor you can find when the environment is opened, is the fuzzy system editor, from where it is possible to access to the others editors.

In this chapter you will learn to:

- Edit the fuzzy system
- Define fuzzy variables
- Draw membership functions
- Write fuzzy rules
- Import Fuzzy Systems

Fig. 6.1 - Fuzzy System Editor



Fuzzy System Editor

The Fuzzy System Editor allows to define the fuzzy system structure, in terms of input and output variables and the associations with the Global or Predefined variables. This action is performed inserting as many blocks, representing Input and Output variables, as you need in your system.

The Input and Output variables blocks represent the variables and allow to access to the editors to define their characteristics by double-clicking on them. Associations with the Global or Predefined Variables can be easily performed and it is highlighted in the fuzzy system structure with tags linked with the variables blocks. The base of knowledge is represented by the Rules Block inserted and linked automatically, that allows to access to the Rules Editor.

Fuzzy System Editor menus

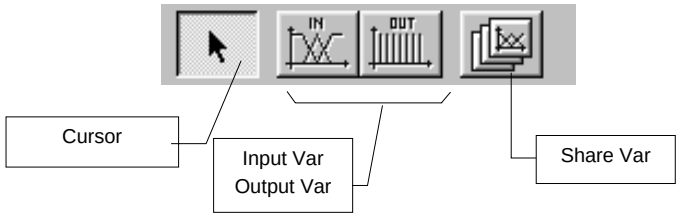
The following menu items are available in the Main Window when the Fuzzy System Editor is open in foreground:

File	Contains commands to create, open, close, save and print window contents and to display the project's information.
Edit	Provides standard editing commands.
View	Contains commands to hide/show toolbars, status bar, Project and Output windows.
Tools	Contains commands to run Debugger, Compiler or Programmer tools.
Insert	Contains commands to insert fuzzy variables.
Window	Contains commands related to windows management.
Help	Contains help commands.

Fuzzy System Editor window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar:

Fig. 6.2 - Fuzzy System Editor toolbar



Fuzzy System Editor window status bar

The status bar displayed at the bottom of the main window when Fuzzy System Editor is open in foreground, provides a brief description of the toolbar command currently pointed by the mouse cursor. In addition, on the right side you can see the number of the defined input variables, output variables and rules.

Fig. 6.3- Fuzzy System Editor status bar

Fuzzy System Editor

The Fuzzy System Editor works as the other Blocks Editors: you can insert, move, delete, rename, open and close the blocks acting as explained in Chapter 5 in this manual. Links are inserted automatically, after writing the fuzzy rules and cannot be disconnected.

If you open the Fuzzy System Editor for the first time, you will find only the Fuzzy Rules Block, that allows to access to the Rules Editor by double-clicking on it. Notice that if the variable and membership functions are not yet defined, you cannot access to the Rule Editor. The Fuzzy Rules Block is inserted automatically and cannot be deleted.

To insert input fuzzy variables:

- click the apposite toolbar button or select INPUT VAR from the menu INSERT
- click the mouse button on the client area where you want to insert the input variable block.

To insert output fuzzy variables:

- click the apposite toolbar button or select OUTPUT VAR from the menu INSERT
- click the mouse button on the client area where you want to insert the output variable block.

The icons representing fuzzy variables allow to access to the programming environment of the variables characteristics, such as associated Membership Functions, Universe of Discourse boundaries, initialization with global or predefined variables.

Note: *You can insert up to 8 input fuzzy variables and up to 128 output fuzzy variables for each fuzzy block.*

Fuzzy Variables Initialization and Storage

To assign to the fuzzy input variable, a value to be computed, you have to associate it to a Global or Predefined variable containing the crisp value.

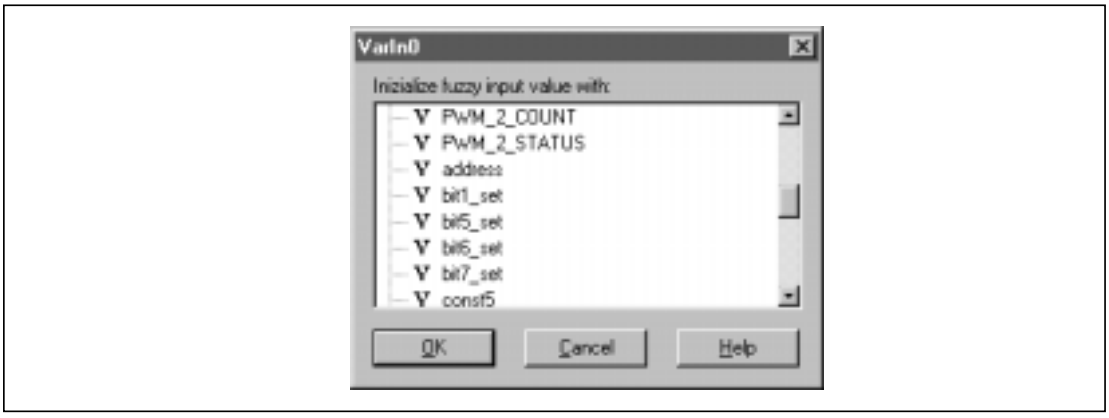
To initialize an input fuzzy variable:

1. Right-click with the mouse on the Input Variable block you want to initialize.

2. Choose INITIALIZE from the pop-up menu. The window containing the list of the available Global and Predefined Variables opens.

3. Select the variable and click OK button.

Fig. 6.4 - Fuzzy Variable Initialization dialog box



To store the fuzzy output in a global or predefined variable it is possible to act as for the initialization:

1. Right-click with the mouse on the output variable block where you want to store the value.

2. Choose STORE IN from the appearing pop-up menu. The window containing the list of the available Global and Predefined Variables opens.

3. Select the variable and click OK button.

The initialization and storage can be carried out also by means of the Var Properties dialog-box (see later in this chapter). After initialization and storage phase, in the Fuzzy System Editor window, tags showing the variables' names are linked to the fuzzy variables blocks (see fig.6.1). Right-clicking over the tag, the pop-up menu opens allowing you to select the EDITcommands, to modify the association, or DELETE, to cancel the association and the tag.

Note: *If the initialization of the variable is omitted, an error message will appear during the compilation.*

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:

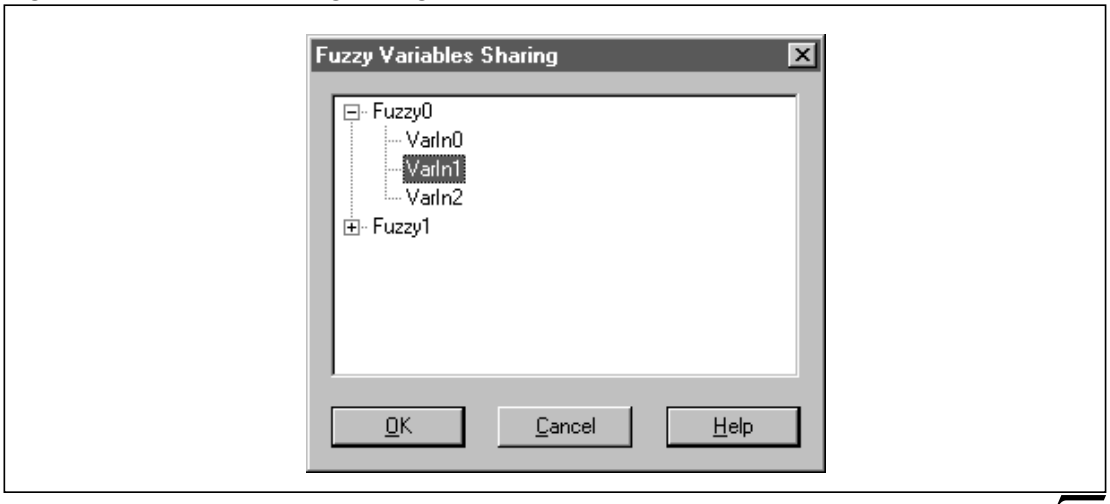
1. Select SHARE VAR from the INSERT menu or click the apposite button in the Fuzzy System Editor window toolbar.

2. A selection window, organized as tree-view, opens. In the first level you can find the already available fuzzy systems; in the second level of each item the available fuzzy variables are listed

3. Choose the fuzzy system, to which belongs the variable to share, by clicking on the plus sign next to the system name to show the available variables.

4. Select the variable to share. The window disappears and the mouse pointer is enabled to insert the variable on the fuzzy system. In both the fuzzy systems sharing the variable, the icon representing it changes its icon indicating that such a variable is shared by several blocks.

Fig. 6.5 - Variables Sharing dialog box



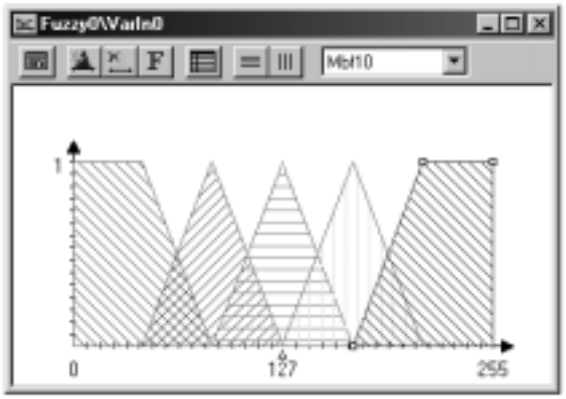
The name of the shared variable is the same in all the blocks in which it is used: a rename of one of them will determine the change in the other fuzzy systems in which it is shared.

Note: *it is not possible to insert in a fuzzy system a shared variable that has the same name of a fuzzy variable already existing in that system. In this case, rename this last variable and try again to insert the shared variable.*

Variables and Membership Functions Editor

Double-clicking the fuzzy variable icon in the Fuzzy System diagram, the editor for the definition of the variable's properties and its associated Membership Functions opens.

Fig. 6.6 - Variables Editor



It allows you to:

- change the variable's name
- modify the Universe of Discourse boundaries
- define or modify the association with the Global and Predefined variables
- define graphically or in numeric way the Membership Functions

Variables Editor menus

The following menu items are available in the Main Window when the Variables Editor is open in foreground:

File Contains commands to create, open, close, save and print window contents and to visualize the project's information.

Edit Provides commands to edit Membership Functions and to copy them on clipboard.

View Contains commands to hide/show toolbars, status bar, Project and Output windows and to open the variables properties dialog box and the MBF Report window

Tools Contains commands to run Debugger, Compiler or Programmer tools.

Options Contains commands to set up the Membership Functions Editor

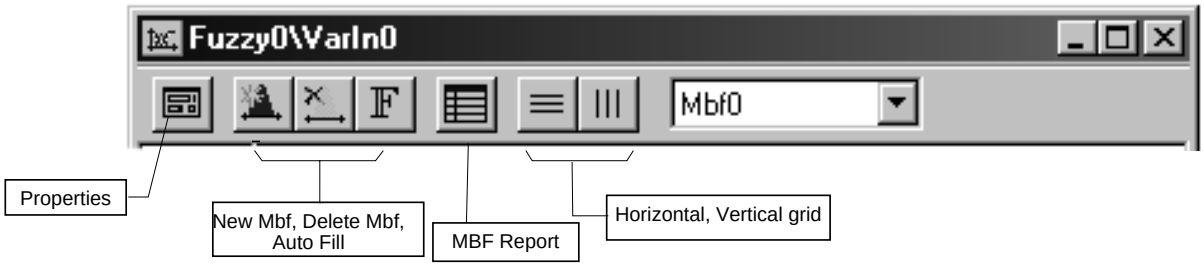
Window Contains commands related to window management.

Help Contains help commands.

Variables Editor window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar of the Variables Editor window:

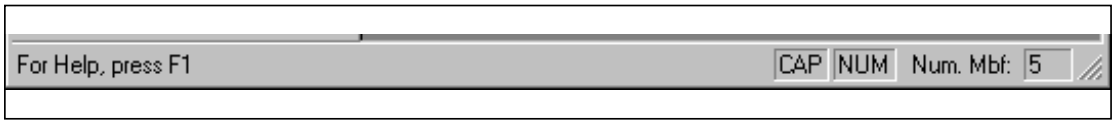
Fig. 6.7 - Variables Editor toolbar



Variables Editor window status bar

The status bar displayed at the bottom of the main window when the Variables Editor is open in foreground, provides a brief description of the toolbar command currently pointed by the mouse cursor. In addition, on the right side you can see the number of the defined Membership Functions.

Fig. 6.8 - Variables Editor status bar



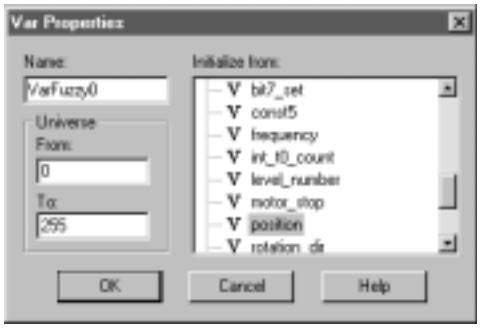
Fuzzy Variables Properties

Double-clicking over the variable's icon or selecting the command OPEN from the pop-up menu opened right-clicking the icon, the Variables an Membership Functions Editor opens, allowing you to access to the dialog box for the definition of the variable's properties. These are the following:

- name
- Universe of Discourse boundaries
- Initialization/Storing variable

To define or modify the variable's properties, select the command VAR PROPERTIES from the VIEW menu or click the apposite toolbar button from the editor's toolbar. The Var Properties dialog box opens.

Fig. 6.9 - Var Properties dialog box



Default name for input variables is VarInX, being X a progressive number starting from 0; default name for output variables is VarOutX. You can change the default name or the previously defined name by writing the new name in the apposite text box.

Note: *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.*

Default values for the Universe of Discourse boundaries are [0 , 255]. These values correspond to the ones managed by the device. You can modify the logical values of the boundaries, to customize the Universe of Discourse and relative values to the one of your real application. These values are automatically converted by the Compiler to the physical device range. You can change the default values, or the previously defined values by writing the new name in the apposite text boxes of the Var Properties dialog-box. Due to its internal resolution, ST52 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 , $255*10^{-6}$].

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

You can associate fuzzy variables with Global or Predefined variables, by selecting the variable from the list "Initialize from" (input variables) or from the list "Store in" (output variables). You can also perform this operation acting as described previously in the "Fuzzy Variable Initialization and Storage" paragraph in this chapter.

Creating a New Membership Function

The definition of the variables is completed by drawing the Membership Functions (M.F.) associated to the variable. M.F. are created and managed by using the commands in the EDIT menu, when the Membership Functions Editor is open in foreground, and from the commands in the pop-up menu opened right-clicking over the editor's client area.

You can create a new M.F. by using:

- the NEW MBF command, to create a single new M.F.
- the Auto Fill tool, to create an entire set of M.F.
- the Rule Editor (only for output variables)
- the MBF Report tool

To create a single new M.F. you can perform the NEW MBF command in the following ways:

- choosing the NEW MBF command from the pop-up menu
- choosing the NEW MBF command from the EDIT>MBF menu
- clicking the apposite editor's toolbar button
- pushing the INS key

The M.F. position into the variable Universe of Discourse is set as follows:

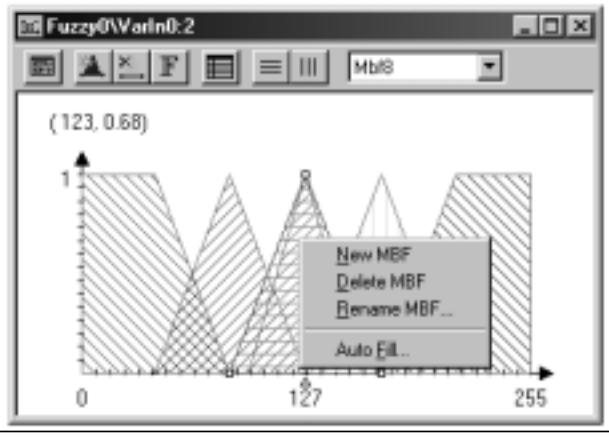
- In the center of the Universe of Discourse, if M.F. is the first to be defined.
- Shifted of the AutoShift parameter with respect to the previously defined M.F.
- Coinciding with the current cursor position in the drawing area, if using the pop-up menu and if this position does not coincide with an existing M.F. The base width of the shape is equal to the double of the Semibase parameter.

For more information see "AutoShift and Semibase" paragraph.

Note: *For output variables only crisp M.F. can be created, according to the hardware constraints of the device. Admissible M.F for input variables are the triangular and the trapezoidal ones.*

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

Fig. 6.10 - Creating new Membership Functions



The Universe of Discourse resizing does not modify this M.F. crisp value but may cut off the M.F. itself and its related rules, if the M.F. value is outside the new Universe of Discourse boundaries.

For further information about the definition of a M.F. by using the Rules Editor, see the “Rules Editor” paragraph.

Auto Fill tool

The Auto Fill tool allows to define a set of M.F equally spaced with just a click of the mouse. There are three available modes to draw the M.F. set as described below.

To draw the M.F. set:

1. Open the M.F Editor.

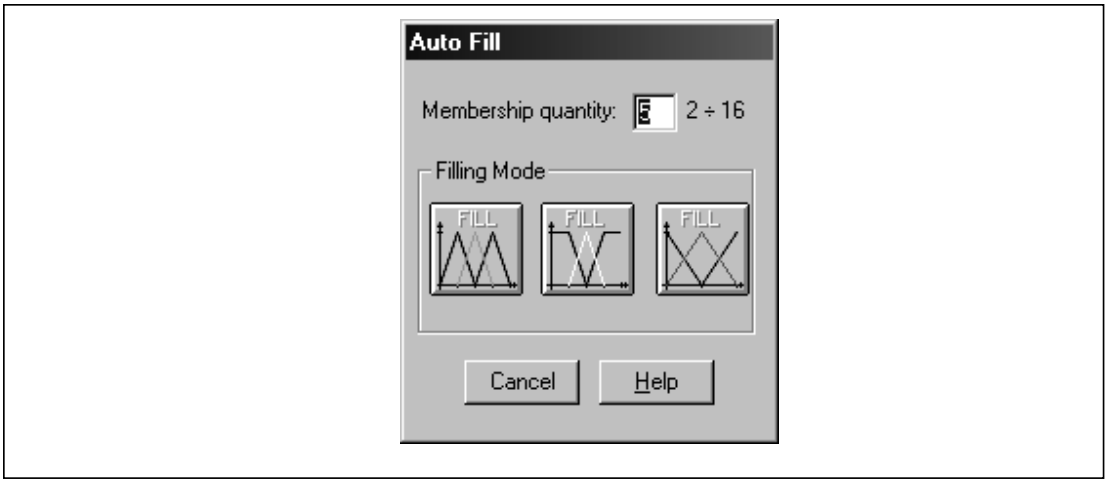
2. Choose AUTO FILL command from the EDIT menu of from the pop-up menu or from the apposite button in the editor’s toolbar.

3. Enter the desired number of fuzzy sets in the apposite text box.

4. Press the desired Filling Mode button.

The AutoFill tool can defines at least two M.F. and up to 16 M.F per time. If you want to define more than 16 M.F. repeat the Auto Fill command again.

Fig. 6.11 - Autofill dialog box



Note: *The Auto Fill tool does not create M.F. completely overlapped over existing M.F. So, if you have chosen 7 M.F. to draw, which two are overlapped, Auto Fill tool creates only 5 M.F.*

The filling modes work as follows:

Draws n isosceles triangular M.F. The semibase of new M.F. 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 central interval (or is shared out between the two central ones, if n=1 is even).

Draws two external rectangle trapezoidal M.F. all other M.F. are triangular. The semibase of new M.F. 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).

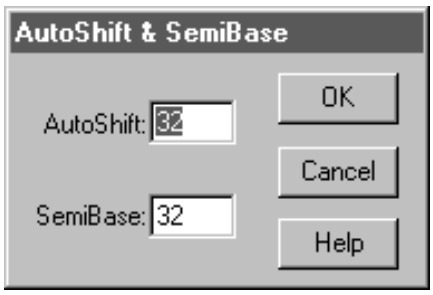
Draws two external rectangle triangular M.F. all other memberships are triangular. The semibase of new M.F. is calculated by splitting the Universe in n-1 equal intervals. Since x-values must be integers, the remainder of the integer division of the domain width by n-1, if any, is added to the central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

Autoshift and Semibase parameter

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

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

Fig. 6.12 - Autoshift and Semibase dialog box



To change these values:

1.Choose SET AUTOSHIFT & SEMIBASE from EDIT menu

2.The related dialog box pops up showing the current values

3.Change the values writing them in the apposite text boxes

4.Press OK

Note: *these values are set using the device internal measurement unit, and are independent from the chosen dimension of the variable domain. Due to the device hardware constraints, the AutoShift and SemiBase values must belong to the [1,255] integer interval.*

Modifying the Membership Functions shapes

After created, the M.F can be easily and quickly modified or moved. To do that, you first have to select the M.F.

To select the M.F. you can either:

- Click over the M.F. shape in the M.F. Editor or
- Select the M.F. name from the apposite drop-down list-box in the M.F. editor toolbar

When selected, the M.F.'s perimeter become red, and the vertexes are surrounded by a little square. These squares allow to modify the M.F. shape, moving the vertexes. The crisp M.F. are similarly highlighted in red, and the vertex (the top of the line) is surrounded by the square.

To modify the shape:

1. Select the M.F.

2. Click & drag the vertexes squares to the desired position

or

Click the mouse button and use the arrows keys to move the vertexes.

Note: *Central vertex can be moved only along the Y coordinate with maximum degree and between the lateral vertexes' X coordinate. The lateral vertexes can be moved only along the lower Y coordinate: they can have different Y coordinate only in the Universe of Discourse boundaries, in order to create trapezoidal M.F.*

To move the shape:

1. Click & drag the shape to the desired position

or

Click the shape and move it by using the arrows keys

2. Release the mouse button

Each M.F. has a name that is set by default to "MbfX", being X a progressive number incremented each time a new M.F. is created.

To change the M.F.'s name:

1. Select the M.F.

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

3. The dialog box to change the M.F. name opens.

4. Write the new name in the apposite text box and click O.K.

Note: *M.F. 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.*

To delete a M.F.:

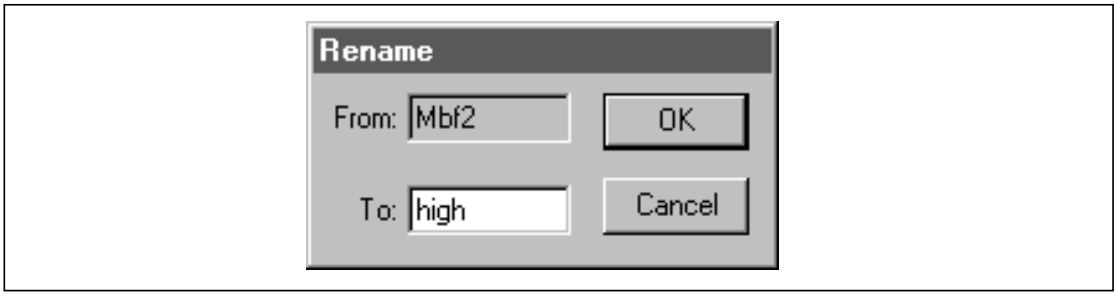
1. Select the M.F.

2. Push the DEL key

or

Select the DELETE MBF command from the EDIT > MBF menu or from the right mouse button pop-up menu.

Fig. 6.13 - Rename MBF dialog box



MBF Report tool

The MBF Report is the tool to visualize and define the M.F. in numeric form. You can perform the same operation with the same command of the graphical way. Each action executed in the MBF Report tool is reflected to the graphical representation and vice versa.

Four fields compose the MBF Report tool window: the first lists the M.F. names, the others contain the vertexes coordinates. Each coordinate is expressed by a couple of value: the first is the position on the Universe of Discourse (X coordinate), the second is the degree of truth (Y coordinate).

To open the MBF Report tool:

- Select the MBF REPORT item form the VIEW menu

  or

- Click the apposite button in the Membership Functions Editor toolbar

With the MBF Report tool you can define, modify, delete the M.F. by using the same commands and menu items as the ones described for the graphical way. In addition you can modify the vertexes' coordinates values or the M.F. names directly in the tool window.

To modify the M.F. name:

1. Select the M.F.

2. Click on the M.F. name.

3. Change the name and push ENTER or click the mouse button elsewhere.

To modify vertexes' coordinates:

1. Select the M.F.

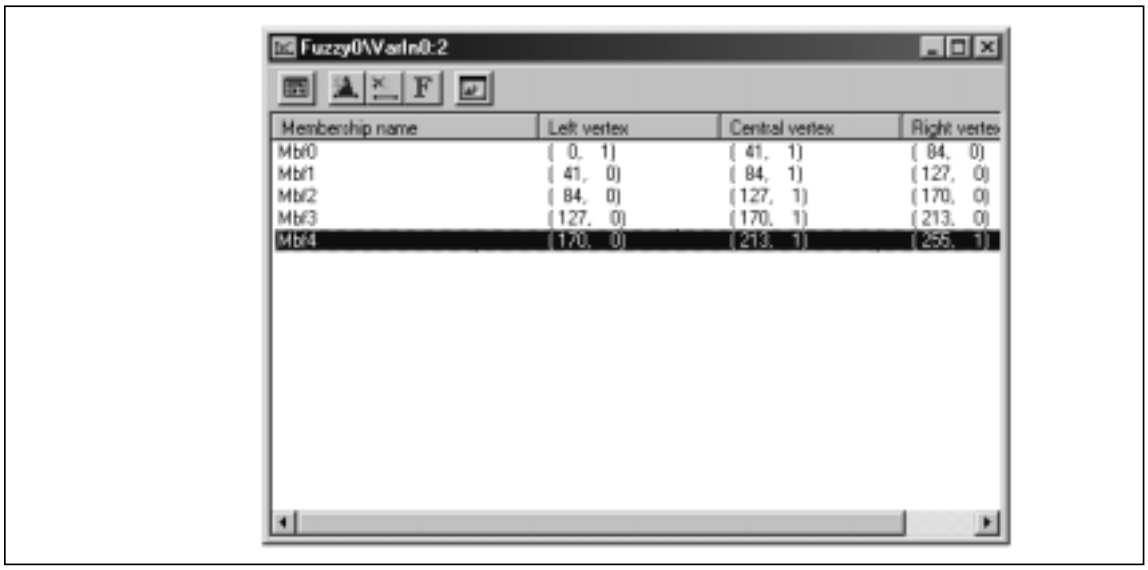
2. Click over the value to be changed

3. Change the value

4. Push ENTER or click the mouse button elsewhere

Note: *Not valid values are not entered, and previous values remain unchanged.*

Fig. 6.14 - M.F. Report window



Membership Functions Editor Options

The Membership Functions Editor can be customized by setting the environment options, by using the commands of the OPTIONS menu. You can set the following:

- Membership Functions Editor window dimension
- M.F. shape type (filled or not)
- hide/show X axis values
- hide/show cursor coordinates
- toggle the cursor coordinates from user's scale to the device scale
- hide/show M.F. names on print
- hide/show horizontal and vertical grids

The M.F. Editor dimension can be easily changed by modifying the window dimension. The M.F. shapes and all the drawing area are scaled proportionally to the window new dimensions.

The M.F. shapes are drawn filled with random colored patterns. You can disable this feature, getting not filled M.F. shapes, by toggling the FILLED MBF item in the OPTIONS menu.

The values of the X axis, i.e. the Universe of Discourse main points, can be hidden or shown toggling the X AXIS VALUES item in the OPTIONS menu.

The cursor coordinate can be hidden or shown by toggling the item CURSOR COORDINATES from the OPTIONS menu. The cursor's vertical coordinate (the degree of truth) can be expressed both in logical scale (from 0 to 1) or in device coordinate (from 0/15 to 15/15). Actually the device can express the degree of truth values with four bits, so the device values lie in the range [0 , 15]. You can toggle from logical to device coordinates and vice versa selecting the item DEVICE COORDINATES in the OPTIONS menu.

Note: *the item DEVICE COORDINATES is disabled when the cursor coordinated are hidden.*

When printing the M.F. set, the name of the label can be included over the shape. To hide or show the M.F. names on print toggle the MBF NAMES ON PRINT item in the OPTIONS menu.

Horizontal and vertical grid can be inserted to help you in drawing the M.F. shapes. To hide/show the grid lines toggle the HORIZONTAL GRID item and/or the VERTICAL GRID item in the OPTIONS menu, or click the apposite toolbar buttons.

Note: *when the MBF Report window is active in foreground, the only available menu items are DEVICE COORDINATES and MBF NAMES ON PRINT.*

Rules Editor

The Rules Editor is the tool to define and modify the fuzzy rules. To run the Rules Editor double-click over the Fuzzy Rules Block in the Fuzzy System editor. If rules have been already defined , the editor opens showing the list of the defined rules.

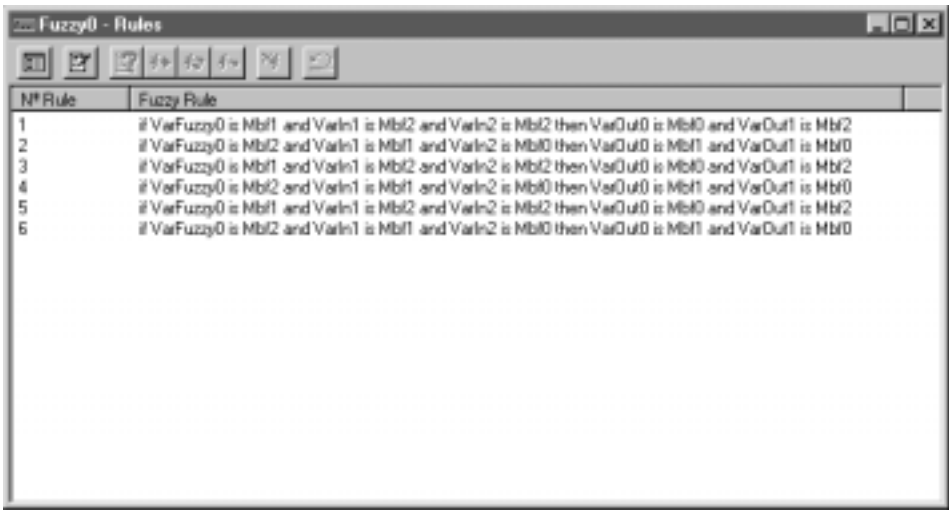
In order to define fuzzy rules, using Rules Editor, it is necessary to define at least one input and one output fuzzy variable with the associated M. F. Rules Editor cannot be activated until these minimal definitions have not been done.

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

In Guided Editor, keywords, if-then operators, variables and membership functions can be easily selected from a list-box for quickly defining the rules. This editing mode allows you to select automatically only syntactically correct objects.

In Manual Editor, 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.

Fig. 6.15 - Rules Editor



The rules list is composed by two fields: the first is the progressive rule number and the second one is the rule. You can use standard editing commands such as COPY, CUT, PASTE and DELETE from the EDIT menu after selecting the rule(s) by clicking over it (them). Multiple selection is possible by using the mouse with the SHIFT and CTRL keys. You can also use the GO TO command from the EDIT menu to select a rule specifying its number.

Rules Editor menus

The following menu items are available in the Main Window when the Rules Editor is open in foreground:

File Contains commands to create, open, close, save and print window contents and to display the project's information.

Edit Provides standard editing commands.

View Contains commands to hide/show toolbars, status bar, Project and Output windows, to navigate along the error messages and to change rules fonts.

Tools Contains commands to run Debugger, Compiler or Programmer tools.

Rules Contains rule related commands, for Manual and Guided editing.

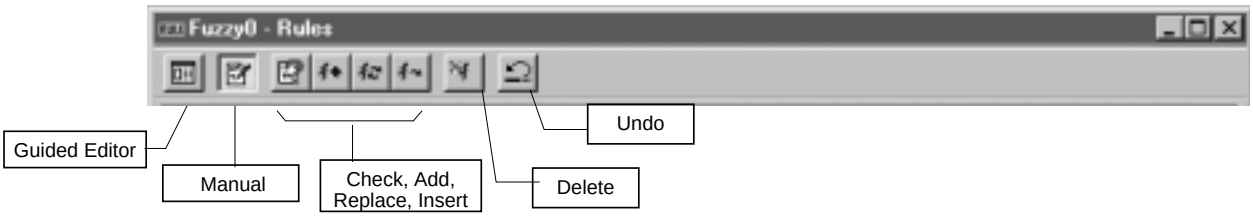
Window Contains commands related to window management.

Help Contains help commands.

Rules Editor window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar of the Rules Editor window.

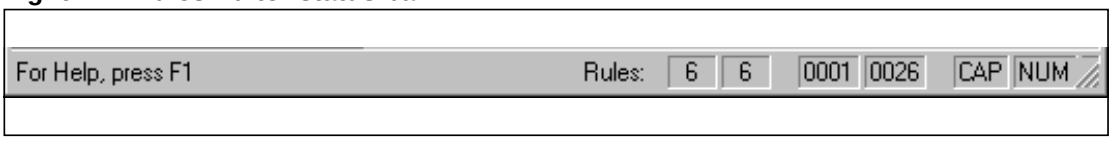
Fig. 6.16 - Rules Editor toolbar



Rules Editor window status bar

The Status Bar displayed at the bottom of the main window when the Rules Editor is open in foreground, provides a brief description of the toolbar commands currently pointed by the mouse cursor. In addition, on the right side you can see the number of the already defined Rules of the currently selected rule, and the current cursor position during Manual editing.

Fig. 6.17 - Rules Editor status bar



Guided Rules Editor

The Guided Rules Editor is the best way to define rules. It allows to write quickly the rules, without syntactical or typing error, supplying you all the admissible keyword, operators an names by means of an easily selectable drop-down list-box.

To start with the Guided Editor, open the Rules Editor and select the item OPEN from the RULES>GUIDED menu, or click the apposite button in the Rules Editor toolbar.

The Rules Editor window is modified in order to supply the controls for the guided editing. The edit-box to show the rule in editing is placed just below the Rules Editor toolbar. Another toolbar is added just below the edit-box, supplying the drop-down list-box for selecting keywords, operators and names; the other buttons perform the INSERT, the ABORT, the UNDO and the REDO commands.

The items listed in the drop-down list-box are updated after each selection, supplying the only syntactically admissible items. In such a way you are prevented to make mistakes.

Fig. 6.18 - Guided Rules Editor

To write the rules, select the items from the drop-down list: the rule in editing is shown in the apposite edit-box. End the rule editing by inserting it in the rules list clicking over the apposite button next to the drop-down list.

Notes in writing the rules:

- The keyword IF is automatically inserted when starting a new rule.
- The keyword IS is automatically inserted after the variables' names selection.
- The NOT modifier can be added before a M.F. name or after the keyword IS.
- Open parentheses are automatically closed when selecting the THEN keyword.
- Output variables are not included in the list if previously selected.
- The top of the list shows the first item of the list but you cannot select it from there, you should select the item from the dropped-down list.

After selecting an output variable, the top of the list becomes a text-box allowing you to specify a numeric value as crisp M.F. instead of selecting a defined M.F. name from the list. In such a way you can specify the crisp output M.F. directly in the rule without specifying them with the Membership Functions Editor.

To specify crisp M.F. Values:

1. Write the value in the top of the drop-down list-box after the selection of the output variable

2. Push the ENTER key

3. Continue the rule editing with the others consequent terms or insert the rule in the rule list

During the rule editing, it is possible to UNDO or REDO the last operation or ABORT the rule editing by using the apposite buttons next to the drop-down list, after the INSERT button.

Manual Rules Editor

Manual editing allows to edit rules by using a normal text-editor tool. This tool is particularly useful if you want to modify existing rules or to create new rules from the existing ones. Using Manual Editor you can also add comments to the edited rules and edit more than one rule at the same time.

Because writing rules with the free text editor it may be possible to make syntactical or typing mistakes, rules are checked before inserting them in the rules list.

To open the Manual Editor, select the item OPEN from the RULES>MANUAL menu, or click the apposite button in the Rules Editor toolbar. If you are writing a rule with the Guided Editor, you can switch to the Manual Editor and complete it manually.

The Rules Editor window is modified in order to supply the text-box where to edit the rules. You can use standard editing commands such as COPY, CUT, PASTE and DELETE using this text-box, by selecting them from the EDIT menu or by using the right mouse button pop-up menu. So you can copy rules from other rules list of different blocks of from other projects.

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 selecting the command CHECK from the RULESL>MANUAL menu or using or clicking the apposite button from the Rule Editor toolbar.

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.

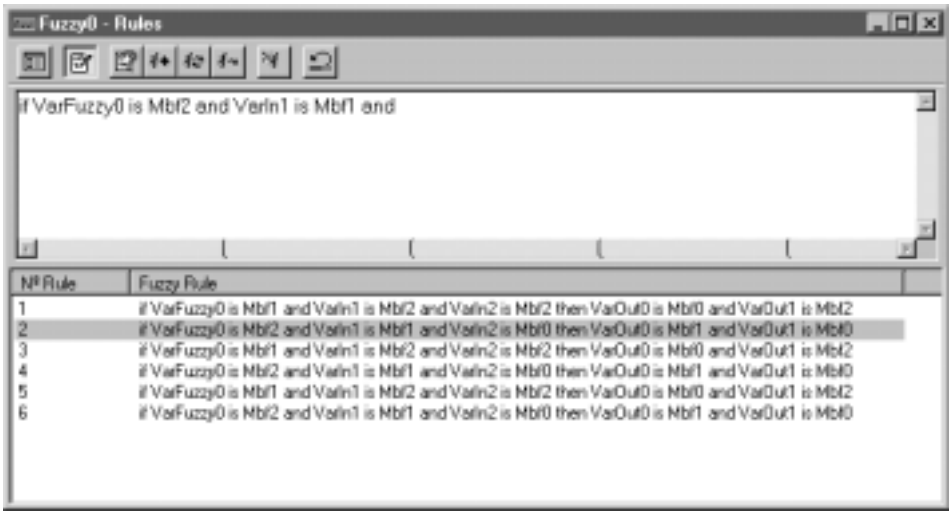
You can also write one or more rules by modifying the existing ones. In this way, the rule editing is performed quickly because the selected rule(s) is copied in the edit-box and it will not be necessary to write the entire rule(s) again:

- 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 with the copy command and then paste them in the Manual Editor windows.

- Modify the parts of the rule(s) you need to change.
- You can now check the syntactic correctness or insert the rule in the rules list

When an error occurs checking or inserting a rule, the Output Window opens displaying the error message. Double clicking over the error, the wrong rule is highlighted allowing you to easily detect the error.

Fig. 6.19 - Manual Rules Editor



Rules List updating

When the rule(s) definition has been completed, both with the Guided and Manual Editor, the rule(s) can be inserted in the rule list by means of the following commands in the RULES menu or by means of the corresponding button in the Rules Editor toolbar:

ADD: syntactically checks the current rule(s) and inserts it at the end of the rule list

INSERT: syntactically checks the current rule(s) and inserts it before the selected one

REPLACE: syntactically checks the current rule(s) and replaces the selected one(s)

Note: *If you have many selected rules the INSERT command places the new rule(s) just before the first selected one. When the REPLACE command is performed all the selected rules are deleted and substituted by the new one(s).*

Rules Editor Constraints

The rule's format depends on the input and output variables defined in the fuzzy system. Then rules can have up to 8 antecedent terms and as many consequent terms as the output variables.

The maximum number of rules that can be defined in the entire project depends on the available Program Memory space. Actually, fuzzy instructions and classical instructions share the same memory space. When the entire program exceeds the Program Memory space, the Compiler gives an appropriate error message and the code is not generated.

Rules Grammar

Use the following grammar when writing the rules:

FuzzyRule	::=	IF <Antecedent> THEN <Consequent> ;
Antecedent	::=	<Antecedent> OR <Antecedent> |
		<Antecedent> AND <Antecedent> |
		NOT <Antecedent> |
		(<Antecedent>) |
		<AntecedentAtom>
AntecedentAtom	::=	<InputVar> IS <Mbf> |
		<InputVar> IS NOT <Mbf>
InputVar	::=	<Identifier>
Mbf	::=	<Identifier>
Consequent	::=	<ConsequentAtom>|
		<Consequent> AND <ConsequentAtom>
ConsequentAtom	::=	<OutputVar> IS <Crisp>
OutputVar	::=	<Identifier>
Crisp	::=	<Identifier> | <Number>
Identifier	::=	[A-Za-z_] {([A-Za-z_] | [0-9])}*32
Number	::=	({<IntegerValue>} | {<RealValue>})
IntegerValue		::= ([0-9]+)
RealValue	::=	(([0-9]*\.[0-9]+){Exponent}?)
Exponent	::=	([eE][-+]?[0-9]+)

Rules Editor Error Messages

Using Manual Rules Editor the following messages and warnings can occur:

“char” not allowed character

The character “char” is not allowed in the context.

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 syntax 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: missing “)”

The rule in editing has not been completed before entering it because a close bracket is expected. Complete the rule with the correct syntax adding as many brackets as necessary.

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.

“then” keyword expected: found ...

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.

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.

Importing Fuzzy Systems

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

The fuzzy systems are described in Fu.L.L. (Fuzzy Logic Language) that is a descriptive language of the fuzzy systems allowing to exchange data between the various STMicrolectronics Fuzzy Logic software tools.

To import a fuzzy system in the project:

1. Insert an “Import From File” block selecting it from the Blocks editor toolbar; you act in the same way as for the other blocks.

2. When you insert the block in the client area, the standard Open dialog box appears allowing you to select the file .ful containing the description of the fuzzy system to be imported.

3. Select the file.

4. Press OK button. The Output window opens showing the success message or eventual error messages.

6. Complete the fuzzy system definition with the fuzzy variables initialization and storing.
Now the imported fuzzy system has the same characteristics of a normal fuzzy system and can be freely modified and inserted in the block diagram.

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 semantics refer to Appendix E. - F.U.L.L. - Fuzzy Logic Language.

7 - ARITHMETIC BLOCK

ARITHMETIC BLOCK

The Arithmetic Block allows to carry out the arithmetic and logic instructions of the device. Double clicking over the Arithmetic Block icon, the related editor opens. It is a free text editor where you can specify the instructions by means of a simple language, which is characterized by part of the instructions set and the syntax of the 'C' language. These instructions are translated during the compilation directly in FSCODE language, as the syntax is the same in the two representations.

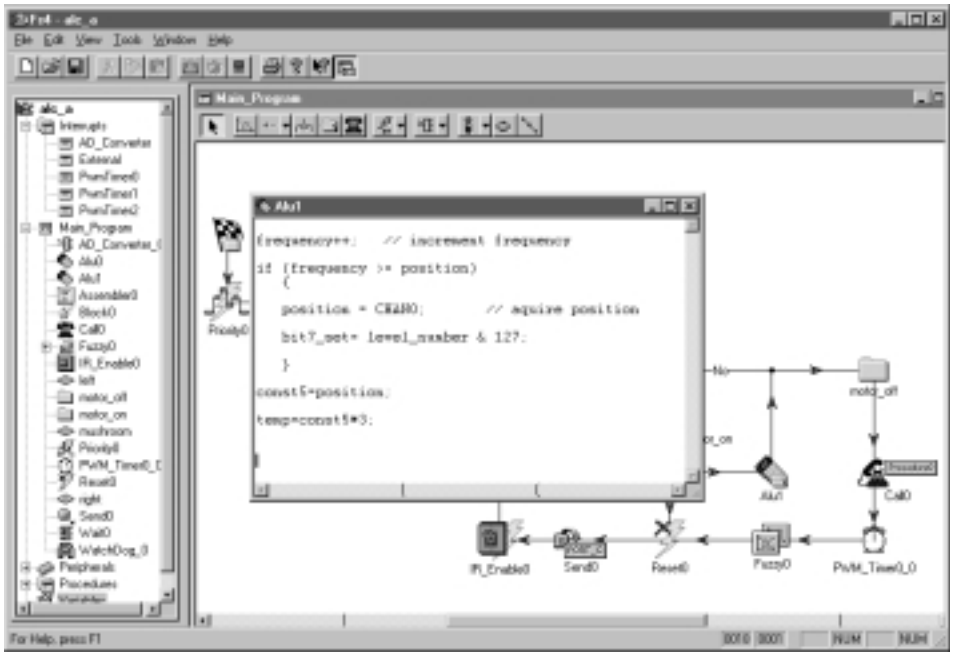
In this chapter you will learn to:

- use the Arithmetic Block editor
- the instructions set syntax
- write the program lines

Arithmetic Block Editor Window

This section provides an overview of the major elements of the Arithmetic Block Editor window such as menus and status bar.

Fig. 7.1 - Arithmetic Block Editor



Arithmetic Block Editor menus

The following menu items are available when the Arithmetic Block is open in foreground:

File Contains commands to create, open, close, save and print window contents and to visualize the project's information.

Edit Provides standard editing and Find & Replace commands.

View Contains commands to hide/show toolbars, status bar, Project and Output windows, to navigate along the error messages and to change the Fonts.

Tools Contains commands to run Debugger, Compiler or Programmer tools.

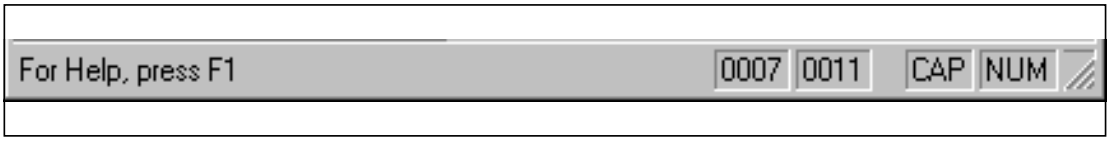
Window Contains commands related to window management.

Help Contains help commands.

Arithmetic Block Editor window status bar

The Status Bar displayed at the bottom of the Blocks Editor provides a brief description of the toolbar command currently pointed by the mouse cursor. In addition, on the right side, you can see the coordinates of the current cursor position.

Fig.7.2 - Arithmetic Blocks Editor status bar



Arithmetic Block Editor

The Arithmetic Block Editor is a normal text editor, like for example the Notepad, that allows you to write the instructions. The editor supplies you the standard editing commands such as:

- COPY
- CUT
- PASTE
- UNDO
- DELETE
- SELECT ALL

You can select these commands from the EDIT menu or from the pop-up menu opened by right clicking over the client-area or from the Main Window toolbar.

In addition the editor supplies the following standard commands:

- FIND
- FIND NEXT
- REPLACE

You can select these commands from the EDIT menu. Moreover you can change the fonts used for the text:

1. Select the FONT command from the VIEW menu

2. Select the desired font, font style and color and click O.K.

Arithmetic Block Instructions

The Arithmetic Block Instructions set is a subset of the ‘C’ language. In addition, some library functions suited to work with the microcontroller have been implemented. The Arithmetic Block instructions can be grouped as follows:

Assignment and mathematical operators:

= - + += -= ++ -- * / %

Logic operators:

& | ~ ^

Control structures and related statements:

if for while goto break continue

Logical functions for conditional expressions:

IsBitSet IsBitReset IsOverflow IsUnderflow DeviceStatus

Functions for bit manipulation:

BitSet BitReset BitNot << >>

Functions for peripherals and interrupts management:

DeviceSet IrqEnable IrqDisable IrqReset IrqEnableMask IrqPriority

Type conversion operators

.High .low

Writing instructions take in account the following rules:

- instructions must end with a semicolon
- instructions cannot have more than one operator (two operands)
- parentheses are not used in arithmetic instructions
- comment lines are allowed preceded, as in ‘C’ language, by the characters // or, if the comment is expressed in several lines, by the characters /* and ended with */.
- the allowed operands are the Global Variables and the Predefined Variables, constants and numeric values
- the keywords must be expressed in lower case
- for other remarks on writing the instruction, refer to the ‘C’ language grammar rules.

Global Variables Types and Cast

The instructions operate on the Global Variables, defined by the user with the apposite tool (see chap. 4: Initial Setting). In addition they operate on the Predefined Variable. You can find the Predefined Variables list for each device of the ST52 family in the Appendix A.

The Global Variables have a type associate; some of this types allow to use signed variables or 16-bits width variables. The available variable types are the following:

BYTE: 8-bit variables without sign, range [0 ,255]

S_BYTE: 8-bit signed variables, range [-128 , 127]

WORD: 16-bit variables without sign, range [0 , 65535]

S_WORD: 16-bit signed variables, range [-32768 , 32767]

Each 16-bit variable uses a couple of memory location to manage the values. The Compiler automatically generates the necessary instructions for managing the operation with the couple of bytes, because the devices of the ST52 family support only byte operations.

Signed values are managed by the Compiler adding a suited value to the logical one, in order to get unsigned values to be stored in the memory locations. So, to store SBYTE type variables, the value 128 is added to the logical value: -128 is stored as 0, 0 as 128 and 127 as 255. In the same way, to store S_WORD type variables, the value 32768 is added to the logical value. These operations are equivalent to set the MSB of the memory locations containing the variable value. Anyway, these operations are automatically performed by the Compiler, in a transparent way for the user.

Note: *Predefined Variables are considered as BYTE type variables.*

The Compiler manages all the conversion of type either in the casting or in the arithmetic operation between variables of different type. For example, the assignation of a variable to another one of different type is managed as follows: suppose that Var1 is S_BYTE type and Var2 is BYTE type and contains the value 100, then the instruction:

```
Var1 = Var2;
```

assigns logically the value 100 to the Var1 variable and the corresponding memory location will contain the value 228 (100 + 128).

If the variable Var2 value is 200, the instruction causes the memory location related to the Var1 variable to store the value 72 (200 +128 – 256), that logically means –56, and the Carry flag to be set. This case can be easily be managed with the function IsOverflow() put in a conditional construct just after the assign instruction; the functions returns true if the carry flag has been set (see later in this paragraph for further information on the IsOverflow function).

When operation between different types are performed, the transformation is automatically performed as follows: suppose to have the previously described Var1 and Var2 and another variable Var3 of S_BYTE type. The instruction:

```
Var3 = Var1 + Var2;
```

is managed automatically by a Compiler macro making the conversion of the operand. to perform the sum between values of the same type of the destination variable type.

The cast between types of different length is performed similarly by an assignment instruction. For example it is possible to copy the BYTE value of Var2 to the variable Word1 of type WORD by the instruction:

```
Word1 = Var1;
```

The viceversa is also possible but if Word1 contains a value greater of 255, the instruction return an overflow and the Var1 is loaded with the less significant byte of the variable Word1.

In some case it is necessary to copy the value contained in a 8-bit variable in the lower or the higher byte of a 16-bit variable. Vice versa it may be useful to store the higher or the lower part of a 16-bit variable in a 8 bit variable. For these reasons, the following syntax is supplied:

Var1**.high** = Var2;
Var1**.low** =Var2;
Var2 = Var1**.high**;
Var2 = Var1**.low**;
where:
Var1 is a 16-bit type variable
Var2 is an 8-bit type variable

Mathematical instructions

The supplied mathematical instructions are the sum, subtraction, multiplication, division and module. It is allowed to specify only one operator in each instruction. The operators are the following:

Var = op1 + op2;	sum between the two operands op1 and op2
Var += op;	equivalent to Var = Var + op;
Var = op1 - op2;	difference between the two operands op1 and op2
Var -= op;	equivalent to Var = Var - op;
Var = -op;	assign to Var the operand op with the sign changed
Var++;	increments the variable
Var - -;	decrements the variable
Var = op1 * op2;	multiplication between the two operands op1 and op2
Var = op1 / op2;	division between the two operands op1 and op2
Var = op1 % op2;	remainder of the division between the two operands op1 and op2

Note: *The operands can be variables or constants. The allowed constants range depends on the destination variable's type.*

It is not possible use variables of different size as operands; for example it is not possible to sum a WORD type variable with a BYTE type variable. You should transform one of the two variables, assigning it to a variable of the same type as the other variable, and perform the operation with the new variable.

The unary operator (-) for changing the variables value's sign, must be used with a signed type destination variable.

In multiplication instructions, the destination variable must be WORD or SWORD type. The operands must be BYTE or SBYTE type. If one of the two operands is SBYTE type, the destination variable must be of SWORD type.

In division and module instructions, the destination and operands must be unsigned type variables.

Logic instructions

The supplied logical instructions are the AND, OR, XOR and NOT. It is allowed to specify only one operator in each instruction. The operators are the following:

Var = op1 & op2;	computes the AND between the two operands op1 and op2
Var &= op;	equivalent to Var = Var & op
Var = op1 | op2;	computes the OR between the two operands op1 and op2
Var |= op;	equivalent to Var = Var | op
Var = op1 ^ op2;	computes the XOR between the two operands op1 and op2
Var ^= op;	equivalent to Var = Var ^ op
Var = ~op;	computes the NOT of the operand op

Note: *The operands can be variables or constants. Only unsigned variable types can be used with the logic instructions.*

Control Structures

The control structures allow to modify the logic flow of the program within the Arithmetic Block. The structures considered from the 'C' language instruction set are IF, WHILE and FOR.

The IF statement controls conditional branching. The body of an IF statement is executed if the value of the expression specified after the IF keyword is true (nonzero). The syntax for the IF statement has two forms:

- **if (expression) statement**
- **if (expression) statement else statement**

In the first form of the syntax, if expression is true (nonzero), statement is executed. If expression is false, statement is ignored. In the second form of syntax, which uses ELSE, the second statement is executed if expression is false. With both forms, control then passes from the if statement to the next statement in the program unless one of the statements contains a GOTO instruction (see below).
The statements are composed by a single instruction or several instruction lines enclosed between braces ({}). The expressions are composed by variables, constants, relational operators and logic operators; they return a TRUE (or nonzero) of FALSE (or zero).
The allowed relational operators are the following:

- == equality
- != inequality
- > greater than
- < less than
- >= greater or equal to
- <= less or equal to

The allowed logical operators are the following:

- && AND
- || OR
- ! NOT

The expressions use the variables, constants and operators following the standard 'C' syntax. Refer to any ANSI 'C' language reference manual for further information about the conditional expression syntax.

In addition, some library function, supplied by FUZZYSTUDIO™4.1 Compiler, can be used in the conditional expressions (see next paragraph).

Inside the body of the IF construct, it is often used the instruction GOTO to pass the control to another part of the program. The syntax of the GOTO instruction is the following:

goto label

where the label must be defined inside the block because its scope is inside the block. The label name must be followed by a colon (:) character.

The WHILE statement lets you repeat a statement until a specified expression becomes false.

The syntax is the following:

while (expression) statement

- The expression is a conditional expression as the ones described for the IF statement.
- The expression is evaluated.
- If expression is initially false, the body of the while statement is never executed, and control passes from the while statement to the next statement in the program.
- If expression is true (nonzero), the body of the statement is executed and the process is repeated.

The statement is composed by a single instruction or several instruction lines enclosed between braces ({}). The WHILE statement can also terminate when a BREAK or GOTO within the statement body is executed. Use the CONTINUE statement to terminate an iteration without exiting the while loop. The CONTINUE statement passes control to the next iteration of the FOR statement.

Similarly to the WHILE, the FOR statement allows you to repeat a statement until a termination condition becomes FALSE (zero). In addition an expression to initialize an index and an expression to increment such index are supplied. The syntax is the following:

for (init_statement ; conditional_expression ; expression) statement ;

The init_statement and the expression are assignment instructions or increment and decrement instructions or procedure calls.

The conditional_expression is a conditional expression as the ones described for the IF statement.

The statement is composed by a single instruction or several instruction lines enclosed between brackets ({}).

The FOR statement can also terminate when a BREAK or GOTO within the statement body is executed. Use the CONTINUE statement to terminate an iteration without exiting the while loop. The CONTINUE statement passes control to the next iteration of the FOR statement.

Logical functions for conditional expressions

To make easy working with the microcontroller, some library functions to be included in the conditional expressions are supplied within the Compiler. They allow to inspect the status of the single bits, to manage overflow and underflow and to manage some information coming from the peripherals. The functions return TRUE or FALSE according to the specified arguments. The available functions are the following:

IsBitSet(*bit,var*)	returns true if the bit No. *bit* of the variable *var* is set (1)
IsBitReset(*bit,var*)	returns true if the bit No. *bit* of the variable *var* is reset (0)
IsOverflow ()	returns true if last arithmetic instructions caused an overflow
IsUnderflow()	returns true if last arithmetic instructions caused an underflow
DeviceStatus (*periph, param*)	returns true if the event in the peripheral *periph*, specified by the parameter *param*, occurs. The list of the key to specify the arguments *periph* and *param* depends on the target device. You can find this list for each device in the Appendix A in this manual

Functions for Peripherals and Interrupts Management

In order to manage the peripherals and interrupts with the Arithmetic Block instructions, some library functions are supplied within the Compiler. They allow to enable or disable the peripherals and the interrupts and execute other actions related to their management. The available functions are the following:

DeviceSet(*periph, param,…***)**	allows to enable or disable and manage the peripherals
IrqEnable()	enables globally the interrupts (only the not-masked interrupt are enabled)
IrqDisable()	disables globally the interrupts
IrqReset(*int1, int2,….***)**	resets the specified pending interrupts
IrqEnableMask(*int1, int2,…***)**	enables selectively the interrupt sources and sets the external interrupt polarity (excluding ST52x420/420Gx)
IrqPriority(*int1, int2,…***)**	establishes the interrupts priority order

The functions parameters depends on the selected target. See Appendix A to get details about the implementation of the functions for each target device.

Note: *The same actions can be programmed using the Peripherals Setting blocks and the Interrupts related blocks.*

Functions for bit manipulation

The devices of the ST52 family do not support instructions for bit manipulations. Anyway, these can be performed using other device instructions. The Compiler supplies the functions that implement these operations:

BitSet(*bit , var*)	sets the bit No. *bit* of the variable *var* to 1
BitReset(*bit, var*)	resets the bit No. *bit* of the variable *var* to 0
BitNot(*bit , var*)	complements the bit No. *bit* of the variable *var*

The following syntax is used to perform shift operations:

Var1 = Var2 << op; left shift

Var1 = Var2 >> op; right shift

Var1 and Var2 are unsigned 8 or 16-bit variables

op is a byte type variable or a constant that indicates the number of times to perform the shift.

Tables and Constants

Tables values and constants can be used in the arithmetic instruction as read-only variables: they cannot be specified as destination. The tables and constants are defined with the Tables window editor accessed through the Project Window (see Chapter 4).

Tables and constants consider the same convention for types and cast as described for the variables. The tables elements are specified using the indexes enclosed between square brackets. The option base is always 0. They are practically considered as vectors of constants.

Examples:

Var0 = Table0[23] ;

Var1 = Table2[2] – Table1[0];

Var2 = Constant0 + Table0[1];

8 - ASSEMBLER BLOCK

ASSEMBLER BLOCK

The Assembler Block allows to program routines at low level. Some parts of a program could need fast response time that can be obtained only with an optimized code. The use of the Assembler instructions is a better solution in these cases. The Assembler Block uses the Assembler instructions of the ST52 family, with some exceptions such as the fuzzy instructions.

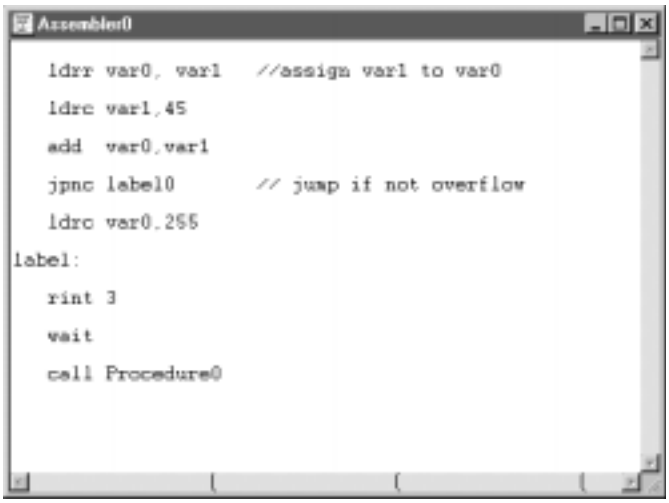
In this chapter you will learn:

- to use the Assembler Block editor
- the instructions set syntax
- write the program lines

Assembler Block Editor Window

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

Fig. 8.1 - Assembler Block Editor



Assembler Block Editor menus

The following menu items are available when the Assembler Block is open in foreground:

File Contains commands to create, open, close, save and print window contents and to visualize the project's information.

Edit Provides standard editing, Find & Replace commands.

View Contains commands to hide/show toolbars, status bar, Project and Output windows, to navigate along the error messages and to change the Font.

Tools Contains commands to run Debugger, Compiler or Programmer tools.

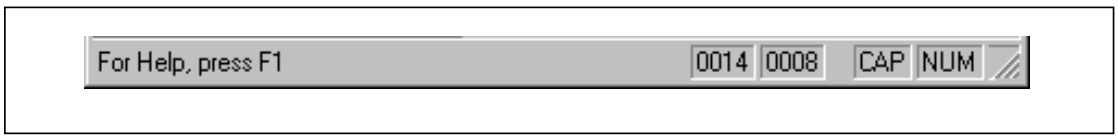
Window Contains commands related to window management.

Help Contains help commands.

Assembler Block Editor window status bar

The status bar displayed at the bottom of the Assembler Blocks Editor provides the coordinates of the current cursor position.

Fig. 8.2 - Assembler Block Editor status bar



Assembler Block Editor

Such as the Arithmetic Block, the Assembler Block Editor is a normal text editor, like for example the Notepad, that allows you to write the instructions. The editor supplies you the standard editing commands such as:

- COPY
- CUT
- PASTE
- UNDO
- DELETE
- SELECT ALL

You can select these commands from the EDIT menu or from the pop-up menu opened by right clicking over the client-area or from the Main Window toolbar.

In addition the editor supplies the following standard commands:

- FIND
- FIND NEXT
- REPLACE

You can select these commands from the EDIT menu.

Moreover you can change the fonts used for the text:

- Select the FONT command from the VIEW menu
- Select the desired font, font style and color and click O.K.

Assembler Block Instructions

The Assembler instructions set is a subset of the one supplied for the ST52 family. The Fuzzy Logic instructions and few others are excluded because they may generate some overlapped code with the one generated by the Compiler. The list of the available instructions is showed in the following tables:

LOAD INSTRUCTIONS		
ldcr	*REG_CONFxx , var*	loads the configuration register *xx* with the variable contents
ldpr	*pred, var*	loads the peripheral register with the variable contents
ldrc	*var, const*	loads the variable with the specified constant
ldri	*var, pred*	loads the variable with the peripheral register contents
ldrr	*var, var*	loads the variable with another variable contents

ARITHMETIC INSTRUCTIONS		
add	*var, var*	sum between two variables
addo	*var, var*	sum with offset between two variables
and	*var, var*	bitwise AND between two variables
asl	*var*	arithmetic shift left of the variable
asr	*var*	arithmetic shift right of the variable
dec	*var*	decrement the variable
div	*word, var*	division between a word type variable and a byte type variable
inc	*var*	increments the variable
mult	*word, var*	multiplication between the low part of the word type variable and a byte type variable
not	*var*	bitwise NOT of the variable
or	*var, var*	bitwise OR between two variables
sub	*var, var*	subtraction between two variables
subo	*var, var*	subtraction with offset between two variables
mirror	*var*	mirroring of the variable contents

JUMP INSTRUCTIONS		
call	label	call user procedure
jp	label	absolute jump
jpc	label	jumps if the carry flag is set
jpnc	label	jumps if the carry flag is reset
jpns	label	jumps if the sign flag is reset
jpnz	label	jumps if the zero flag is reset
jps	label	jumps if the sign flag is set
jpz	label	jumps if the zero flag is set

INTERRUPT RELATED INSTRUCTIONS	
halt	puts the device in halt state
mdgi	disables the interrupt (used by the Compiler macro)
megi	enables the interrupt (used by the Compiler macro)
reti	returns from interrupt
rint *n*	resets interrupt number *n*
udgi	disables the interrupt (used by the user)
uegi	enables the interrupt (used by the user)
waiti	waits for interrupt

MISCELLANEOUS	
nop	no operation
wdtrfr	refresh the Watchdog counter
wdtslp	stop the Watchdog

To get more information about Assembler instructions see Appendix C in this manual.

Using the Assembler instructions in the Assembler Block you should follow these indications:

- Memory locations are addressed by using Global and Predefined Variables: it is not allowed to specify the direct address.
- The Configuration Registers are addressed by using the REG_CONFxx Predefined Variables.
- The data are sent to the peripheral by using the appropriate writable Predefined Variable with the **ldpr** instruction.
- Data from peripheral are stored in the ram by using the appropriate Predefined Variable with the **ldri** instruction.

- Only byte type variables can be used with the Assembler instructions, with the exception of the **mult** and the **div** instructions.
- The **mult** instruction in the Assembler Block is different from the one in the device instruction set, because a word type variable must be specified as destination register. Only the less significative byte of the variable is considered as first operand of the multiplication. The result is put on the whole word type variable.
- The first operand of the **div** instruction must be a word type variable. The resulting quotient is put in the less significative byte of the word type variable, the remainder in the most significative one.
- The jump instruction must specify a label defined inside the same Assembler Block.
- The call instruction must specify an existing user procedure name defined within the Project Window.

9 - CONDITIONAL BLOCK

CONDITIONAL BLOCK

The Conditional Block allows to modify the logic flow of the program according to a specified condition operating on the Global and Predefined Variables. The conditional block is connected to one input and two output links, tagged with YES and NO, which are connected respectively to the part of the program to be executed if the condition is TRUE and to the part to be executed if the condition is FALSE.

In this chapter you will learn how to:

- use the Conditional Block editor
- write the conditions

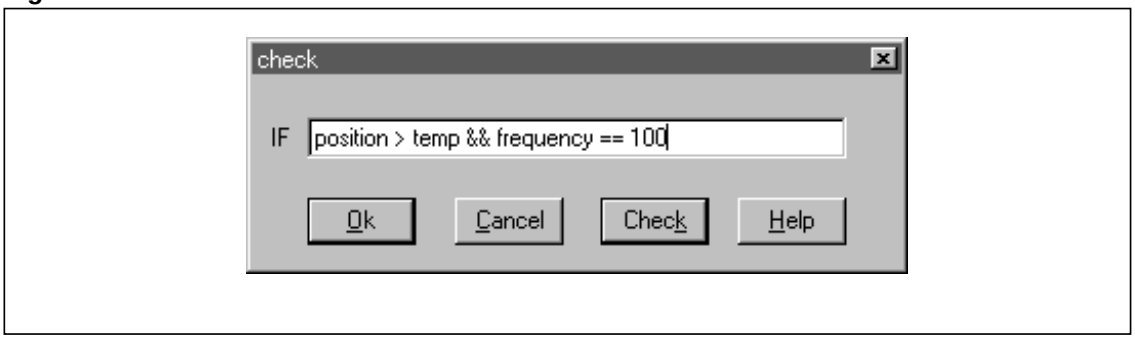
Conditional Block Editor

The Conditional Block Editor is a normal text editor, like for example the Notepad, that allows you to write the instructions. The editor provides you the standard editing commands such as:

- COPY
- CUT
- PASTE
- DELETE
- UNDO

You can select these commands from the EDIT menu or from the pop-up menu opened by right clicking over the client-area or from the Main Window toolbar.

Fig. 9.1 - Conditional Block Editor



Conditional Block Grammar

The Conditional Block's Editor allows to write the condition that determines the logic flow of the program. The condition can be written as for the Arithmetic Block's instructions, using the same grammar.

The Conditional Block instruction is equivalent to the conditional construct IF. The keywords IF is omitted, so you have to specify only the condition. If the expression representing the condition is true (nonzero), the control of the program is passed to the block connected to the YES link. If expression is false, the control passes to the block connected to the NO link.

The expressions are composed by variables, constants, relational operators and logic operators; they return a TRUE (or nonzero) of FALSE (or zero).

The allowed relational operators are the following:

- == equality
- != inequality
- > greater than
- < less than
- >= greater or equal to
- <= less or equal to

The allowed logical operators are the following:

- && AND
- || OR
- ! NOT

The expressions use the variables, constants and operators following the standard 'C' syntax. Refer to any ANSI 'C' language reference manual for further information about the conditional expression syntax.

In addition, some library functions, supplied by FUZZYSTUDIO™4.1 Compiler, can be used in conditional expressions. They allow to inspect the status of the single bits, to manage overflow and underflow and to manage some information coming from the peripherals. The functions return TRUE or FALSE according to the specified arguments. The available functions are the following:

IsBitSet(*bit,var*);	returns true if the bit No. *bit* of the variable *var* is set (1)
IsBitReset(*bit, var*);	returns true if the bit No. *bit* of the variable *var* is reset (0)
IsOverflow();	returns true if last arithmetic instructions caused an overflow
IsUnderflow();	returns true if last arithmetic instructions caused an underflow
DeviceStatus(*periph ,param*);	returns true if the event in the peripheral *periph*, specified by the parameter *param*, occurs. The list of the key to specify the arguments periph and param depends on the target device. You can find this list for each device in the Appendix A in this manual.

10 - BLOCKS FOR PERIPHERALS MANAGEMENT

FUZZYSTUDIO™4.1 Blocks Editor supplies the blocks to easily interact with the peripherals. To exchange data with the peripherals you can use the *Send* and *Receive Block*s. The Peripherals Blocks are used to manage the peripherals operations. There is one Peripherals Block for each peripheral present on the target device. See Appendix A to learn about the Peripherals Blocks supplied for each target device.

In this chapter you will learn how to use:

- the *Send Block*
- the *Receive Block*
- the *Peripherals Block*

Send and Receive Block

The *Send Block* is used to transfer data to a specified peripheral as for example, the counter value of the Timer. The *Receive Block* is used to read data from a peripheral such as for example, the values from the A/D Converter channels. To do that you have to specify one Global or Predefined variable and one peripheral device.

The peripherals are identified by the Predefined Variables: in the Send Block you can find the write only or the write/read Predefined Variables as destination; in the Receive Block you can find the read only or the read/write Predefined Variables as source.

Send Block

The Send Block editor is composed by:

- the tree-list containing the available Global and Predefined Variables, Tables and constants to be used as source.
- the drop-down list box containing the available Predefined Variables representing the destination devices. Read-only Predefined Variables are not included in the list.

Operate as follows:

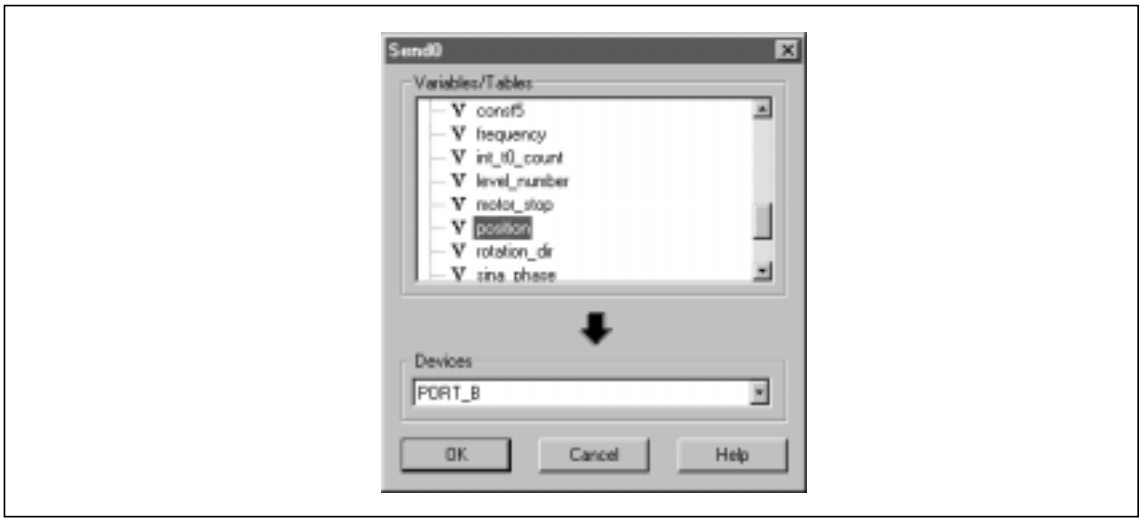
1. Select the source variable or the Table element from the tree-list

2. Select the destination Predefined Variable from the drop-down list

3. Click OK button

The value contained in the source variable is sent to the peripheral device buffer identified by the selected Predefined Variable.

Fig. 10.1 - Send Block Editor



Receive Block

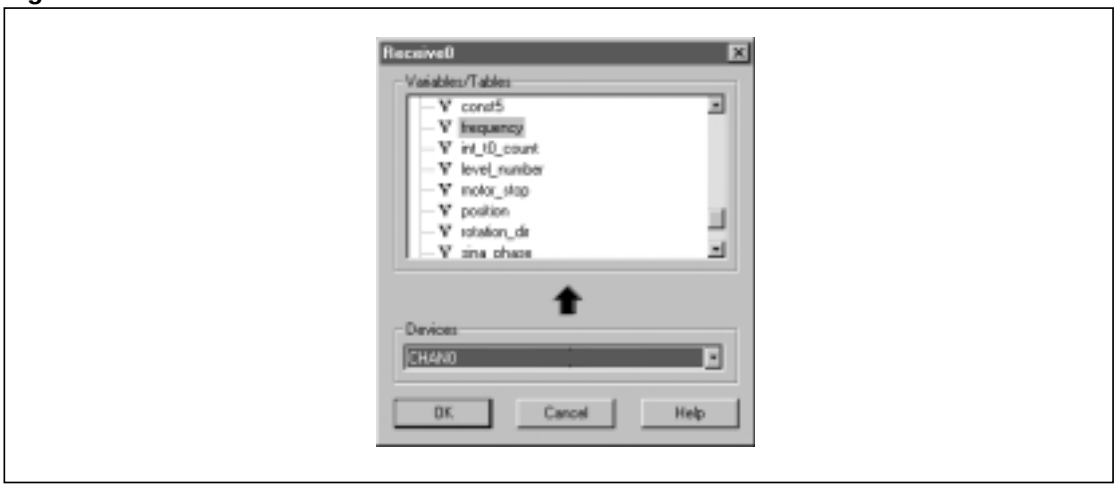
The Receive Block editor is composed by:

- the tree-list containing the available Global and Predefined Variables to be used as destination.
- the drop-down list box containing the available Predefined Variables representing the source devices. Write-only Predefined Variables are not included in the list.

Operate as follows:

1. Select the destination variable from the tree-list.

Fig. 10. 2 - Receive Block Editor



2. Select the source Predefined Variable from the drop-down list.

3. Click OK button.

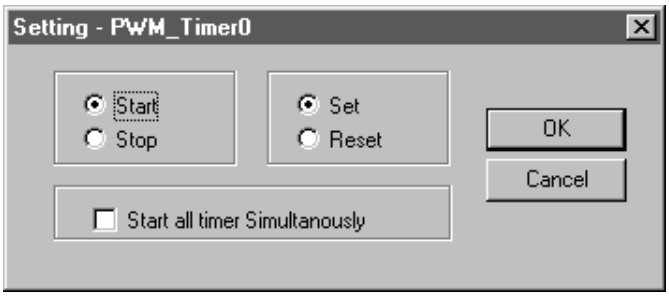
The value read from the peripheral device buffer, identified by the selected Predefined variables, is stored in the selected destination variable.

Peripherals Blocks

The Peripherals Block is a group of blocks used to operate run-time on the peripherals. It is used mainly to start or stop the peripherals; in addition, some peripherals' configurations can be changed run-time.

The number and the type of Peripherals Blocks depend on the peripherals included in the target device. The related editors' environments are suited to the peripheral type. Check-boxes, list-boxes, and other controls allow you to program easily the action to be executed by the peripheral. The Peripherals Blocks for each target device are described in the Appendix A of this manual.

Fig. 10.3 Peripherals Block editor example



Note: *Each Peripherals Block has its own icon representing the considered peripheral; also the default label name recalls the peripheral type.*

11 - INTERRUPTS RELATED BLOCKS

In this chapter, the groups of blocks used to manage the interrupts are described. They allow to perform the following operations:

- Enable globally the interrupts
- Disable globally the interrupts
- Reset the pending interrupts
- Enable/disable (mask) the interrupts
- Set the interrupts priority levels

The associated editors are similar in each target device, with the difference that the interrupt sources to be considered change in each device. See Appendix A to know the interrupt sources included in each target device.

In this chapter you will learn to use:

- the Interrupts Enable Block
- the Interrupts Disable Block
- the Interrupts Reset Block
- the Interrupts Mask Block
- the Interrupts Priority Block

Interrupts Enable Block

The Interrupts Enable block is used to globally enable the interrupts. Inserting this block the not-masked interrupts can be serviced. This block does not modify the mask of each interrupt source, so the disabled interrupt sources will be not be acknowledged remaining pending.

This block has not an associated editor because no specification is needed: the related action is completely specified just inserting the block in the diagram.

Note: *you do not need to insert the Interrupts Enable Block at the beginning of the program because the interrupts are globally enabled by default. You have to use this block after using the Interrupts Disable Block, if you want to enable the interrupts again.*

The Interrupts Enable Block corresponds to the insertion of the UEGI Assembler instruction.

Warning: During the execution of FUZZYSTUDIO™4.1 instructions or single-action blocks, the interrupts are disabled by using the apposite Assembler instruction MDGI. So the not-masked interrupts cannot be serviced, until the MEGI instruction has been specified at the end of the high-level instruction execution, although the Interrupt Enable Block has been inserted. You can avoid this by using the Folder block properly configured (see chapter 12).

Interrupts Disable Block

The Interrupts Disable Block is used to disable globally the interrupts. Inserting this block the interrupts cannot be serviced until an Interrupts Enable Block is executed. This block does not modify the mask of each interrupt source so that the not-masked interrupts can be serviced after the execution of the Enable Block.

This block has not associated editor because no specification is needed: the related action is completely specified just inserting the block in the diagram.

The Interrupts Disable Block corresponds to the insertion of the UDGI Assembler instruction.

Interrupts Reset Block

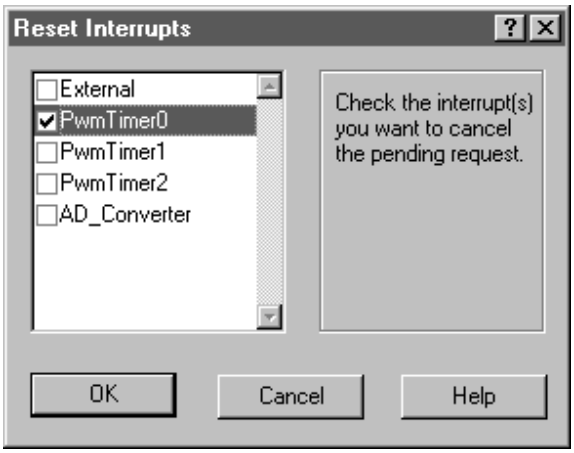
If an interrupt request occurs when the interrupt is masked, it is not serviced and remains pending; after enabling the interrupt, it is immediately serviced. This may be an unwanted event: for this reason it is supplied the way to cancel the pending interrupt requests before enable them.

The Interrupts Reset Block allows you to select the pending interrupt(s) to be reset. The editor is composed by a check list-box which items are the available interrupt sources of the selected target. See Appendix A to know the list of the available interrupts for each device.

To reset the pending interrupt request(s):

1. Check the interrupt source(s) in the check-list by clicking the corresponding check-box(es)
2. Click OK button

Fig. 11. 1 - Interrupt Reset Block editor



Interrupts Mask Block

The Interrupt Mask Block is used to enable or disable the interrupt sources selectively. When disabled (masked), the interrupt source cannot be acknowledged remaining pending; it will be serviced as soon as it is enabled.

The editor is composed by a check list-box showing the available interrupt sources of the selected target. See Appendix A to know the list of the available interrupts for each device.

To enable interrupt(s):

1. Check the interrupt source(s) in the check-list by clicking the corresponding check-box(es).

2. Click OK button.

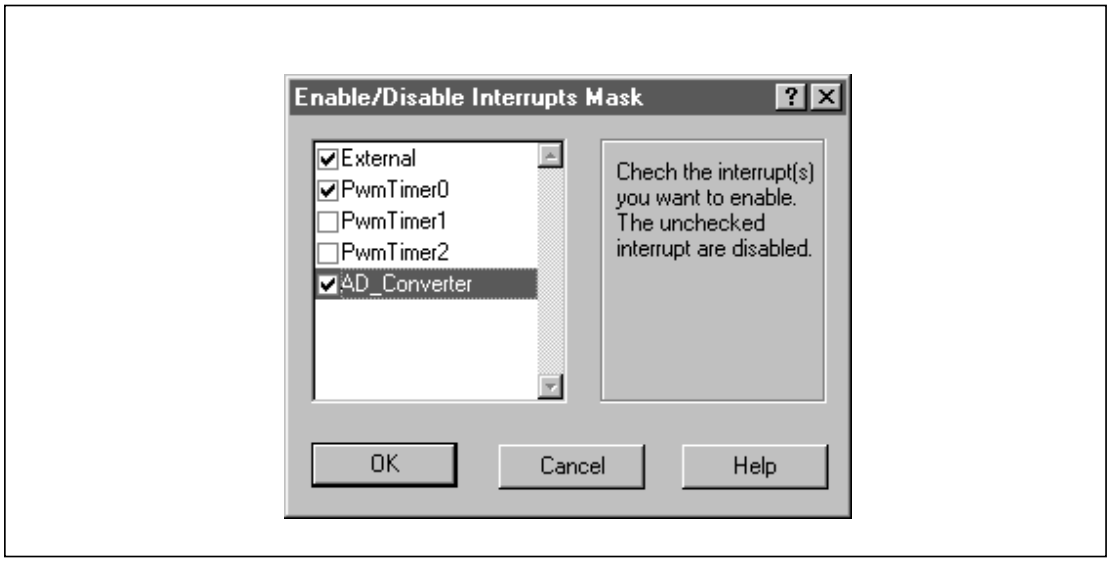
To disable interrupt(s)

1. Leave unchecked the interrupt items to be disabled.

2. Click OK button.

Note: *When the check-boxes are checked the interrupts are not masked, when unchecked they are masked.*

Fig. 11.2 - Interrupt Mask Block editor



Interrupts Priority Block

When more interrupts occur simultaneously or when an interrupt occurs during the execution of an interrupt service routine, the decision about which interrupt has to be serviced is taken according to the priority level of the active interrupts.

FUZZYSTUDIO™4.1 Blocks Editor supplies the Interrupts Priority Block to establish the priority order. The editor is composed by a list-box which shows the available interrupt sources of the selected target, listed with the currently set priority order. See Appendix A to know the list of the available interrupts for each device.

To change the priority level of an interrupt source you have to change its position in the list:

1. Select the interrupt source by clicking on it in the list

2. Keeping the mouse button down, drag the item to the desired position

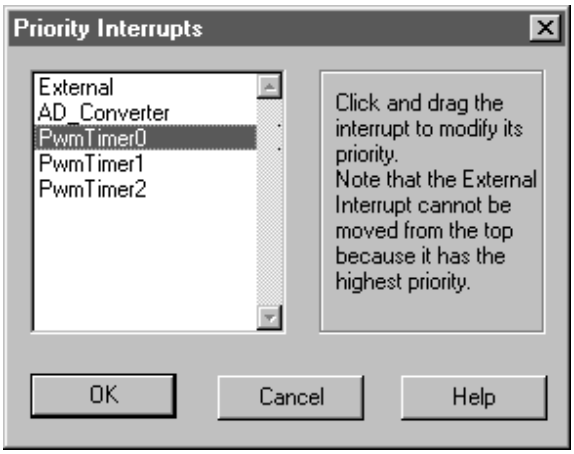
3. Change the position of the other interrupts which you want to modify the priority level

4. Click OK button

Note: *The ST52 family devices may have interrupt(s) with fixed priority: in most cases it is the External Interrupt that has top level fixed priority. The relative item(s) in the Interrupt Priority Block list cannot be moved, keeping always the position related to its fixed priority level (usually the top of the list).*

Warning: Due to the architecture of the ST52x420, when using this device as target, it is recommended to use the priority block only at the beginning of the program, before starting any interrupt source because, in some cases, unwanted interrupts may be serviced.

Fig. 10.3 - Interrupt Priority Block editor



12 - OTHER BLOCKS

In this chapter, the remaining blocks not yet presented are described. These blocks are not less important or less used, but they are grouped in the present chapter because they are used for particular and special functions. They are supplied to implement some important features of the microcontroller or to improve the readability of the block diagram. Some of them have not an associated editor or do not generate any instruction.

In this chapter you will learn to use:

- Call Block
- Wait Block
- Halt Block
- Restart and Return Blocks
- IRQ and RETI Blocks
- Folder and Exit Blocks

Call Block

CALL

The Call Block is used to implement the user's procedures calls. We have already described how to implement procedures in the Chapter 3 with the Project Window tree-view. The Call Block allows you to use these procedures inside the program.

The procedures are part of program that can be recalled several times in a different part of the program. The procedures allow to save memory space and to make the program more compact and readable. There are not parameters to be specified in the procedure calls and there are not local variables inside the procedure scope: data are passed to the procedures by means of the Global Variables. The procedures are formally subroutines.

The Call Block editor allows to specify the user procedures to be called. It is composed by a list-box with all the available user procedures and by a text-box indicating the previously selected procedure name (if not selected yet it is specified "none").

To select the procedure to call:

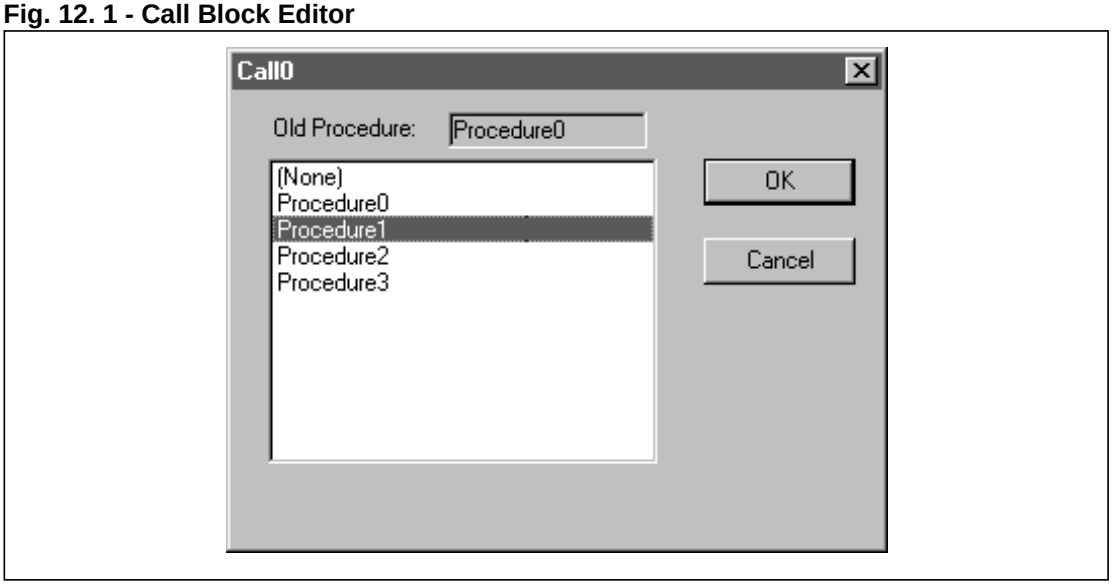
1. Click on the procedure name in the list-box

2. Click OK button or double-click on the procedure name

The Call Block icon is modified showing a tag with the selected procedure name. You can hide this tag selecting the command "Hide Procedure Label" from the pop-up menu opened right-clicking the block's icon.

Note: *Deleting a procedure, you are requested for confirmation twice: the second time you are informed about the Call Blocks that are using the procedure, if any. These blocks are not deleted but they loose the association with the old procedure and are set to "none".*

Fig. 12. 1 - Call Block Editor



Wait and Halt Blocks

These blocks are used to put the device in the Wait or Halt state. These blocks have not associated editor because no specification is needed: the related action is completely specified just inserting the block in the diagram.

WAIT

The Wait Block stops the execution of the programs and the CPU clock until the occurrence of an enabled interrupt. The peripherals and the oscillator remain active. The Wait Block corresponds to the WAIT Assembler instruction.

Warning: it is possible to exit from the Wait state only if the interrupt is serviced. If the request is not serviced because the interrupt is not enabled or the interrupts are globally disabled or the Wait block has been inserted in a too high priority interrupt routine, the processor will never exit from the Wait state. Example: if you use the Wait Block in the top level interrupt service routine you will never exit from the Wait state until a reset occurs.

HALT

The Halt block stops the program execution, the CPU clock, the peripherals and the oscillator in order to have the lowest current consumption. Only the External Interrupt request can wake-up the microcontroller from the Halt state. If the External Interrupt is enabled then the related service routine is executed; otherwise the instruction after the Halt is executed. The Halt Block corresponds to the HALT Assembler instruction.

Note: *If the Watchdog is enabled the Halt instruction is skipped and the Halt state is not entered.*

Restart and Return Blocks

RESTART

The Restart Block is used to implement a closed loop in the program or to restart particular conditions without resetting the device. Actually the Restart Block determines an absolute jump to the beginning of the Main Program.

The Restart Block has not associated editor because no specification is needed: the related action is completely specified just inserting the block in the diagram.

Note: *the Restart Block jumps to the beginning of the program without reconfiguring the peripherals.*

RETURN

When using a Procedure's Blocks Editor, the Restart Block is replaced by the Return Block. It performs the Return from the Procedure, passing the control to the instruction following the Call Block that called the Procedure. Also the Return Block has not associated editor. It corresponds to the RET Assembler instruction.

Note: *You must insert at least one Return Block in each Procedure's block diagram.*

IRQ and RETI Blocks

When you are using the Block Editor in the interrupt service routines, the Start Block and the Restart Block are replaced in the toolbar respectively by the IRQ Block and the RETI Block.

These blocks have not an associated editor: the related action is completely specified just inserting the block in the diagram.

IRQ

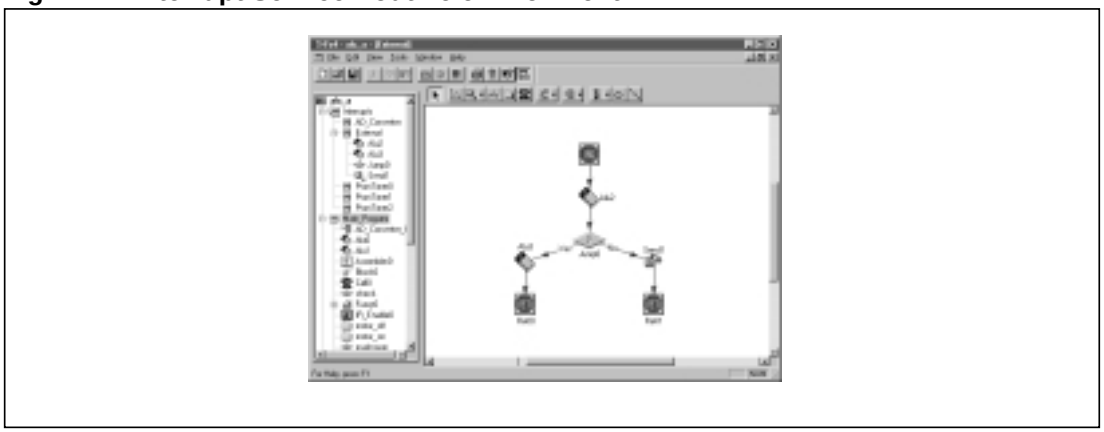
The IRQ Block states the beginning of the interrupt service routine; as the Start Block, the IRQ block is inserted automatically and cannot be deleted. It can have only one output link attached.

Note: *The IRQ Block does not correspond to any instruction or functionality of the microcontroller: it is used only to indicate the beginning of the block diagram.*

RETI

The interrupt service routine is completed with one or more exit points inserted in the block diagram by using the RETI block. This block determines the return from interrupt and the passing of the control to the instruction next to the interrupted one. The RETI block corresponds to the RETI Assembler instruction.

Fig. 12.2 - Interrupt Service Routine environment



Note: *You must insert at least one RETI Block in each interrupt service routine block diagram that you are using in your program.*

Folder Block and Exit Block

FOLDER

The Folder Block does not implement any microcontroller's functionality: it is useful just to organize better your block diagram, improving the program organization and readability. Actually, the block-diagram may become very complex and tangled. The Folder Block allows you to group some part of your program inside a sub-diagram contained inside the Folder, without logically modifying the entire block-diagram, but just for viewing and organization purposes. In addition, it allows to specify some parts of the program that are to be compiled in a special way.

The Folder Block can be inserted as the other blocks. Double clicking on the icon, the Folder environment opens. It is a Block Diagram editor similar to the one of the Main Program, with the difference that one or more exit points should be inserted at the end of the sub-diagram by using the Exit Block.

EXIT

The Exit Block does not implement any microcontroller's functionality. It indicates the link of the sub-diagram to be connected with the output link of the Folder Block. The Exit Block replaces the Restart Block in the Folder's Blocks Editor. The Folder's input link is connected to the Folder's Start Block.

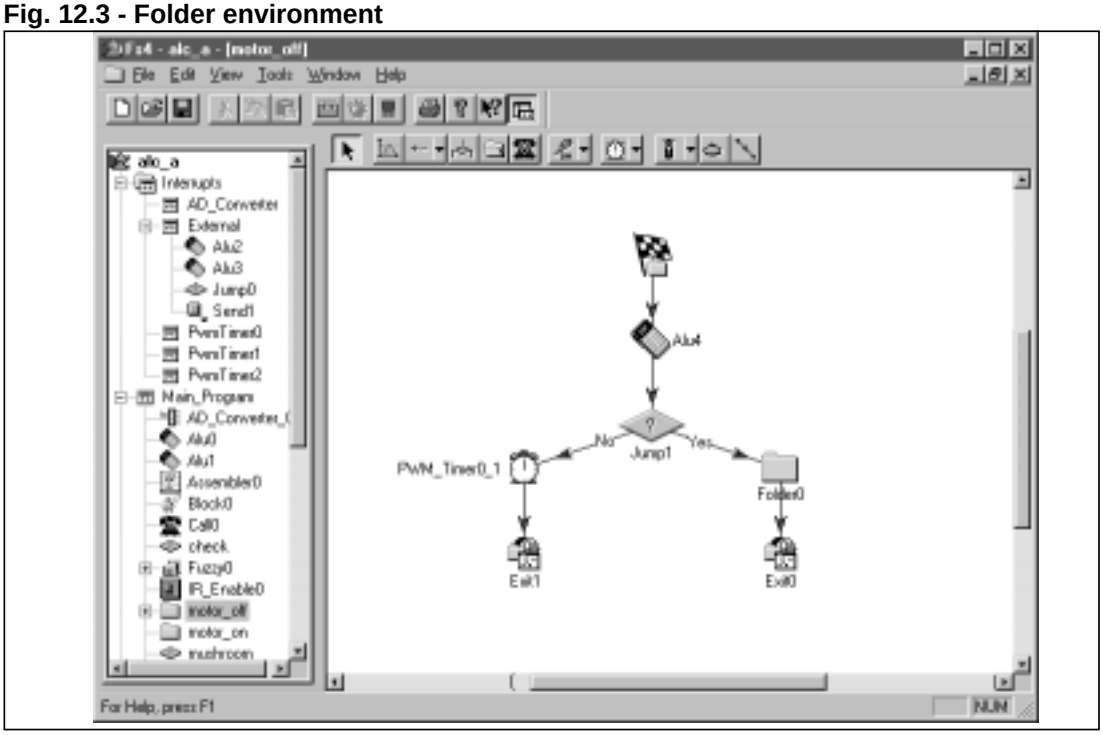
To operate with the Folders:

1. Insert the Folder Block (refer to Inserting Blocks paragraph in chapter 5)

2. Double-click on its icon to open the Folder

3. Design the block sub-diagram specifying the commands and end it with at least one Exit block

Fig. 12.3 - Folder environment



Folders with Compiler options

The Folder block allows also to group some parts of programs that should be compiled in a special way. There are three types of Folders according with the settings: Safe, Locked and Unlocked.

LOCKED

These compilation modality regards the introduction of the MDGI and MEGI Assembler instructions inside the generated code in order to prevent the Compiler macro from interrupt.

Actually to each high-level instruction corresponds a set of Assembler instruction i.e. a macro. The macro uses temporary variables shared between all the macros and that contains temporary values valid only during the macro execution. If an interrupt occurs during the macro execution, these temporary values are lost and the macro cannot be completed correctly.

UNLOCKED

For this reason, the macro starts with a MDGI instruction that disables the interrupt protecting the macro, and ends with the MEGI Assembler instruction that enables the interrupts again. This causes the increasing of program length and the lowering of the execution speed.

Some parts of programs may do not need this features because the interrupts are not enabled or the macro can be interrupted without any problem. In this case, it is possible to use the Locked or Unlocked Folders types.

The three types of folder work as follows:

Safe: it is the normal type of folder as described previously. The code is generated as in the Main Program with the instruction MDGI and MEGI inserted at the start and the end of each macro. This is the safest mode but it is the most space and time consuming.

Locked: in this mode the interrupts are disabled only at the start and enabled at the end of the Folder. So you are prevented by unwanted interrupts and the code is optimized because the MEGI and MDGI are specified just one time in all the Folders.

Unlocked: this is the less safe type because the MEGI and MDGI instructions are never specified inside the Folder. Use this block if the interrupts are disabled or not used or if the interrupt service routines do not use high-level instructions.

To set the Folder type act as follows:

1. Insert the Folder block as for the other blocks

2. Right-click over the block to open the pop-up menu

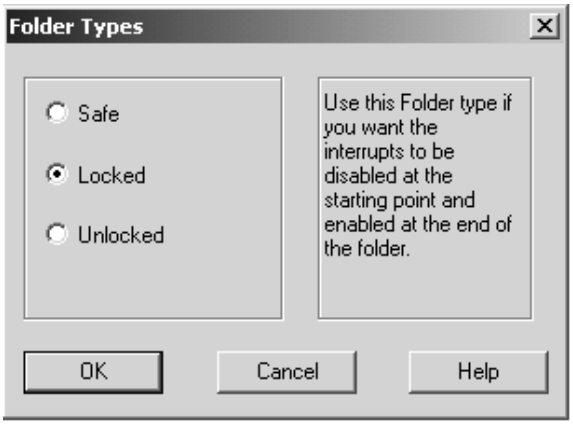
3. Select the SETTING PROPERTIES command: the Folder Type dialog-box opens

4. Choose the Folder type by clicking the corresponding radiobutton.

5. Click O.K. button.

Note: *Be careful when using nested Folders with different compiler options: the code is generated with the modality corresponding to the options set for the top-level folder (only locked or unlocked type). The code related to the nested folders will be generated using that modality, despite the option set.*

Fig. 12.4 - Folder types dialog box



13 - COMPILER

COMPILER

After ending the program writing, you can generate the machine code to be loaded in the device's memories by using the *Compiler* tool. The Compiler also generates other files used by the FUZZYSTUDIO™4.1 tools.

The results of the compilation are showed in the Output Windows, which allows you to interact easily with the program in case of compilation errors.

You can also customize the compilation operation by specifying the Compiler options, in order to get the type of compilation you need for your project.

In this chapter you will learn to:

- Launch the Compiler
- Check for compilation errors
- Set the Compiler options

In addition, in this chapter you will find the list of the Compiler error messages.

Project Compilation

After setting the Compiler options if needed (see next paragraph), you can run the Compiler and check for eventual errors.

To launch the compilation:

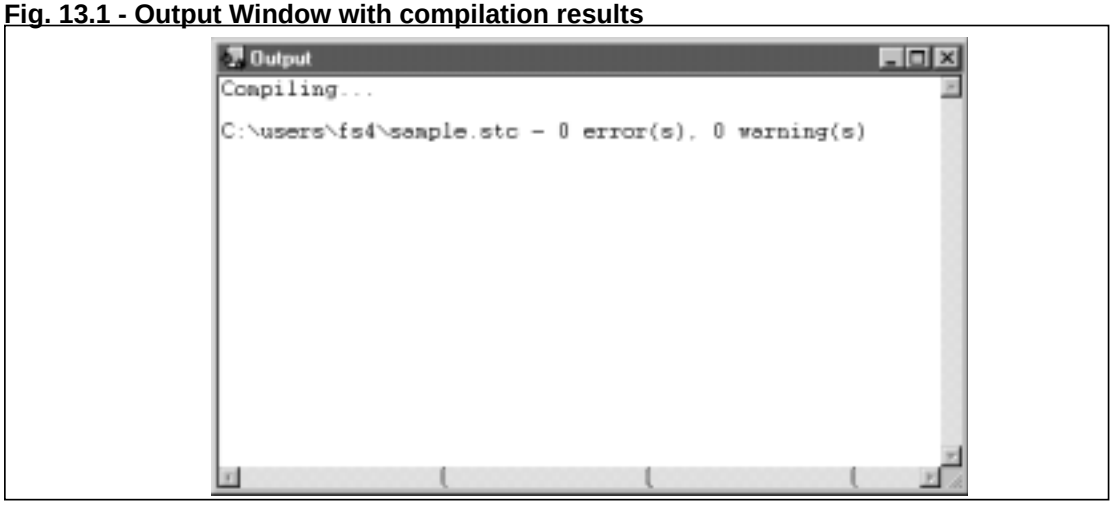
- select the COMPILER>RUN command from the TOOL menu

  or

- click the apposite toolbar button in the Main window.

The Output Window opens showing the result of the compilation: the error messages and warnings or the message of compilation success. You can access to the instruction

Fig. 13.1 - Output Window with compilation results



that caused an error by double-clicking on the error message in the Output Window. The wrong instruction is put in foreground and highlighted.

You can navigate along the error messages by using the Next Error and Previous Error commands:

To inspect the next error in the errors list:

- select NEXT ERROR command from the VIEW menu of the main window

  or

- push F4 key

To inspect the previous error in the errors list:

- select PREVIOUS ERROR command from the VIEW menu of the main window

  or

- push SHIFT+F4 key

Files generated during compilation

The Compiler generates some intermediate files before the final code. They are in the order:

- the script file describing the program in FSCODE language with extension .STC
- the corresponding assembler code file with the extension .ASM
- the file containing the information data for the Debugger with extension .DBIand DBX
- the machine code file with extension .BIN

The .STC file contains the description of the program in the FSCODE language. This is a subset of the 'C' language that allows you to examine the program with a listing, without graphical description. It is used in the Debugger to debug the program step-by-step in which each step is a FSCODE language instruction.

The FSCODE file is used by the Compiler as source to generate the corresponding Assembler program. Also the assembler listing is used in the Debugger to follow the emulation of the program.

The .DBI and .DBX files contain reserved data used by the Debugger to link the high-level description with the low-level characteristics.

The .BIN code contains the data to be loaded in the device memory in binary format. This file is used as source by the Programmer tool.

Compiler Options

The Compiler Options allows you to customize the compilation phase. You can choose the compilation mode with the overflow control, if you need it, or optimize the code excluding the instructions for the interrupts control.

If you are using the functions IsOverflow() and IsUnderflow() inside your program, you must use the compilation mode with the Overflow Control. Using this compilation mode some additional code is added to some of the high-level instruction macros, in order to set properly the carry (C) or the sign (S) flag of the microcontroller, so the functions IsOverflow() and IsUnderflow() can check them correctly. The consequence is a less optimize code, due to the additional instructions added to some instruction with different types of operands. No instructions are added when using operation between BYTE type only operands.

To choose the compilation with overflow/undeflow control check the command COMPILE>OVER/UNDEFLOW CHECK from the menu TOOLS. To disable this compilation modality uncheck the same command.

Note: *If you are not using the IsOverfow() and IsUnderflow() function you are recommended to compile in normal mode in order to get optimized code.*

The other way to set compilation options is to use the Folder block in order to optimize the generated code, excluding the instructions for the interrupt control MEGI and MDGI. You can avoid the use of these Assembler instructions, added inside each high-level instruction macro, in order to decrease the program length and increase the execution speed when the interrupts are disabled. See Chapter 12 to get more information about the Folder block options.

Compiler Error Messages

In the following you can find the list of error messages and warnings that can occur during the program compilation.

Compilation errors

«allowed only constant arguments»

The argument required in the IRQEnableMask () function must be constant. Check for typing errors.

«allowed only constant expression»

Only constant or constant expressions are allowed in the context.

«xxx already defined»

The name xxx was already defined in the program. Change the name.

«break instruction is misplaced»

The break instruction has been used in a not allowed context. See the instruction grammar.

«cannot allocate an array of size 0»

An array of size 0 was declared. The constant expression used to allocate or declare an array must be an integral type greater than zero.

«configuration register expected»

The specified instruction needs a configuration register as operand.

«constant expression out of range xxx»

The constant expression value is out of range [0, 255] for BYTE expression value or [-128, 127] for S_BYTE expression value or [0, 65535] for WORD type or [–32768, 32767] for then S_WORD type.

«constant out of range»

The specified constant is out of range [0 , 255] for the BYTE type or [-128 , 127] for the S_BYTE type.

«xxx yyy[zzz] contains too many elements for dynamic indexing»

The device does not support dynamic indexing with arrays larger than 256 elements.

«continue instruction is misplaced»

The continue instruction has been used in a not allowed context. See the instruction grammar.

«“DeviceSet” requires only constant argument»

The argument required to the DeviseSet() function must be constant. Check for syntax errors.

«“DeviceSet” syntax error»

A generic error has been found in the specified DeviceSet() function. Check for typing errors.

«“DeviceSet” too few argument»

Few arguments than the required ones have been inserted into the function DeviceSet(). Check for syntax errors.

«“DeviceSet” wrong fifth parameter; ONCE or CONTINUOUS expected»

The fifth parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only the keywords ONCE and CONTINUOUS can be used.

«“DeviceSet” wrong fourth parameter; integer in the range [1..8] expected»

The fourth parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only integer number fro 0 to 8 can be used.

«“DeviceSet” wrong number of parameters»

The number of parameters specified in the function DeviceSet() is not correct. Check for missing or exceeding parameters.

«“DeviceSet” wrong second parameter; expected START»

The second parameter in the DeviseSet() function used in the instruction to set AllTimer is wrong. Only the keywords START can be used.

«“DeviceSet” wrong second parameter; RESTART or DISABLE expected»

The second parameter in the DeviseSet() function used in the instruction to set the Watchdog is wrong. Only the keywords RESTART and DISABLE can be used.

«“DeviceSet” wrong second parameter; SET or RESET expected»

The second parameter in the DeviseSet() function used in the instruction to set the Timer is wrong. Only the keywords SET and RESET can be used.

«“DeviceSet” wrong second parameter; START or STOP expected»

The second parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only the keywords START and STOP can be used.

«“DeviceSet” wrong seventh parameter; DIVIDED or FULL expected»

The seventh parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only the keywords DIVIDED and FULL can be used.

«“DeviceSet” wrong sixth parameter; SINGLE or SEQUENCE expected»

The sixth parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only the keywords SINGLE and SEQUENCE can be used.

«“DeviceSet” wrong third parameter; SET or RESET expected»

The third parameter in the DeviseSet() function used in the instruction to set the A/D converter is wrong. Only the keywords SET and RESET can be used.

«“DeviceStatus” requires only constant argument»

The argument required in the DeviseStatus function must be constant. Check for syntax errors.

«“DeviceStatus” requires two parameters»

The DeviceStatus function has been specified with a number of parameters different to the reqired ones.

«expression value xxx is out of range»

The expression value is out of range [0, 255] for BYTE expression value or [-128, 127] for S_BYTE expression value or [0, 65535] for WORD type or [–32768, 32767] for then S_WORD type.

«first argument must be a constant expression. »

The first argument is not a constant expression as required. Check for syntax errors

«first operand must be ‘WORD’ variable»

The first operand in DIV or MULT instruction must be a WORD type.

«xxx fuzzy variable already declared»

The fuzzy variable name xxx has been already defined. Change the fuzzy variable name.

«Fuzzy Rule syntax not supported»

The device cannot process the specified Fuzzy Rule. This occurs only with some particular rules having eight antecedent terms.

«incorrect use of vector xxx»

The xxx vector has not been used correctly. Check if the square brackets are missing or for other syntactic ot typing errors.

«Index too big. Allowed 0-xxx»

An index out of the allowed range as been specified in the instruction. Check for typing errors.

«input Fuzzy variable xxx not initialized in block yyy»

The input Fuzzy variable has not initialization variable specified as source in the Fuzzy block yyy. Open the block and initialize the Fuzzy variable by selecting a Global or Predefined variable.

«internal error»

Internal error occurred in FS4 Compiler. Please contact STMicroelectronics Fuzzy Logic B.

«xxx interrupt identifier already used»

The name xxx, specified as interrupt identifier within the interrupt-related function, has been specified twice in the function call.

«invalid character xxx»

The character xxx is a not valid character inside the source code.

«invalid crisp membership function»

The value specified as crisp membership function is out of the universe of discourse defined for the variable it belongs to. This error normally does not occur: it is an Internal Error, please contact STMicroelectronics Fuzzy logic B.U.

«invalid device»

The specified device cannot be used with the DeviceStatus function.

«invalid function call “xxx”»

The specified function has been called in wrong way. Check for syntax errors.

«invalid number of arguments; required x arguments. »

The specified procedure requires x arguments. Check for syntax errors.

«xxx invalid parameter»

The parameter xxx specified in the function is not valid in the context. Check for typing errors or for the instruction syntax.

«xxx invalid second parameter»

The second parameter in the DeviceStatus function is not valid.

«invalid setting for A/D converter. Check the peripheral configuration»

The setting specified with the DeviceSet() function to set the A/D Converter is not valid for the current configuration of the peripheral. This may occur if you changed the device pins configuration or the A/D configuration without modifying the already inserted peripherals blocks related to the A/D converter.

«xxx is a constant»

xxx is a constant, but the specified function requires a variable. Check for syntax errors.

«xxx is a read only variable»

The variable xxx is read-only and cannot be specified before the assignation operator ‘=’

«xxx is a reserved word»

The name xxx used for label or variable name cannot be used because it is reserved.

«xxx is a write-only variable»

The variable xxx is write-only and cannot be specified after the assignation operator '='

«IsBitReset function requires two parameters»

Invalid number of parameters; IsBitReset function requires two parameters.

«IsBitSet function requires two parameters»

Invalid number of parameters; IsBitSet function requires two parameters.

«xxx is not a vector variable»

The name "xxx" has been used as vector variable. Check for syntax errors.

«IsOverFlow function requires no parameters»

Parameters have been specified with the function IsOverFlow but this function does not require any parameter.

«xxx is read only»

The variable or table xxx is read-only and cannot be specified as left value in assignation instructions.

«IsUnderFlow function requires no parameters»

Parameters have been specified with the function IsUnderFlow but this function does not require any parameter.

«label xxx already defined»

The label name xxx has been already used in another part of the program. Change the label name.

«looped link after the block xxx»

The output link of the block xxx is connected to itself. This connection cannot be translated with any instruction. Connect the link to another link or block.

«membership function xxx already defined»

The membership function name xxx was alreay used for the same Fuzzy Variable. Change the membership function name.

«memory overflow. Var xxx not added»

The xxx variable has not been added in the apposite tool because there is not enaugh RAM locations in the device to allocate the variable.

«mismatched operand type»

The instruction uses a variable whose type is not allowed in the context. See the instruction grammar.

«missing argument xxx»

The name xxx, specified as interrupt identifier within the interrupt-related function, must be present in the IrqPriority() function call.

«missing conditional expression in block xxx»

The Conditional Block xxx has been inserted in the block diagram but the conditional expression has not been specified yet. Open the block and write the conditional expression.

«missing source and destination in block xxx»

The destination global or predefined variable has not been specified yet in the Send or Receive Block xxx. Open the block and select the source variable.

«number too big»

The given number is too large to be held in the specified type. Choose a larger variable's type to hold the given value.

«only constant array are supported»

It is not possible to modify table's values. Tables cannot be specified as left value in assignment instructions.

«pending branch in block xxx»

The output link of the block xxx is not connected to any other part of the program. Connect the link to the next block in the block diagram.

«pending 'No' branch in block xxx»

The output 'No' link of the Conditional Block xxx is not connected to any other part of the program. Connect the link to the next block in the block diagram.

«pending 'Yes' branch in block xxx»

The output 'Yes' link of the Conditional Block xxx is not connected to any other part of the program. Connect the link to the next block in the block diagram.

«predefined variable expected»

The specified instruction needs a predefined variable as operand.

«procedure xxx already defined»

The procedure name xxx has been already used in another procedure. Change one of the procedure names.

«procedure "xxx" called with too many arguments»

Invalid number of arguments; "xxx" procedure requires a different number of arguments.

«procedure "xxx" cannot be called with the call instruction»

You are trying to call a predefined procedure explicitely. Only user defined procedures can be called with the call instruction.

«second argument must be a lvalue»

The second argument is not a lvalue (left value in assignment) as required. Check for syntax errors.

«second argument must be a variable»

The second argument is a constant expression, but a variable is required. Check for syntax errors.

«syntax error»

Generic syntax error in writing instruction. Check for typing errors.

«syntax error reading "xxx"»

FSCcode contains an illegal character. This is an internal error : please contact STMicroelectronics Fuzzy Logic B.U.

«the operand must be a Global Variable»

A predefined variable has been specified as operand where only Global Variables can be accepted.

«the operand must be a variable»

An incorrect item has been found where a variable should be placed. Check the instruction grammar.

«this function can not be used with current compiler option»

The function IsOverflow or IsUnderflow have been used without setting the overflow control compiling option.

«too many operands»

More operands than expected have been specified in the instruction. With arithmetic operations, up to two operands with one operator can be used.

«type xxx unknown»

The specified type has not been recognized by the Compiler: the only types allowed are: BYTE, S_BYTE, WORD, S_WORD. Check for typing errors.

«type = xxx unsupported instruction»

The assignment instruction is not supported for the specified variable type. Check for syntax or typing errors.

«xxx undeclared fuzzy variable»

The xxx fuzzy variable has been used but it has not been declared yet.

«xxx undefined»

The operand xxx is undefined. Check for typing errors.

«xxx undefined device»

The xxx device name specified in the function call instruction does not exist. Check the device name for typing errors.

«xxx undefined for yyy»

The item xxx is misplaced for the item yyy. Check for syntax errors.

«xxx undefined fuzzy variable»

The specified fuzzy variable xxx was not defined. Define the fuzzy variable with the Fuzzy System editor before to use it.

«undefined interrupt»

The interrupt name specified in the call instruction does not exist. Check for typing errors.

«undefined label»

The label specified in the jump instruction has not been defined yet.

«xxx undefined membership function»

The specified membership function xxx was not defined. Define the membership function with the Membership Functions editor before to use it.

«undefined procedure»

The procedure specified in the call instruction does not exist. Check the procedure name for typing errors.

«xxx unknown»

The item name “xxx” is unknown; define the item before using it.

«unknown device»

The device name specified in the DeviceSet() function is unknow. Check for typing errors.

«xxx unsupported»

The procedure xxx is not supported.

«unsupported expression»

The Compiler cannot process the specified expression because it has too many operands and operators or a not supported syntax.

«unsupported instruction»

The specified instruction is not supported inside ASM blocks.

«unsupported Instruction on a Word variable»

L’istruzione specificata non puo’ essere utilizzata con variabili Word.

«variable expected»

The specified instruction needs a variable as operand.

«word type constant out of range»

The specified constant is out of range [0 , 65535] for the WORD type or [-32768 , 32767] for the S_WORD type.

«wrong number of parameters»

The number of parameters specified in the function is not correct. Check for missing or exceeding parameters.

Compilation Warnings

«bit operation with signed variables»

Bitwise operation with signed variables could be inconsistent.

«bit shift operation with signed variables»

Bitwise shift operation with signed variables could be inconsistent.

«loop in block xxx»

The output link of the block xxx is connected to its input link. This may cause an infinite loop.

«“DeviceSet” too many parameters; ignored the last xxx»

More parameters than the requested ones have been specified for the configuration of the specified peripheral; those surpluses will be ignored.

«dynamic indexing is allowed for (BYTE, S_BYTE) tables with up to 256 entries»

For the target device indirect access is allowed only for (BYTE, S_BYTE) tables with up to 256 entries.

«dynamic indexing is allowed for (WORD, S_WORD) tables with up to 128 entries»

For the target device indirect access is allowed only for (WORD, S_WORD) tables with up to 128 entries.

« “for” body looped due to condition statically true»

The condition of the FOR loop is always true. In this case loop’s instructions will always be performed.

« “for” statement ignored due to condition statically false»

The condition of the FOR loop is always false. In this case loop’s instructions will never be performed.

« “if” condition statically false, direct “else” execution»

The condition of the IF statement is always false. In this case the instructions of the block ELSE will automatically be performed.

« “if” condition statically true, direct “then” execution»

The condition of the IF statement is always true. In this case the instructions of the block THEN will automatically be performed.

« “if” ignored due to condition statically false»

The condition of the IF statement is always false. In this case the instructions of the block IF will never be performed.

«xxx may be reserved»

The name xxx is reserved for the target device.

«missing procedures name»

The procedure’s name has not been specified yet in the Call Block. Open the block and select the procedure to be called

«missing source and destination in block xxx»

The destination global or predefined variable has not been specified yet in the Send or Receive Block xxx. Open the block and select the source variable

«no interrupt selected in Interrupt Reset Block»

An Interrupt Reset Block has been inserted without specifying any interrupt source to be reset. Specify the interrupt pending you want to cancel or delete the block.

«not linked block xxx»

The block xxx has been inserted but not connected in the block diagram.

«unused Fuzzy variable xxx in block yyy»

The Fuzzy variable xxx in the Fuzzy Block yyy has been defined but it is never used in the Fuzzy Rules.

«“while” body looped due to condition statically true»

The condition of the WHILE loop is always true. In this case loop’s instructions will always be performed.

«“while” statement ignored due to condition statically false»

The condition of the WHILE loop is always false. In this case loop’s instructions will never be performed.

14 - DEBUGGER

DEBUGGER

The Debugger is the tool that allows to test the developed program by means of the chip's simulation. The Debugger graphical environment allows to choose and visualize 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.
- Tracing of the source program with step-by-step debugging, executing a program line for each step.
- Visualization and step-by-step tracing of the generated Assembler.
- Watch of the internal values and signals, with the possibility of evaluating user's expressions.
- Definition of signals applied to the pins by means of the Stimulus files.
- Dump of the Data and Program Memory in decimal, hexadecimal and binary format.
- Trace sequence of the program block in execution.
- Status report of the Debugger.
- Stop of simulation by means of breakpoints on events.

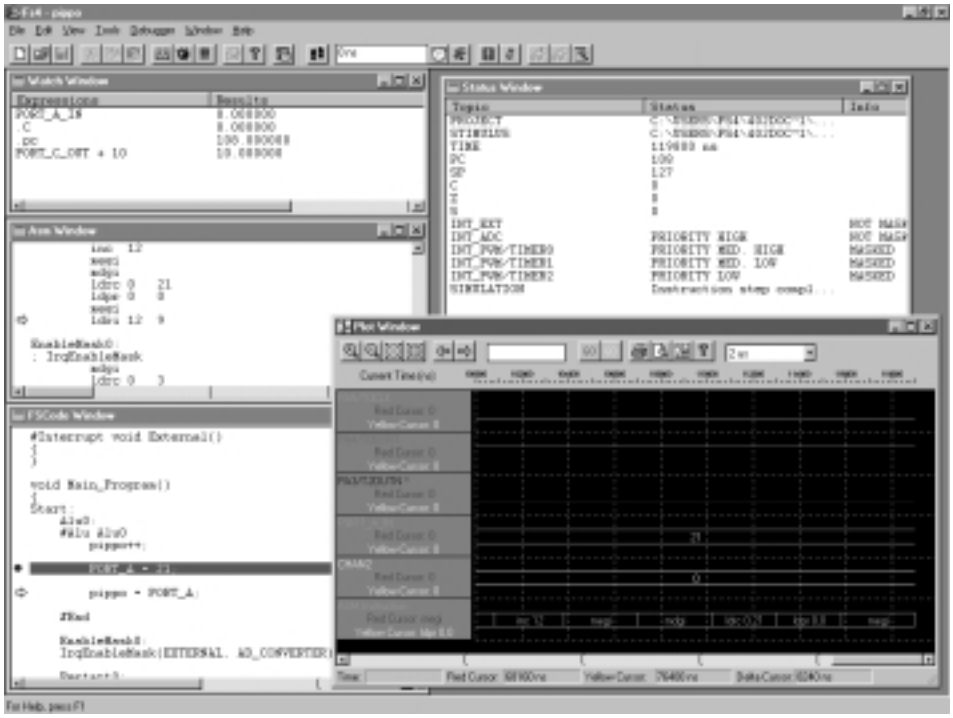
In this chapter you will learn to:

- Run the Debugger in different modes.
- Set the Watches and the items to plot graphically.
- Write the Stimulus file to set the signals applied to the external pins.
- Use the Plot Window to examine the result of the simulation.
- Get information from the various views available.

Debugger Window

This section provides an overview of the major elements of the Debugger window such as menus, toolbar and status bar, available when the Debugger session is open.

Fig. 14. - Debugger environment



Debugger menus

The following menu items are available when the Debugger is active:

File Contains commands to create, open, close, save and print window contents and to visualize the project's information.

Edit Provides standard editing commands.

View Contains commands to hide/show toolbars, status bar, Project and Output windows.

Tools Contains commands to close Debugger, Compiler or Programmer tools.

Debugger Contains command related to the Debugger's functions.

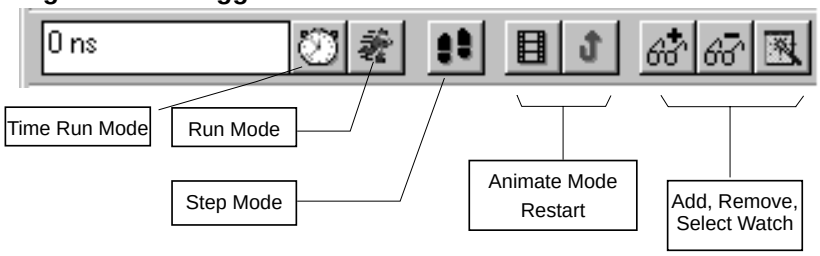
Window Contains commands related to window management.

Help Contains help commands.

Debugger window toolbar

The most frequently used commands can be executed quickly by clicking over the corresponding buttons available on the toolbar.

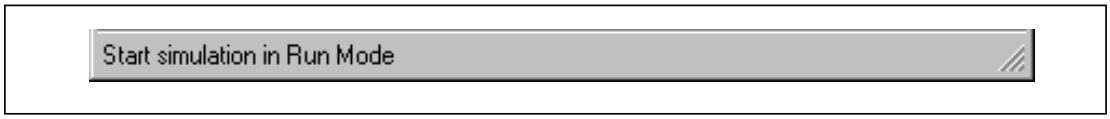
Fig. 14.2 - Debugger toolbar



Blocks Editor window Status Bar

The Status Bar displayed at the bottom of the Blocks Editor provides a brief description of the toolbar command currently pointed by the mouse cursor.

Fig. 14.3 Debugger status bar



Opening and Closing the Debugger

After opening the Debugger session for the first time, the status of the simulation and of the workspace is saved. So, each time you open again the debugger session of the single project, you will find the same situation that you left when the debugger session was closed. The status is joined with the project: opening another project you can find the status of that project.

To Open the Debugger session select the item DEBUGGER > OPEN from the TOOL menu or click the apposite toolbar button in the main window. If the session is opened for the first time or the last time no Debugger windows were open, the FSCODE window is opened.

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.*

You can close the Debugger session in two ways that are slightly different:

- Select the command CLOSE DEBUGGER WINDOWS from the DEBUGGER menu if you simply want to close all the debugger windows.
- Select the command DEBUGGER > CLOSE from the TOOLS menu if you want to close all the windows and discharge the computer memory from the debugger data.

To reset the simulation and restart from the time 0, clearing all the data, select the command RESTART from the DEBUGGER menu or click the apposite toolbar button in the debugger toolbar.

Debugger Working Modes

The Debugger environment supplies you with several views to easily manage the simulation data. You can find a detailed description of these views in the next paragraphs. There are four modes available to launch the simulation : Step, Run, Time Run, Animate.

Step Mode

This mode allows you to simulate the program step-by-step in order to debug the developed program instructions. It allows to execute a single instruction and view the results.

The step depends on the current window in foreground. If the ASM window (see later in this chapter) is in foreground the single step corresponds to the execution of the current Assembler instruction. If the FSCODE window (see later in this chapter) is in foreground the step is the single FSCODE instruction that corresponds to several Assembler lines. If none of the above mentioned windows is in foreground, the step is determined by the last of two that was in foreground.

To execute one step of simulation select the command STEP in the DEBUGGER menu or, better, click the apposite Debugger toolbar menu.

Run Mode

This mode executes the simulation until a stop command is issued by the user or a breakpoint or an exception (see later) is encountered.

To start the simulation in Run mode select the command RUN SIMULATION from the DEBUGGER menu or click the apposite toolbar button.

To stop the simulation in Run mode click the “Stop Simulation” button in the Stop dialog-box open in the upper-right side of the FUZZYSTUDIO™4.1 main window after the RUN command.

Note: *During the execution of the simulation, data in the Debugger windows are not updated until the execution is stopped.*

Time Run Mode

This mode is similar to the Run mode with the addition that the program can be stopped after the execution of a period of simulation time specified by the user.

To run simulation in Time Run mode:

- Specify the time and the time unit (ns, us, ms, s) in the apposite text box in the Debugger toolbar
- Click the Time Run toolbar button.
- To stop the simulation before the end of the specified period, click the “Stop Simulation” button in the Stop dialog-box opened in the upper-right side of the FUZZYSTUDIO™ 4.1 main window after the RUN command.

Note: *The execution in Time Run mode ends exactly after the completion of the last Assembler instruction. This causes the simulation to be executed for a little bit longer than the specified one. The exact execution time in nanoseconds is showed in the text box of the Debugger toolbar after the conclusion of the simulation interval.*

Animate Mode

In this mode the simulation is executed step-by-step (see Step mode previously described) continuously, until the simulation is stopped. After each step data are updated, allowing you to inspect in animation the evolution of the simulation. The delay between steps is configurable by the user, allowing to slower or increase the animation speed.

To run a simulation in Animate mode:

- Select the ANIMATE command from the DEBUGGER menu or click the apposite toolbar button.
- To stop the simulation, click the “Stop Simulation” button in the Stop dialog-box opened in the upper-right side of the FUZZYSTUDIO™ 4.1 main window after the ANIMATE command.

To change the delay between steps:

- Select the command OPTIONS > ANIMATION SPEED from the DEBUGGER menu. The Options dialog-box opens.
- Scroll the bar to slower or increase the animation speed.

Note: *The simulation is stopped when a breakpoint or exception is encountered.*

FSCODE Window

The FSCODE window shows the listing of the program in FSCODE language, allowing you to follow the tracing of the program instructions during the simulation. An arrow indicates the current line to be executed.

Fig. 14.4 - FSCODE window

To open the FSCODE window select the command FSCODE WINDOW from the DEBUGGER menu.

In the FSCODE window it is possible to set breakpoints on the program lines and select items to watch. To set a breakpoint in a program line:

- Right-click the program line you want to stop: the pop-up menu appears.
- Select the BREAKPOINT command from the pop-up menu.
- A bullet appears at the beginning of the program line.

To cancel the breakpoint select again the command from the pop-up menu or use the apposite dialog-box (see the “Breakpoint and Exceptions paragraph”).

To select an item to watch:

- Right-click the item to watch: the pop-up menu appears.
- Select the command ADD TO WATCH from the pop-up menu.
- The selected item appears in the list of watches in the Watch Editor.

Note: *the watches selection do not distinguish not allowed items, so you may insert terms with undefined value.*

ASM Window

The ASM window shows the listing of the program in Assembler language, allowing you to follow the tracing of the program Assembler instructions during the simulation. An arrow indicates the current line to be executed.

Fig. 14.5 ASM Window

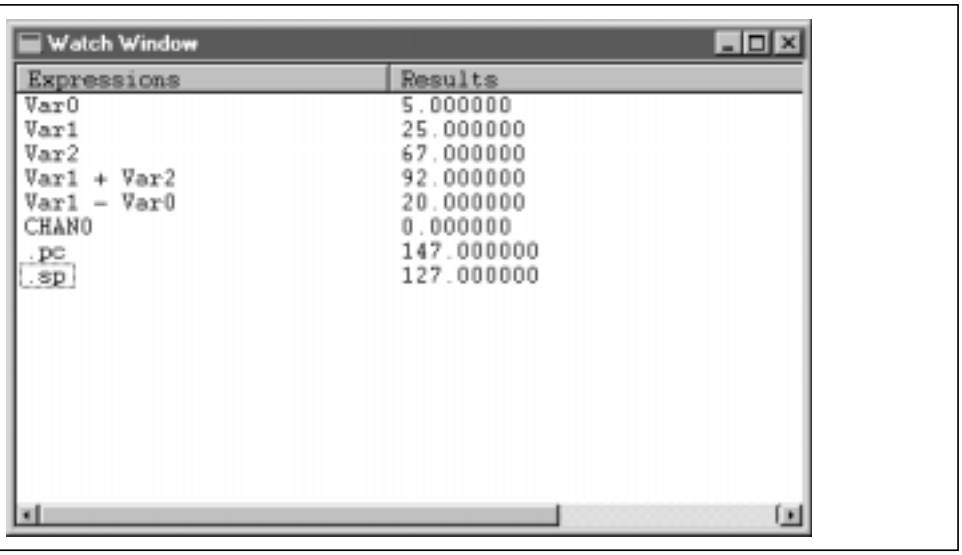
To open the Assembler window select the command ASM WINDOW from the DEBUGGER menu.
In the ASM window it is possible to set breakpoints on the program lines in the same way as previously described for the FSCODE window.

Watch Editor

The Watch Editor is the tool to inspect the variables and signals current value (watches). You can insert or remove the watches using the apposite commands; in addition the Watch Editor supplies an expression evaluator in order to get values from an arithmetic expression of several watches.

To open the Watch Editor select the command WATCH EDITOR from the DEBUGGER menu.

Fig. 14.6 - Watch Editor window



The Watch Editor window consists in two parts: in the left part you can insert and edit the items or the expressions to watch; in the right part the corresponding values are showed.

To add a new item or expression:

- Right-click anywhere on the Watch Editor client area or click the apposite Debugger toolbar button.
- A text-box becomes available in the expression list.
- Write the item name or expression in the text-box.
-

Note: *Inserting a wrong name or a non correct expression, the expression evaluator returns respectively the error message "Undefined Name" and "Invalid Expression". Another message that can occur is "Division by zero".*

To make the watch insertion easier, the Watch Editor supplies the Watch Select tool that allows to select the available items from the list organized as a tree-view. Act as follows:

- Click the Watch Select button from the Debugger toolbar.
- Navigate along the tree-view to find the item to watch.
- Select the item or an items' category.
- Click the Add button or double-click the item: the item is inserted in the expression list box in editing.

- Search for another item to be inserted or close the Watch Select tool clicking the Close button.

There is another way to insert automatically the items to watch by using the FSCODE window:

- Open the FSCODE window or put it in foreground.
- Right-click the item to watch: the pop-up menu appears.
- Select the command ADD TO WATCH from the pop-up menu.
- The selected item appears in the list of watches in the Watch Editor.

Note: *the watches selection do not distinguish not allowed items, so you may insert terms with undefined value.*

Expressions syntax

The text-boxes to edit expressions allow to freely write the expressions. So you have to take care in what you write, otherwise the expression cannot be evaluated. You must use only user-defined or predefined items and only allowed operands.

The allowed operators are the following (the function is the same as C language):

Arithmetic operators:

+ - * / - (unary)

Relational operators (return 0 or 1):

> < >= <= !=

Logic operators (return 0 or 1):

! && ||

Bitwise logic operators:

~ >> << & | ^

In addition to the user-defined and predefined variables, some special items can be used in the Watch Editor. They represent some registers of the device and must be preceded by a dot:

Registers:

.ram[x]	x = RAM address
.eprom[x]	x = EPROM address
.chan[x]	x = A/D Channel number
.reg_conf[x]	x = Configuration Register address
.reg_input[x]	x = Input Register address
.reg_output[x]	x = Output Register address
.reg_fuzzy[x]	x = Fuzzy Input Register number

Core items:

.pc	Program Counter
.time	Simulation time in nanoseconds
.sp	Stack Pointer
.C	Carry Flag
.Z	Zero Flag

.S	Sign Flag
Fuzzy buffers:	
.stack0	First buffer to perform MAX or MIN operation.
.stack1	Second buffer to perform MAX or MIN operation.
.num	Numerator of defuzzyfication formula.
.den	Denominator of defuzzyfication formula.
.teta	Activation value of the Antecedent part of the Rule.

Note: *When you want to watch a fuzzy variable, you must specify also the fuzzy block name to which the variable belongs. The format to be used is: fuzzy_block_name.variable_name*.

Breakpoints

The Breakpoints allow to stop the simulation, in order to inspect the results, when an event is triggered or when a program line is reached.

You can easily add or remove line breakpoints operating with the FSCODE and ASM windows. In addition the Breakpoints dialog-box allows to edit event breakpoints and show all the active ones.

To add a line breakpoint:

- Open the FSCODE or the ASM window or put it in foreground.
- Right-click the program line you want to stop: the pop-up menu appears.
- Select the BREAKPOINT command from the pop-up menu
- A bullet appears at the beginning of the program line

To cancel the breakpoint select again the command from the pop-up menu or use the apposite dialog-box (see below).

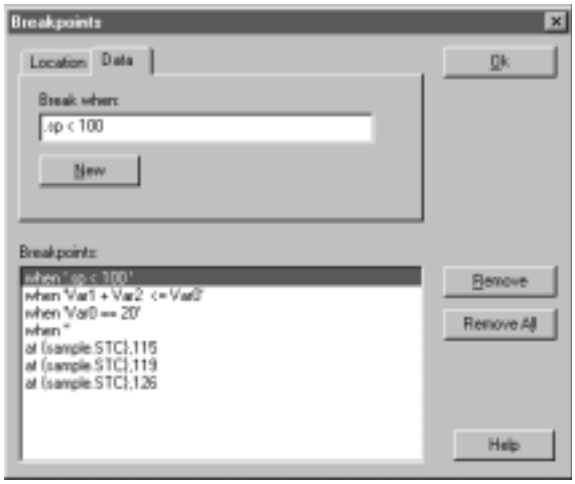
To set event breakpoints you have to open the Breakpoint dialog-box by selecting the command BREAKPOINTS... from the Debugger menu.

The dialog-box is composed by a text-box that allows to set the condition that triggers the breakpoint, and a list-box where all set breakpoints are listed (both line and event types).

To add a breakpoint on event, you have to specify the condition that determines the stop of simulation:

- Select the “Event” sheet in the top of dialog-box.
- Write the condition using the syntax described below. The condition is inserted when you write and it appears in the list-box.
- Click the New button if you want to insert another breakpoint.
- Click O.K. to close dialog-box and confirm your settings.

Fig. 14.7 - Breakpoints dialog-box



The condition is an expression having as operands the variables and the signals available and the syntax and operators described in the Watch Editor paragraph.

Examples of conditions:
PA0 == 1

Var0 <= 10

Var0+Var1 != Var2

.ram[20] == 30 && .ram[21] == 19

.pc > 1000 || .time > 200000

Warning: no control is performed in what you write as breakpoint condition: not valid expressions are automatically not considered.
You can also remove the active breakpoints by using the Breakpoints dialog-box:

- Select the breakpoint to be removed
- Click the Remove button
- If you want to remove all the active breakpoints click the “Remove All” button.

Exceptions

Exceptions are particular fault conditions that can be intercepted to stop the simulation. You can ignore those situation or set the stop of the simulation by using the Exceptions dialog-box:

1. Select the EXCEPTIONS… command from the DEBUGGER menu

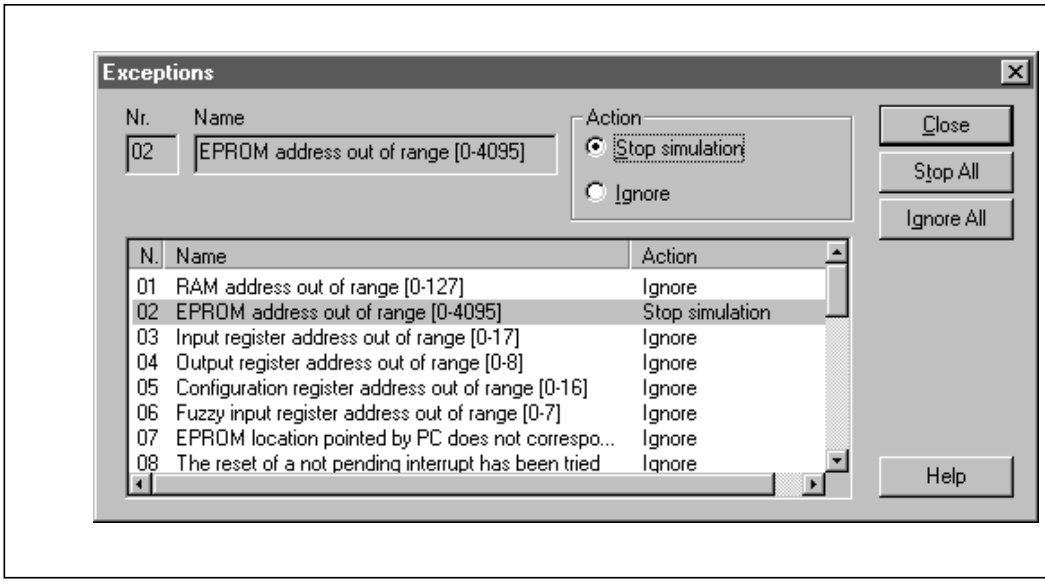
2. Select the Exception from the exception list-box

3. Check the “Stop simulation” or the “Ignore” radiobutton in the “Action” section.

4. Click the “Stop All” button if you want all the exceptions active or click th e”Ignore All” if you want to disable all them.

5. Click the Close button.

Fig. 14.8 - Exceptions dialog-box



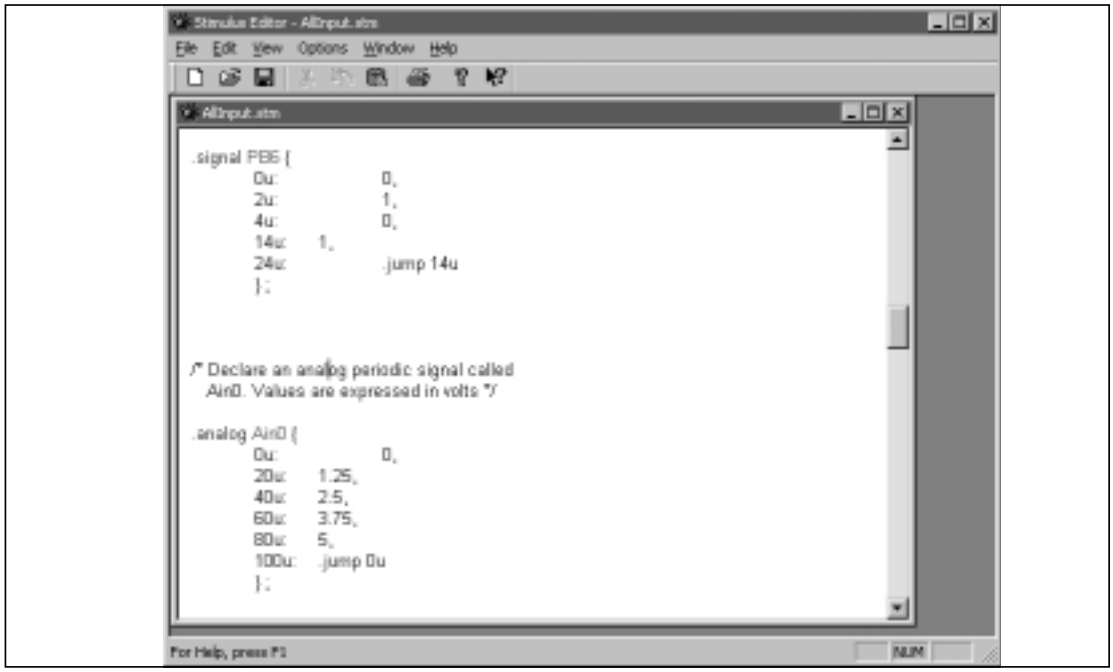
The Exceptions are characterized by a code and a message that depends on the target device. You can find the list of the Exceptions for each devices in the Appendix A.
When an enabled Exception occurs a dialog-box appears showing the exceptions occurred in the last cycle.

- Click the Stop button to stop the simulation and inspect the exception causes.
- Click the Ignore button to ignore the event and continue the simulation until the exception will occurred again.

Stimulus Editor

The Stimulus Editor is a tool that allows to write the Stimulus file used to specify the signals to be aplied to the device pins. The tool is external to the FUZZYSTUDIO environment but can be run from this. It is a text editor where you can write a program by using a simple representation language whose syntax is described lated in this paragraph.

Fig. 14.9 - Stimulus Editor



To run the Stimulus Editor select the STIMULUS EDITOR command from the DEBUGGER menu. The program supplies the standard editing command and the possibility to load previously defined stimulus files. The text editor recognizes the keyword and the device pins' name and highlights them by using a colored and underlined font when you write: blue for the keywords and red for the pins. This allows you to check typing errors when you write.

The SET STIMULUS FILE command in the DEBUGGER menu has to be used to specify the stimulus file to be used. The command determines the opening of the standard Open dialog-box that allows you to select the stimulus file edited with the Stimulus Editor. The Stimulus files have .stm extension.

The SET STIMULUS FILE command establishes the stimulus file currently used by the Debugger. You can modify this file with the Stimulus Editor adter executing the command: to make effective the modification you have to used the COMPILE STIMULUS FILE command from the DEBUGGER menu. This command is also automatically performed with the SET STIMULUS FILE command.

The COMPILER STIMULUS FILE command performs the compilationof the stimuls file. If the file does not exist or in case of syntax errors, a window opens showing the error messages.

Note: *The setting or compilation of the Stimuls File substitues the Stimulus file currently in use.*

Generic structure of a Stimulus File

The structure of the Stimulus file is composed by some sections of the following types:

- Digital Signals description
- Analog signals description
- Periodic signals description
- Buses declaration
- Buses description
- Thresholds declaration
- Random signals declaration
- Comments

You do not need to use all the sections and they can be used without any order or priority as many times as you want (excluding thresholds). The bus declaration must be specified before the bus description.

Digital signals description

This section is used to specify the digital signals in input to the pins of the device. The syntax is the following:

```
.signal      pin_name{
time:        value,
time:        value,
……..
time;         value
};
```

where:

- *pin_name* is the device pin with the same name as described in the data-sheet. You can find the list of these names in Appendix A.
- *time* is the time value when a signal variation occurs. The time value is expressed with a real number and the time unit appended (for example: 10n, 20.5u, 120m, 2.2s)
- *value* is the new value of the signal at the specified time. In the case of .signal costruct the only allowed values are 0 and 1.

The following examples generate a step function on the port pin PB4 and an impulse of 500ns on the pin PA2:

```
.signal PB4 {
   0u:       0,
   5u:       1
};
.signal PA2 {
   0n:       0,
   10u:      1,
   10,5u:    0
   };
```

Analog signals description

This section is used to specify the analog signals in input to the pins of the device. You can use this description not only for analog pins but also for digital pins: the emulator convert the analog value in the digital value 0 or 1 according to the thresholds set in the apposite declaration section (see later in this paragraph). The syntax is the following:

.analog pin_name {
time: value,
time: value,
……..
time: value
};

where:

- *pin_name* is the device pin with the same name as described in the data-sheet. You can find the list of these names in the Appendix A.
- *time* is the time value when a signal variation occurs. The time value is expressed with a real number and the time unit appended (for example: 10n, 20.5u, 120m, 2.2s)
- *value* is the new value of the analog signal at the specified time. In the case of **.analog** costruct the only allowed values are real number between 0 and 5 (Volts).

Example:

.analog Ain3 {
0u: 0.5,
5u: 1.2,
10u: 4.6,
20u: 0.9
};

Grouping signals in buses

It is possible to group signals in buses by means of the buses declaration and use them with the .bus construct. To group signals use the following declaration syntax:

.declare bus_name = pin_name, pin_name,….., pin_name ;

where:

- *bus_name* is a user defined name that identifies the group of signals. It is then specified in the **.bus** construct.
- *pin_name* is the device pin with the same name as described in the data-sheet. You can find the list of these names in Appendix A.

Examples:

.declare PORTC = PC0, PC1, PC2, PC3;
.declare INPUT_DATA = PA0, PA1, PA4, PA6, PA7

The description of the buses is performed by using the **.bus** construct whose syntax is the following:

```
.bus bus_name {
time:        value,
time:        value,
……..……..
time:        value
};
```

where:

- *bus_name* is the name of the bus as specified in the declaration.
- *time* is the time value when a bus value variation occurs. The time value is expressed with a real number and the time unit appended (for example: 10n, 20.5u, 120m, 2.2s)
- *value* is the new value of the bus at the specified time. In the case of **.bus** construct the allowed values depend on the length of the bus: the maximum value is (2^n)-1, being n the number of signals that compose the bus; the minimum value is 0.

Examples:

```
.bus PORTC {
   0u:       0,
   10u:      10,
   20u:      8,
   30u:      15,
   40u:      3
   } ;


.bus INPUT_DATA {
   0u:       0,
   10u:      10,
   20u:      20,
   30u:      30
   } ;
```

Periodic signals

There are two ways to define periodic signals: the **.clock** construct and the **.jump** statement.

The **.clock** construct allows to define periodic digital signals with specified duty-cycle. The syntax is the following:

.clock pin_name {

period,

initial_value,

duty_cycle

};

where:

- *pin_name* is the device pin with the same name as described in the data-sheet. You can find the list of these names in the Appendix A.
- *period* is the period of the signal. The value is expressed with a real number and the time unit appended (for example: 10n, 20.5u, 120m, 2.2s).
- *initial_value* is the value of the signal at the start time. The only allowed values are 0 and 1.
- *duty_cycle* is the ratio between the time when the signal value is 1 (Ton) and the entire period. The duty cycle can be expressed as the duration of Ton, expressed with a real number and the time unit appended (for example: 10n, 20.5u, 120m, 2.2s), or as percent (for example 30%, 23,4% etc.). This parameter can be omitted: in this case a signal with 50% of duty-cycle is generated.

Examples:

Declare a digital periodic signal as a clock, applied on PB5 pin. It has a period of 24u, 0 as init value, and a Ton of 12u :

```
.clock PB5 {
  24u ,
  0,
  12u
  } ;
```

Declare a digital periodic signal as a clock, applied on pin PA5. It has a period of 20u, 1 as init value, and a dut-cycle of 25%

```
.clock PA5 {
  20u ,
  1 ,
  25%
  } ;
```

Declare a digital periodic signal as a clock, applied on pin PA6. It has a period of 30u, 1 as init value, and a duty-cycle of 50%

```
.clock PA6 {
  20u ,
  1
  } ;
```

The other way to generate more complex periodic signals, both digital and analog, is to use the **.jump** statements. These statements can be used in the **.signal**, **.analog** and **.bus** constructs in the place of the last signal value specified. They allow to repeat the specified signal from the specified time.

Examples:

The following example generates a periodic signal having period of 24 microseconds and 50% duty-cycle

```
.signal PB2 {
   0u:        0,
   12u:       1,
   24u:       .jump 0u
   } ;
```

The following example generates a periodic signal after a start-up sequence. After the first cycle the signal generates a period of 10 microseconds and 50% duty-cycle.

```
.signal PB6 {
   0u:        0,
   2u:        1,
   4u:        0,
   14u:       1,
   24u:       .jump 4u
   } ;
```

The following example generates an analog periodic signal.

```
.analog Ain0 {
   0u:        0,
   20u:       1.25,
   40u:       2.5,
   60u:       3.75,
   80u:       5,
   100u:      .jump 0u
   } ;
```

Random signals

When a program is tested, many times random signals can be useful. You can create this kind of signals by using the **.random** statement with the **.signal**, **.analog** and **.bus** contructs. The syntax is the following:

.construct_name pin_name **.random** *time* **;**

where:

- *construct_name* is .signal, .analog or .bus
- *pin_name* is the device pin with the same name as described in the data-sheet. You can find the list of these names in the Appendix A.
- *time* is the period of the generation of a new random value.

Examples:

generation of a random digital signal whose value is updated every 100 microseconds:
.signal PB3 .random 100u ;

generation of a random analog signal whose value is updated every 15 milliseconds:
.analog Ain2 .random 15m ;

Thresholds declaration

The analog signal generation can be used also with digital pins. In this case the emulator must know the threshold values to convert the analog value in a digital level 0 or 1. To specify the threshold values use the **.vh** and **.vl** keywords with the following syntax:

.vh *value* ; (high threshold voltage value)

.vl *value* ; (low threshold voltage value)

the *value* parameter is the voltage value of the threshold expressed with a real number between 0 and 5.

Note: *if the thresholds declaration is omitted the default value are assumed: 2.0 for vh and 0.8 for vl.*

Comments

Comments can be inserted inside the Stimulus file in order to improve the readability of the stimulus program. Use the characters /* to start a comment sequence and the characters */ to end the sequence.

Stimulus Editor Error Messages

The following error messages and warnings can occur when compiling the stimulus file:

invalid stimulus file

The stimulus file contains errors and cannot be inserted in the simulation. Check for syntax or typing errors.

invalid analog value: it must be in the range [0.0-5.0]

A wrong value has been specified in the .analog statement as analog signal value. The allowed values are real numbers between the range [0.0 – 5.0] volts.

invalid digital value: it must be 0 or 1

A wrong value has been specified as digital signal value. The allowed values are 0 and 1 corresponding respectively to a low and high voltage value supplied to a digital pin.

.vh instruction: prototype must be ".vh voltage ;"

The threshold instruction .vh has not been specified correctly. Check for syntax or typing errors.

.vl instruction: prototype must be ".vl voltage ;"

The threshold instruction .vl has not been specified correctly. Check for syntax or typing errors.

.vh instruction: too few parameters

The threshold instruction .vh has not been specified correctly. Check if the threshold value is missing of for syntax or typing errors.

.vl instruction: too few parameters

The threshold instruction .vl has not been specified correctly. Check if the threshold value is missing of for syntax or typing errors.

.vh instruction: too much parameters

The threshold instruction .vh has not been specified correctly. Check if the threshold value has been specified correctly and if it has been specified just once.

.vl instruction : too much parameters

The threshold instruction .vl has not been specified correctly. Check if the threshold value has been specified correctly and if it has been specified just once.

"vh" parameter already defined

The high threshold parameter vh has been already defined with a .vh instruction previously specified.

"vh" parameter lower than "vl" parameter

The specified high threshold value is lower than the low one. Check the values specified in the .vh and .vl instruction or as default and correct them.

vl parameter already defined

The low threshold parameter vl has been already defined with a .vl instruction previously specified.

.declare instruction : prototype must be ".declare bus name = list of signal names ;"

The .declare instruction has not been specified correctly. Check for syntax or typing errors.

.declare instruction : missing bus name

The .declare statement has not been specified correctly: the symbolic name for the bus has not been specified correctly or it is missing. Add the bus name just after the .declare instruction.

.declare instruction : missing =

The .declare statement has not been specified correctly: the assignment character '=' has not been specified correctly or it is missing. Add the '=' symbol just after the bus name.

.declare instruction : missing , between signals names

The .declare statement has not been specified correctly: the signal names have not been separated correctly with the ',' character.

.signal instruction: invalid prototype

The .signal instruction has not been specified correctly. Check for syntax or typing errors.

.signal instruction: prototype must be ".signal signal name { list of signal values } ;"

The .signal statement using the list of signal values has not been specified correctly. Check for syntax or typing errors.

.signal instruction: prototype must be ".signal signal name .random time ;"

The .signal instruction using the .random statement has not been specified correctly. Check for syntax or typing errors.

.signal instruction : missing signal name

The .signal instruction has not been specified correctly: the symbolic name for the signal is missing. Add the signal name just after the .signal instruction.

.signal instruction : missing {

The .signal instruction has not been specified correctly: the open brace '{' is missing. Add the brace just after the signal name to enclose the signal values.

.signal instruction : missing }

The .signal instruction has not been specified correctly: the close brace '}' is missing. Add the brace at the end of the signal values to close the list.

.signal instruction : the list of signal values is empty

The .signal instruction has not been specified correctly: no value has been specified in the list. Include the signal values in the list.

.signal instruction : the list of signal values is not valid

The .signal instruction has not been specified correctly: the list of the signal values contains syntax or typing errors.

.analog instruction : invalid prototype

The .analog instruction has not been specified correctly. Check for syntax or typing errors.

.analog instruction : prototype must be ".analog analog name { list of analog values } ;"

The .analog statement using the list of analog values has not been specified correctly. Check for syntax or typing errors.

.analog instruction : prototype must be ".analog analog name .random time ;"

The .analog instruction using the .random statement has not been specified correctly. Check for syntax or typing errors.

.analog instruction : missing analog name

The .analog instruction has not been specified correctly: the symbolic name for the analog signal is missing. Add the signal name just after the .analog instruction.

.analog instruction: missing {

The .analog instruction has not been specified correctly: the open brace '{' is missing. Add the brace just after the analog signal name to enclose the analog signal values.

.analog instruction: missing }

The .analog instruction has not been specified correctly: the close brace '}' is missing. Add the brace at the end of the analog signal values to close the list.

.analog instruction: the list of analog values is empty

The .analog instruction has not been specified correctly: no value has been specified in the list. Include the analog signal values in the list.

.analog instruction: the list of analog values is not valid

The .analog instruction has not been specified correctly: the list of the signal values contains syntax or typing errors.

.bus instruction: invalid prototype

The .bus instruction has not been specified correctly. Check for syntax or typing errors.

.bus instruction: prototype must be ".bus bus name { list of bus values } ;"

The .bus statement using the list of bus values has not been specified correctly. Check for syntax or typing errors.

.bus instruction: prototype must be ".bus bus name .random time ;"

The .analog instruction using the .random statement has not been specified correctly. Check for syntax or typing errors.

.bus instruction : missing bus name

The .bus instruction has not been specified correctly: the symbolic name is missing. Add the bus name just after the .bus instruction.

.bus instruction : missing {

The .bus instruction has not been specified correctly: the open brace '{' is missing. Add the brace just after the bus name to enclose the bus values.

.bus instruction: missing }

The .bus instruction has not been specified correctly: the close brace '}' is missing. Add the brace at the end of the bus values to close the list.

.bus instruction: the list of bus values is empty

The .bus instruction has not been specified correctly: no value has been specified in the list. Include the bus values in the list.

.bus instruction: the list of bus values is not valid

The .bus instruction has not been specified correctly: the list of the bus values contains syntax or typing errors.

.bus instruction: bus name not found

The bus name specified in the .bus statement has not been declared before with the .declare statement. Declare the bus name or check for typing errors.

.clock instruction : invalid prototype

The .clock instruction has not been specified correctly. Check for syntax or typing errors.

.clock instruction : prototype must be ".clock clock name { period, initValue, dutycylcle or Ton} ;"

The .clock instruction has not been specified correctly. Check for syntax or typing errors.

.clock instruction : missing clock name

The .clock instruction has not been specified correctly: the symbolic name for the clock signal is missing. Add the clock signal name just after the .clock instruction.

.clock instruction : missing {

The .clock instruction has not been specified correctly: the open brace '{' is missing. Add the brace just after the clock signal name to enclose the clock parameters.

.clock instruction : missing }

The .clock instruction has not been specified correctly: the close brace '}' is missing. Add the brace at the end of the clock parameters list.

.clock instruction : the list of clock parameters is empty

The .clock instruction has not been specified correctly: no parameter has been specified. Include the clock signal parameters in the list.

.clock instruction : the list of clock parameters is not valid

The .clock instruction has not been specified correctly: the list of the bus values contains syntax or typing errors.

.clock instruction : the Ton is greater than the period

The specified Ton parameter value is not valid because greater than the period: remember that the Ton is a time part of the period so it must be lower than the period parameter value. Specify a correct value for the Ton or for the period or check for typing errors.

"name" is an invalid signal name

The signal name "name" is not a valid device pin name. Check the correct pin name in the list of the available ones in the Appendix A of this manual.

Signal "name" already defined

The signal name "name" has been previously defined. Choose another signal name.

Simulation Plot

The simulation results are represented in their time evolution, in a graphic way, in the Plot window where signals and buses are traced. In a few words, it works as an oscilloscope with the probes connected to the hardware of the selected device. The items to be plotted are selected by means of the Plot Select dialog-box, where all the available signals, variables and buses are listed.

The signals are functions with two values that are the logical states 0 and 1 and they are represented with a continuous broken line. The buses indicate the content of the register or a set of signals and they are represented with a continuous stripe, broken in the points where the values change; the values are written inside the stripe.

After the stop of simulation the signals can be carefully examined in their evolution. Many commands such as zoom, cursors, jumps and others are available to make it easy to inspect the results. UNDO and REDO commands are available for these kind of actions.

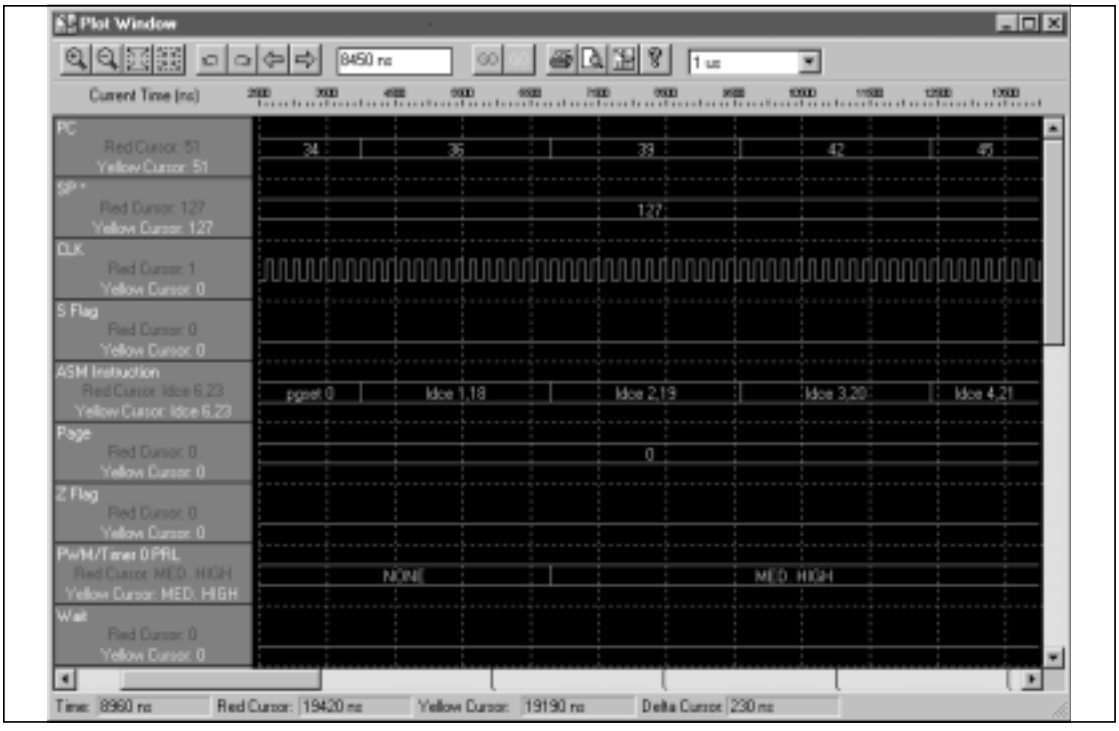
Plot window

The Plot window is organized in two main parts: in the left side you can find the list of the selected items to be plotted, in the right side the corresponding signals lines and buses stripes. Just below the item name, you can find the values of the item in the times pointed by the red and yellow cursors (see later in this chapter). If the cursors are hidden, the default value (i.e. the value of the item at the start of the simulation) is showed.

You can set the current plot item by clicking over its name in the left side of the window. An asterisk (*) just after the item name indicates the current plot item.

In the upper part of the window, just below the toolbar, you can find the indication of the currently displayed simulation time in nanoseconds.

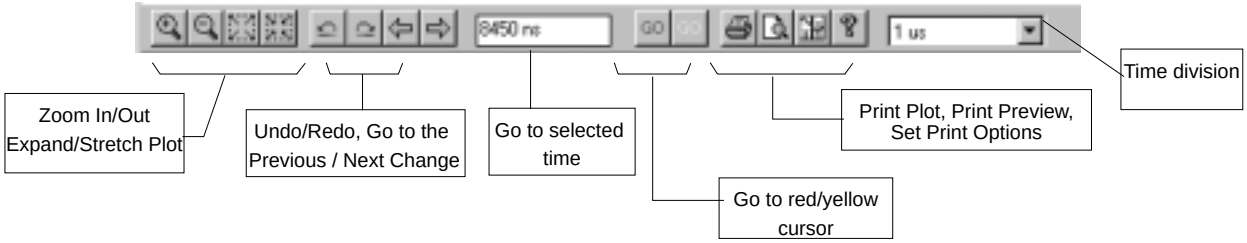
Fig.14.11 - Plot window



Plot window toolbar

The Plot window includes a toolbar to help you to perform the commands quickly. To execute a task by means of a button, juck click the related button on the toolbar.

Fig.14.12 - Plot window toolbar



Plot window status bar

The status bar displayed at the bottom of the Plot window contains information about the time points of the simulation currently pointed by the mouse pointer or by the cursors.

Fig. 14.13 - Plot window status bar

The first field indicates the time pointed by the mouse pointer; the second field is the position of the red cursor; the third field is the position of the yellow cursor; the last field is the time difference between the position of the two cursors (see later in this chapter to learn about the cursors).

Selecting plot items

You can select all the items to plot by means of the Plot Select tool. This consists on a check-list organized as tree-view. The items are organized in categories that are represented by the nodes of the tree, the leaves are the items to be selected.

The main categories are the following:

Pins: device external pins

Variables: Global and Predefined Variables

Peripherals: signals and variables related to the peripherals

Core: signals related to the Control Unit and interrupts

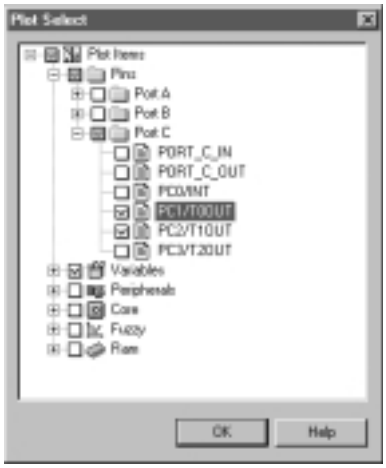
Fuzzy: buffers for the fuzzy computation

Ram: RAM memory addresses

These categories nodes are further divided in sub-categories nodes in order to make the items search easier. In addition they allow to select all the items belonging to the categories.

To open the Plot Select dialog-box chose the command PLOT SELECT... from the menu DEBUGGER.

Fig. 14.14 - Plot Select tree-view



To select one or more items:

Expand the nodes and subnodes to which the item belongs to. You can do this by clicking the plus sign next to the category name or double clicking on this last.

1. Check the check-box next the item name.

2. Repeat to select other items.

3. Check the node or sub-node check-box to select all the items belonging to the category.

4. Uncheck the items and categories that you do not want to plot anymore.

5. Click OK button to confirm the selection.

Note: *You can always select and unselect the items during the debugger session. If the simulation has been already started, the history of the added items is not retrieved.*

Zooming simulation

After stopping the simulation, the results are drawn in the Plot window with the currently set scale. The drawing scale is represented by the single step of the grid: the time division can assume values from 50 ns to 500 ms. You can modify the current representation scale by using the command available on the Plot window toolbar:

- The Zoom Out command increases the time division.
- The Zoom In command decreases the time division.
- The Expand command executes the maximum zoom of the window.
- The Stretch command allow to visualize the entire drawing.
- The Time Division drop-down list allows you to select the desired time division.

By using these commands you can modify the scale of the drawing to examine the result with different degree of detail.

In addition it is possible to zoom in a selected part of the chart by clicking and dragging the mouse to open a selection frame. The selected part will be displayed according with the most suitable time division and centered to the center of the frame.

Cursors

The Plot window supplies two cursors to take measures in the simulation results: the Yellow and the Red cursors. They allow to watch time where they are placed, the value of the items in that time and the time difference between the positions of two cursors.

The time measurements can be watched in the Plot window status bar; the item's values are showed just below the item name.

The cursors can be hidden or shown: right-clicking the drawing, the pop-up menu appears where you can select the commands that allow to insert the cursors where you clicked or hide them:

- INSERT RED CURSOR
- INSERT YELLOW CURSOR
- HIDE RED CURSOR
- HIDE YELLOW CURSOR

The cursors can be moved along the drawing by click and drag operations. Repeating the insert command , the cursor is moved in the new place where you clicked.

Go To

The Plot window supplies some commands to jump to one point of the simulation to another one, according to the type of command. These commands are available in the Plot window toolbar:

- GO TO RED CURSOR
- GO TO YELLOW CURSORS
- GO TO NEXT CHANGE
- GO TO PREVIOUS CHANGE

The last two commands allow to follow the changes of the currently selected signal or variable. To select the items click over the corresponding box.

In addition, the toolbar supplies a text box where you can specify the time where to jump. To do this you have to specify the time and the time unit (ns, us, ms, s). If the time specification is not correct the jump is not executed.

Customizing the Plot window

You can customize the plot window changing the colors and the height of the items' boxes. In addition it is possible to change the numerical representation of the buses values.

To change the color of the drawing:

- Right-click the mouse over the item to change the color.
- Select SET COLOR command from the appearing pop-up menu.
- Choose the color from the appearing dialog-box.

To change the height of the item's box:

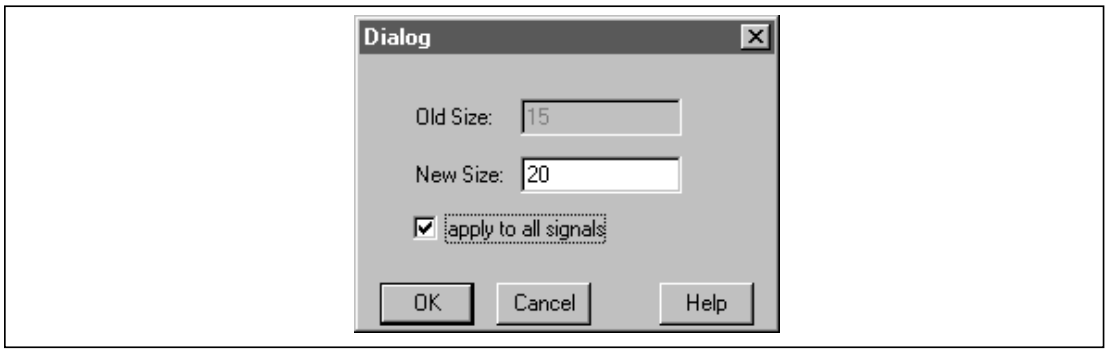
- Right-click the mouse over the item.
- Select SET HEIGHT command from the appearing pop-up menu.
- In the appearing dialog box you can read the currently set height.
- Change the value in the apposite text-box, specifying a value between 5 and 20.

- Check the “apply all signals” check-box if you want to change the height to all the items boxes.

Right-clicking on a bus type item the pop-up menu supplies the commands to change the numerical representation on the buses values. There are three types of representation available: decimal, hexadecimal and binary. By using these commands you can change:

- the type of the Red Cursor values.
- the type of the Yellow Cursor values.
- the type of the values in the drawing.

Fig. 14.15 - Change height dialog-box



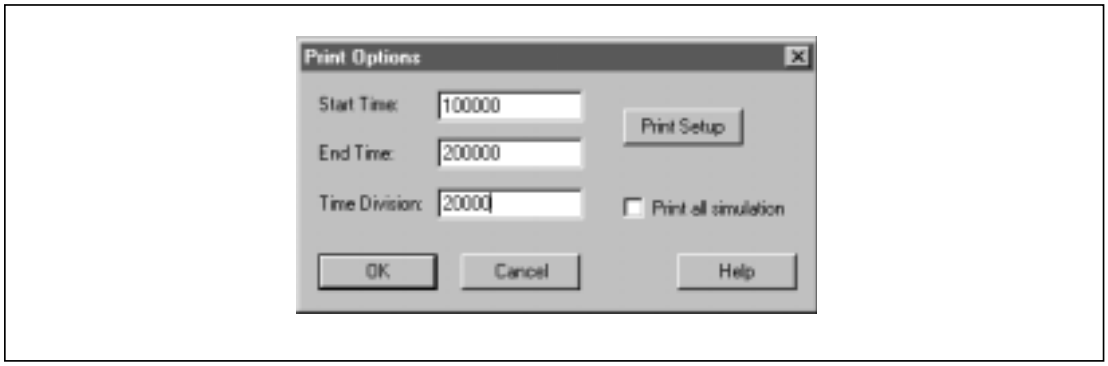
Plot Print Options

The Plot Print Options can be accessed by the apposite command in the Plot window toolbar. It allows to specify the part of the simulation drawing to print.

You can choose to print all the simulations by checking the apposite check-box: when un-checked you have to specify the following data:

- the Start Time of the part to print.
- the End Time.
- the Time Division to set the grid on print.

Fig. 14.16 - Plot Print Options dialog-box



The values must be expressed in nanoseconds.

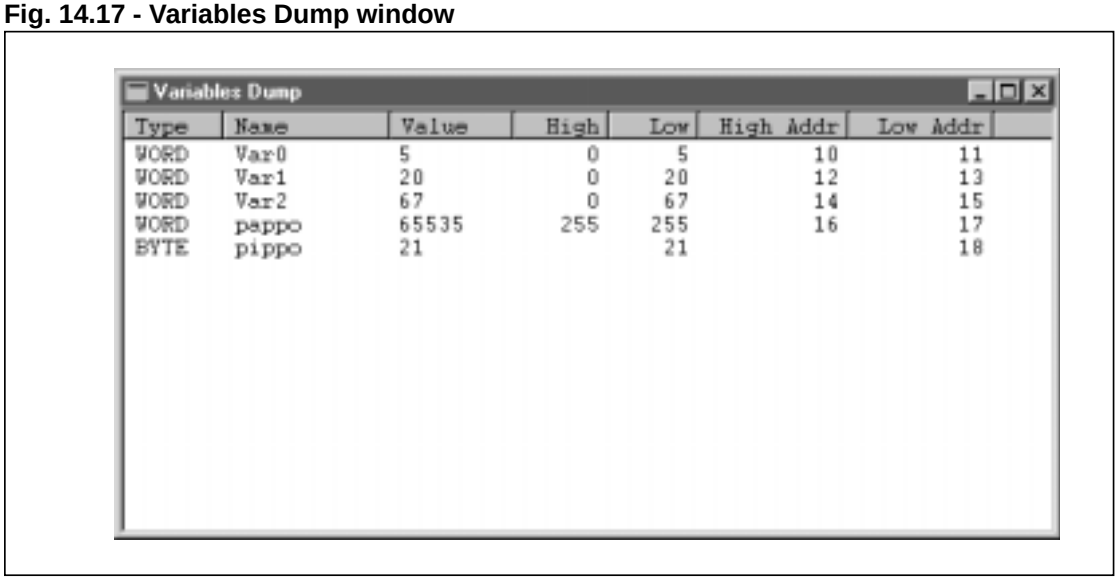
Note: *The simulation is printed in one page; for this reason the scale of representation is chosen automatically in order to fit the printing on the page.*

Variables Dump window

The Debugger supplies the possibility to inspect the Global Variables current value and modify it. You can do that by using the Variables Dump window.

To open the Variable Dump window select the VARIABLE DUMP command from the DEBUGGER menu.

Fig. 14.17 - Variables Dump window



The window shows the list of the variables and it is composed by seven fields:

Type: the variable's type.

Name: the variable's name.

Value: the variable's current logic value.

High: the value contained in the high RAM location that stores the variable (if 8-bit type the field is empty).

Low: the value contained in the low RAM location that stores the variable

High Addr: the address of the high RAM location that stores the variable (if 8-bit type the field is empty).

Low Addr: the address of the low RAM location that stores the variable

You can edit and modify the VALUE field by clicking on it to change the variables' current value during the simulation.

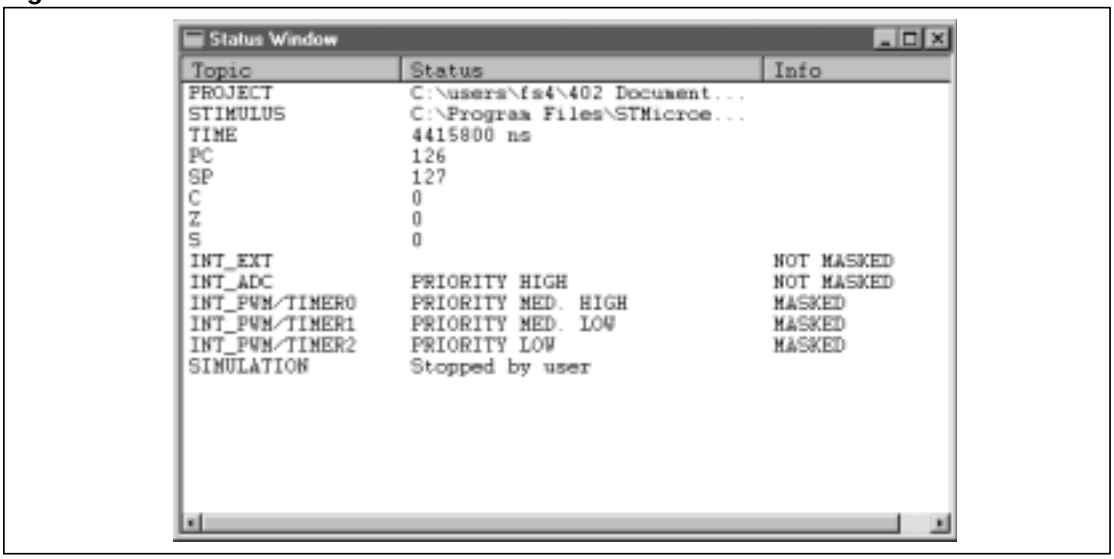
Note: *The values modification are considered just after restarting the simulation: you cannot see the modified values in the Watch Editor and in the other parts of the Debugger before that moment.*

Status Window

The Status window reports some useful information concerning the status of the debugger simulation.

You can open the Status window by selecting the STATUS WINDOW command from the DEBUGGER menu.

Fig. 14.18 - Status window



The data shown in the Status window are the following:

- The project name and path currently used by the debugger.
- The currently used Stimulus file (name and path).
- The current simulation time.
- The Program Counter (PC) current value.
- The Stack Pointer (SP) current value.
- The flags (C, Z, S) status.
- The interrupts priority level and the current status (masked, not masked)
- The action that caused the last simulation stop (user stop, breakpoint or time completed).

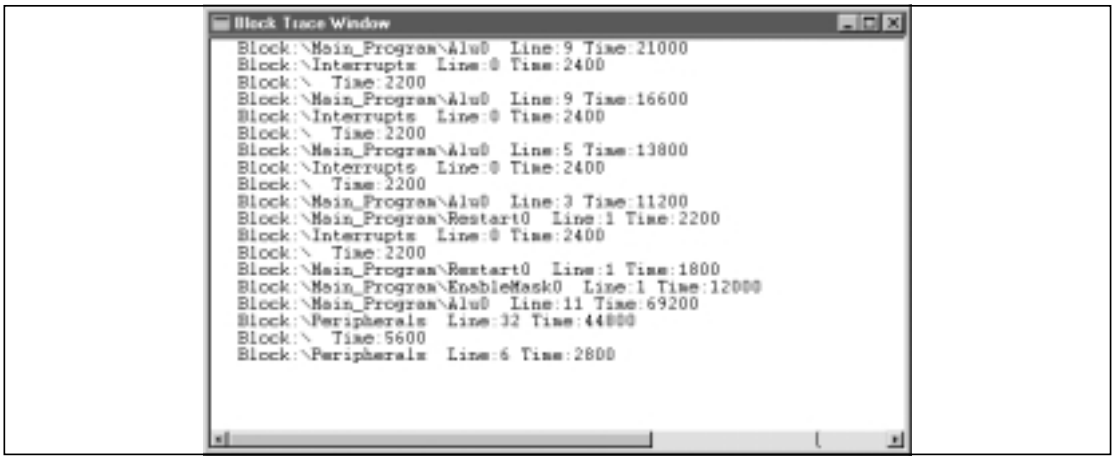
Note: *The Status window is updated only when the simulation is stopped (by the user or at the end of a time or instruction step).*

Block Trace Window

The Block Trace window shows the project blocks processed by the simulation. In addition the current line inside the block and the processing time are supplied, allowing to estimate the processing time of the single parts of the program.

To open the Block Trace window select the BLOCK TRACE command from the DEBUGGER menu.

Fig. 14.19 - Block Trace window



Note: *The last block processed is showed at the top of the list.*

Memory Dump

The Debugger environment supplies the dump of the Program Memory and Data Memory. The Program Memory Dump window and the Data Memory window show the memory location contents in hexadecimal format.

- To open the Program Memory Dump window select the command PROGRAM MEMORY DUMP from the DEBUGGER menu.

Fig. 14.20 - Program Memory Dump window

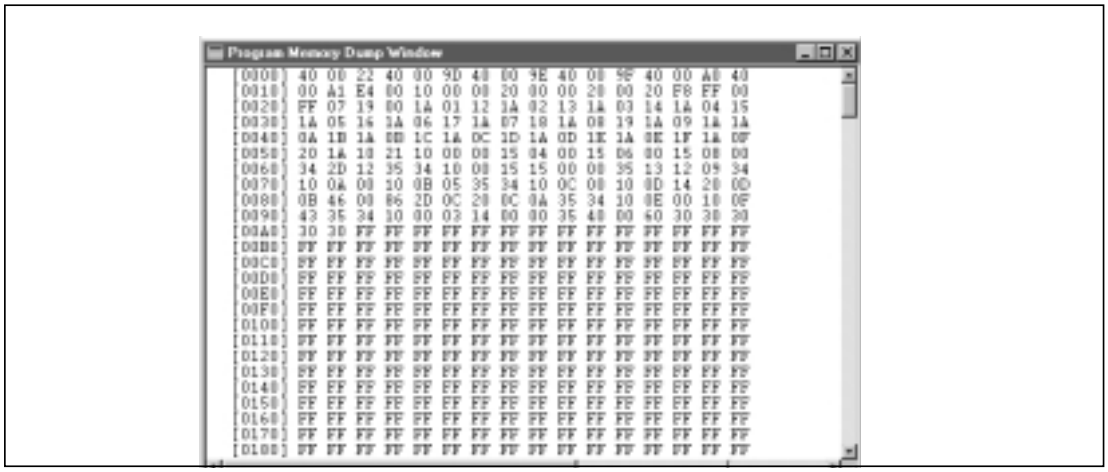
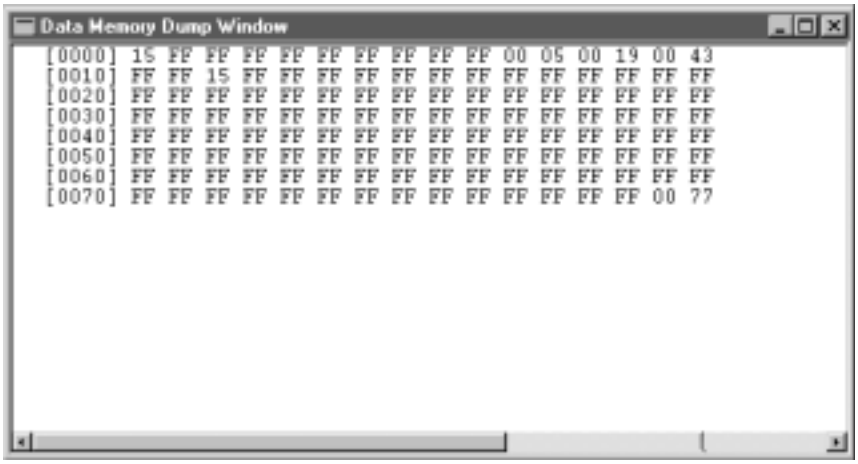


Fig. 14.21 - Data Memory Dump window



- To open the Data Memory Dump window select the command DATA MEMORY DUMP from the DEBUGGER menu.

Options

The Debugger environment allows you to customize the Font and to set the speed in the Animation mode.

To set the font:

- Select the command OPTIONS>FONT from the DEBUGGER menu.
- The standard Font dialog-box opens allowing you to select the desired font.

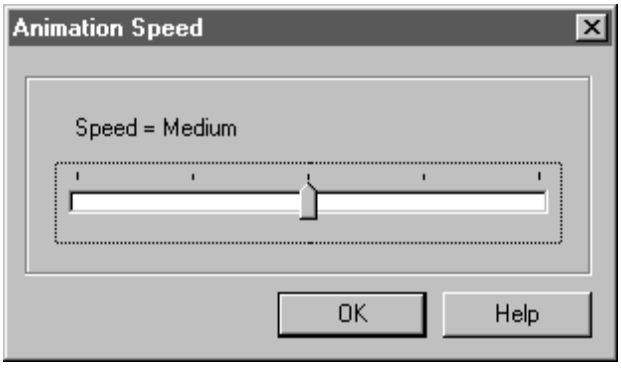
Note: the font setting affect only the current window in foreground.

To set the Animation mode speed:

- Select the command OPTIONS > ANIMATION SPEED from the DEBUGGER menu
- Scroll the bar to slower or increase the animation speed in the Animation Speed dialog-box.

Note: *There are five speeds available: very slow, slow, medium, fast, very fast.*

Fig. 14.22 - Animation Speed dialog-box



15 - DEVICE PROGRAMMING

PROGRAMMER

The Programmer tool allows you to download the machine code on the device's memory. You can also read the memory contents and read/write the ID code of the device.

The Programmer tool works together with the Programming Board supplied with FUZZYSTUDIO™4 Kit. Be sure that you are using the correct programming board for the selected target device and that it is correctly installed. Refer to the Appendix B to get more information about the Programming Board and its installation.

In this chapter you will learn to:

- launch the Programmer
- setting the download options

Moreover, the list of the Programmer messages is supplied.

Device Programming

After the machine code has been successfully completed, it is possible to program the target device inserted in the apposite board.

Make sure the device has been inserted in the correct way, that the board is turned on and check if it is connected to the computer's parallel port. Moreover, make sure the program has been compiled for the correct device and that the programming board is the suitable one for that device.

It is possible to choose and customize the operations to carry out during the downloading phase, specifying the options by using the tab-sheet accessible by means of the Download Options command. Refer to next paragraph for a detailed description of the available options.

To start the programming of the device inserted in the socket select the command PROGRAMMER>RUN from the TOOLS menu or click on the apposite toolbar button. The output window opens displaying the evolution of the downloading phases and the eventual error messages.

Device programming status messages

The list of possible messages displayed in the output window is the following one:

Blank Check
The protocol has started to verify the device is not programmed.

Blank Checking Device
The device is being verified that it is not programmed.

Device Lock
The protocol has started the lock of the device memory.

Done
The download has been successfully completed.

Done with Errors
The download has not been successfully completed because an error occurred.

ID Code Read
The protocol has started to read the ID Code.

ID Code Write
The protocol has started to write the ID Code file into the device memory.

Locking Device
The device memory is being locked .

Memory Read
The protocol has started the dumping phase of the device memory.

Memory Write
The protocol has started the writing phase into the device memory.

Reading ID Code
The ID Code is being read in the device memory.

Reading File
The binary file, containing the program to be written into the device memory, is being read.

Reading Memory
The device memory to generate the dump file is being read.

Testing Lock Bit
The device is being verified that it is not locked.

Writing File
The dump file of the device memory is being generated.

Writing ID Code
The ID Code is being written in the memory device.

Writing Memory
The device memory is being programmed.

Device programming error messages

Device is Locked

It has been attempted an operation in a locked device. The only operation allowed on a locked device is the reading of the ID Code.

Device not Blank

The inserted device has not been canceled properly or it is badly placed in the socket.

Error Reading File filename

An error has occurred during the reading of the file containing the binary code to be loaded into the device memory or during the reading of the ID Code. Check if the file exists or if it is corrupted; in the case of the ID Code, check if the name and the path have been specified correctly.

Error Writing File filename

An error has occurred during the reading of the dump file or during the generation of the ID Code. Check if there is enough space on the disk or if the file name has been correctly specified.

Out of Memory

A memory error has occurred. Try closing some open programs.

Unable to Lock Device

The device locking has not been successfully completed. The device could be either already protected or damaged.

Unable to use I/O Ports

You are trying to use the Programmer under Windows NT Operating System or a wrong parallel port address has been specified or the port is out of order.

Write Memory Error

An error has occurred during the programming of the memory device. Check if the device has been correctly inserted or if it is already programmed enabling the Black Check control. Or the device could be damaged.

Wrong Binary File

The binary code file to be loaded into the device memory is corrupted or the file format is not correct. Try to generate again the binary code file.

Programming Options

Before starting the downloading phase of the device, it is possible to specify the operations to carry out during this phase.

To open the Download Code Options tab-sheet select the item PROGRAMMER> OPTIONS from TOOLS menu or click the apposite toolbar button. Choose the “Download Options” sheet to set the most common actions to be performed during the downloading phase; select the sheet “Advanced” to perform more advanced settings.

Download Options settings

In this tab it is possible to enable/disable the following actions:

Download Binary File
Memory device programming.

Blank Check
Verify if the device has been erased.

Lock Device
Device read protection.

Fig. 15. 1 - Download Code Options dialog-box

Note: *The device reading protection prevents the reading of the Memory Program contents. A device protected in reading can be unprotected only by erasing the EPROM memory to exposure to the UV rays. The only operation that it is possible to carry out on a reading-protected device is the reading of the ID code.*

Advanced Settings

In this sheet it is possible to specify the advanced options related to the actions started during the programming phase. It is suggested to maintain unchanged the default parameters in case you are not sure about the changes to carry out.

Fig. 15. 2 - Advanced Settings

Memory Dump File
In this edit-box specify the file name including the file path in which the data read from the program memory during the downloading phase are to be found. The default name is *download.log*; if no name is specified the dump file is not created. The data are shown in hexadecimal format.

Protocol Delay
This parameter (suitable values are between the range 80 400) indicates the speed of data transmission through the parallel port. The default value is 100; specify a greater value if you want to speed up the transmission. The maximum speed you can specify depends from the computer's speed. In case of too high speed the downloading could fail. If you prefer it, it is possible to calibrate automatically the speed factor by checking the Automatic check-box.

Source ID Code File
In this edit-box you can specify the file name including the path containing the ID Code, in text format, to write in the apposite memory space of the device. The data are written during the programming phase of the device but they can be written in an already programmed device if that area has not already been programmed. If you do not want to write the memory containing the ID Code do not specify any source file. The total space available is 64 bytes; the file must be formed by 64 characters: further characters will be ignored.

Destination ID Code File

In this edit-box you can specify the file name including the path containing the ID Code, read from the apposite memory space of the device. In order not to carry out the reading of the ID Code file, do not specify any destination file. The file format is textual and is shown both in numeric and ASCII format.

I/O Port

This edit-box is used to specify the address of the parallel port connected to the board. The default address is 0X378 (LPT1).

APPENDIXES

APPENDIXES

A - FEATURES DEPENDENT ON THE TARGET DEVICE

In this Appendix are described the features that change according with the selected target device of the ST52x4xx family. For each device, the description focuses on the following items:

- Predefined Variables
- Library Functions parameters
- Peripherals Configuration sheets
- Peripherals Setting Blocks
- Interrupts Blocks
- Memory space and variables number availability
- Stimulus Editor pin names
- Exception list

ST52x420/420Gx Features

Predefined Variables

Read only Variables			
CHAN0	Channel 0 A/D converter	Address: 1	Read
CHAN1	Channel 1 A/D converter	Address: 2	Read
CHAN2	Channel 2 A/D converter	Address: 3	Read
CHAN3	Channel 3 A/D converter	Address: 4	Read
CHAN4	Channel 4 A/D converter	Address: 5	Read
CHAN5	Channel 5 A/D converter	Address: 6	Read
CHAN6	Channel 6 A/D converter	Address: 7	Read
CHAN7	Channel 7 A/D converter	Address: 8	Read
PWM_0_STATUS	Timer-PWM 0 Status Register	Address: 13	Read
PWM_1_STATUS	Timer-PWM 1 Status Register	Address: 15	Read
PWM_2_STATUS	Timer-PWM 2 Status Register	Address: 17	Read

Write only Variables			
PWM_0_RELOAD	Timer-PWM 0 Reload Register	Address: 4	Write
PWM_1_RELOAD	Timer-PWM 1 Reload Register	Address: 6	Write
PWM_2_RELOAD	Timer-PWM 2 Reload Register	Address: 8	Write

Read-Write Variables			
PORT_A	Port A Input Register	Address 9	Read
	Port A Output Register	Address 0	Write
PORT_B	Port B Input Register	Address 10	Read
	Port B Output Register	Address 1	Write
PORT_C	Port C Input Register	Address 11	Read
	Port C Output Register	Address 2	Write
PWM_0_COUNT	Timer-PWM 0 Counter	Address 12	Read
	Timer-PWM 0 Counter	Address3	Write
PWM_1_COUNT	Timer-PWM 1 Counter	Address 14	Read
	Timer-PWM 1 Counter	Address 5	Write
PWM_2_COUNT	Timer-PWM 2 Counter	Address 16	Read
	Timer-PWM 2 Counter	Address 7	Write

Other Predefined Variables

The following Predefined Variables are write-only variables to be used only in Assembler Block with the instructions LDCE and LDCR and in Arithmetic Block with the assignation operator (=) where the variable must be written only on the left side. They are used to the chip configuration.

REG_CONF0	Configuration Register 0	Address 0 (Write)
REG_CONF1	Configuration Register 1	Address 1 (Write)
REG_CONF2	Configuration Register 2	Address 2 (Write)
REG_CONF3	Configuration Register 3	Address 3 (Write)
REG_CONF4	Configuration Register 4	Address 4 (Write)
REG_CONF5	Configuration Register 5	Address 5 (Write)
REG_CONF6	Configuration Register 6	Address 6 (Write)
REG_CONF7	Configuration Register 7	Address 7 (Write)
REG_CONF8	Configuration Register 8	Address 8 (Write)
REG_CONF9	Configuration Register 9	Address 9 (Write)
REG_CONF10	Configuration Register 10	Address 10 (Write)
REG_CONF11	Configuration Register 11	Address 11 (Write)
REG_CONF12	Configuration Register 12	Address 12 (Write)
REG_CONF13	Configuration Register 13	Address 13 (Write)
REG_CONF14	Configuration Register 14	Address 14 (Write)
REG_CONF15	Configuration Register 15	Address 15 (Write)
REG_CONF16	Configuration Register 16	Address 16 (Write)

DeviceStatus() Function Parameters

The DeviceStatus library function arguments are characterized by a peripheral identifier and one parameter:

DeviceStatus(*periph, param);*

These parameters depend on the selected target device. In the ST52x420/420Gx they can be the following:

periph:

PWM_0	identifies the PWM/Timer 0
PWM_1	identifies the PWM/Timer 1
PWM_2	identifies the PWM/Timer 2

param:

SET	identifies the Set status of the PWM/Timer
RESET	identifies the Reset status of the PWM/Timer
START	identifies the Start status of the PWM/Timer
STOP	identifies the Stop status of the PWM/Timer

Note: *The parameters must be expressed in capital letters*

DeviceSet() function parameters

The DeviceSet library function arguments are characterized by a peripheral identifier and one parameter:

DeviceSet(*periph, param1);*

These parameters depend on the selected target device. In the ST52x420/420Gx they can be the following:

periph:

ADC	identifies the A/D Converter
PWM_0	identifies the PWM/Timer 0
PWM_1	identifies the PWM/Timer 1
PWM_2	identifies the PWM/Timer 2
START_PWM_TIMER	identifies all the PWM in order to start them simulaneously

param1:

Related to the the ADC, PWM_0, PWM_1, PWM_2 first parameter:

START	resets the PWM/Timer or A/D Converter
STOP	stops the PWM/Timer or A/D Converter
SET	sets the PWM/Timers
RESET	resets the PWM/Timer or A/D Converter

Related to the START_PWM_TIMERS first parameter:

1	identifies the PWM/Timer 0
2	identifies the PWM/Timer 1
4	identifies the PWM/Timer 2

The PWM/Timers to be started are identified by the sum of parameters or separating them with | (or)

param2:

Related to the the ADC first parameter:

0-n	identifies the AD channel

param3:

Related to the the ADC first parameter:

SINGLE	sets the conversion mode to a single conversion
CONTINUOUS	sets the conversion mode as continuous conversion

param4:

Related to the the ADC first parameter:

SINGLE	sets the channel mode to the single channel conversion
SEQUENCE	sets the channel mode in the sequence of channel conversion

param5:

Related to the the ADC first parameter:

FULL	sets the A/D converter frequency at the full speed
DIVIDED	sets the A/D converter frequency at the half speed

In PWM mode the SET parameter is equivalent to the START one and the RESET parameter to the STOP one. In Timer when the Start command is configured as external the SET parameter allows to put the peripheral in Set mode waiting the external start signal.

Note: some parameters can be omitted according to the other arguments and the action to perform. The parameters must be expressed in capital letters.

Interrupt Related Functions

The Standard Library supplies the following functions:

IrqEnable();	enables globally the interrupts
IrqDisable();	disables globally the interrupts
IrqReset(*int1, int2,…..*);	resets the pending interrupts
IrqEnableMask(*int1, int2,…..*);	enables the interrupts selectively
IrqPriority(int*1, int2,…..*);	sets the interrupts priority order

The interrupts identifiers int1, int2,….., can be the following:

EXTERNAL	identifies the External interrupt
AD_CONVERTER	identifies the A/D Converter interrupt
PWMTIMER0	identifies the PWM/Timer 0 interrupt
PWMTIMER1	identifies the PWM/Timer 1 interrupt
PWMTIMER2	identifies the PWM/Timer 2 interrupt

The IrqReset function arguments are the identifiers of the pending interrupts to be reset: missing interrupts are not reset.

The IrqEnableMask function arguments are the identifiers of the interrupts to be enabled: missing interrupts are disabled.

The IrqPriority function arguments are the identifiers of all the interrupt (except the external interrupt that have fixed top level priority) written with the priority order.

Note: *The identifiers must be expressed in capital letters.*

Peripherals Configuration Sheets

The Peripherals Configuration property-sheet for ST52x420/420Gx is composed by the following pages:

- Chip Clock
- Port Pins
- A/D Converter
- Watchdog
- PWM-Timer 0
- PWM-Timer 1
- PWM-Timer 2

The setting of the chip clock and of the port pins involves some changes in the configuration parameters of the other pages. For example, changing the device frequency, the available counting times for Watchdog are modified as well as some parameters of the PWM-Timers. Another example: the availability of the A/D channels number depends on the pins configured as analog input. For this reason, it is better to set the chip frequency and the port pins before the peripherals configuration.

Chip Clock sheet

In this page you can find the controls for the setting of the device clock frequency. The available frequencies in the ST52x420/Gx device are in the range 1 MHz up to 20 MHz.

To set the frequency:

- select from the drop-down list the desired frequency

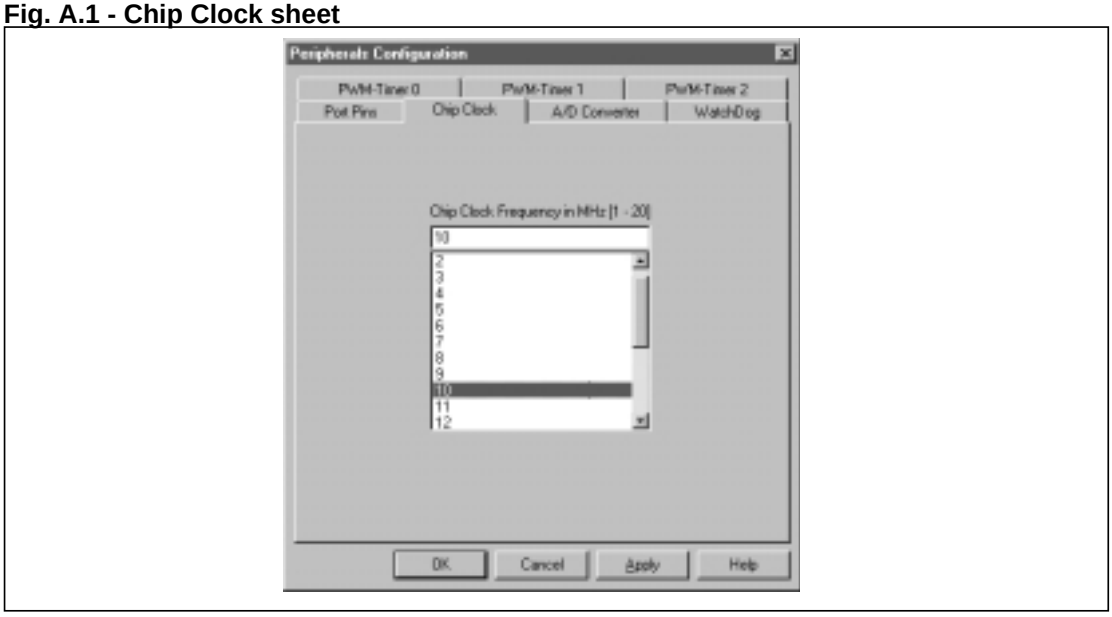
or

- write the desired frequency in the apposite text box

Default clock frequency is 5 MHz.

Changing the frequency will affect the configuration of the PWM-Timers Prescaler, Reload registers (if used) and the Watchdog counting time.

Fig. A.1 - Chip Clock sheet



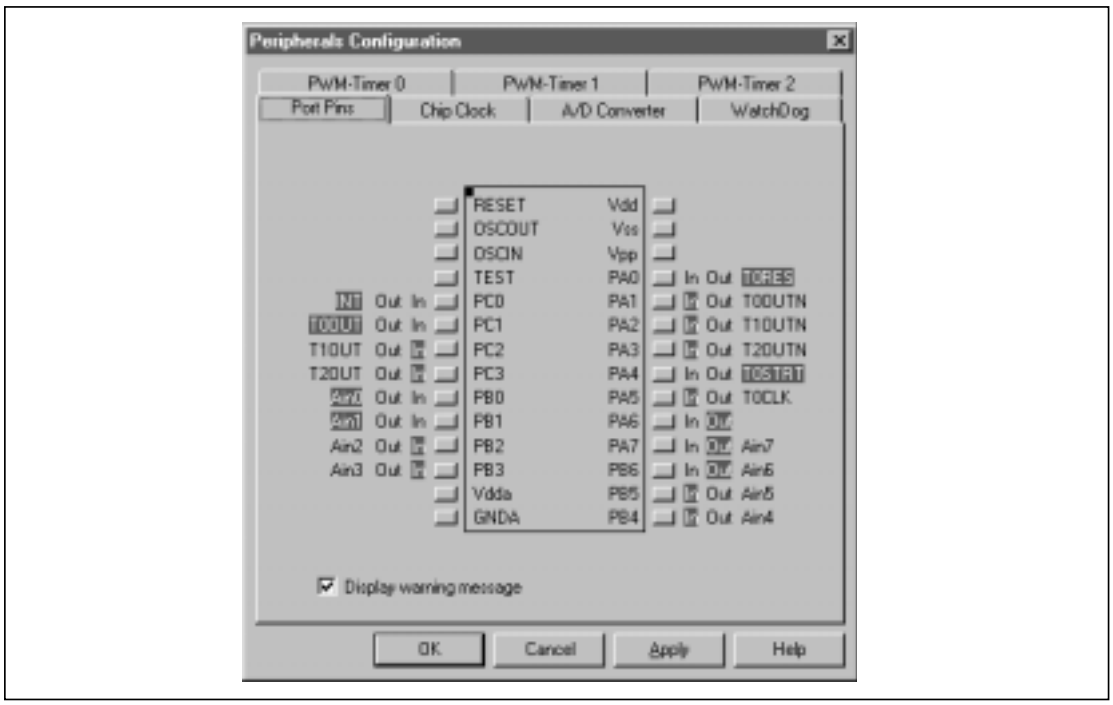
Port Pins sheet

The Port Pin sheet shows the layout of the ST52x420/420Gx device indicating the pins and the associated functionalities (input, output or alternate function).

Each pin is a button to change the configuration of the pin itself just clicking on them: the current configured function is highlighted and rotates through the available ones. Moreover, you can directly choose the pin function by clicking on the function name next to the pin.

Some pins cannot be configured (such as TEST or Vdd) and are shown just for having a complete look of the device: actually the pin position is the same as the real device.

Fig. A.2 - Port Pins sheet

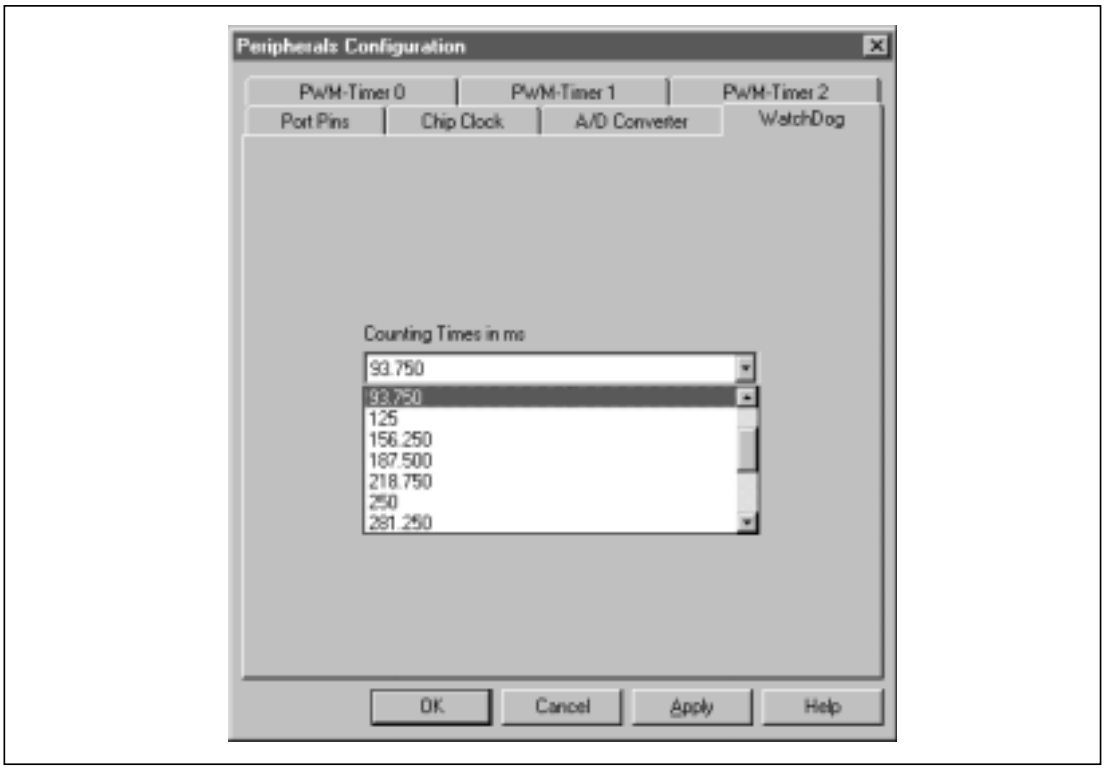


The check-box “Display warning message” allows to disable the warning messages when configuring the pins. An example of warning messages is the one that warns, when you pass from the configuration of the pin as analog input to digital input or output, that the predefined variable related to the A/D channel will not be available.

Watchdog sheet

The only setting to perform is the choice of the counting time of the Watchdog. This can be achieved selecting the time in milliseconds nearest to the desired counting time from the available drop-down list-box.

Fig. A.3 - Watchdog sheet



Note: *The contents of the list-box depends on the selected chip clock frequency.*

PWM-Timer 0 sheet

The PWM-Timer sheet is composed by several sections, each allowing the configuration of a peripheral feature.

Working Mode and Frequency Setting sections

The first choice you should perform is the Working Mode:

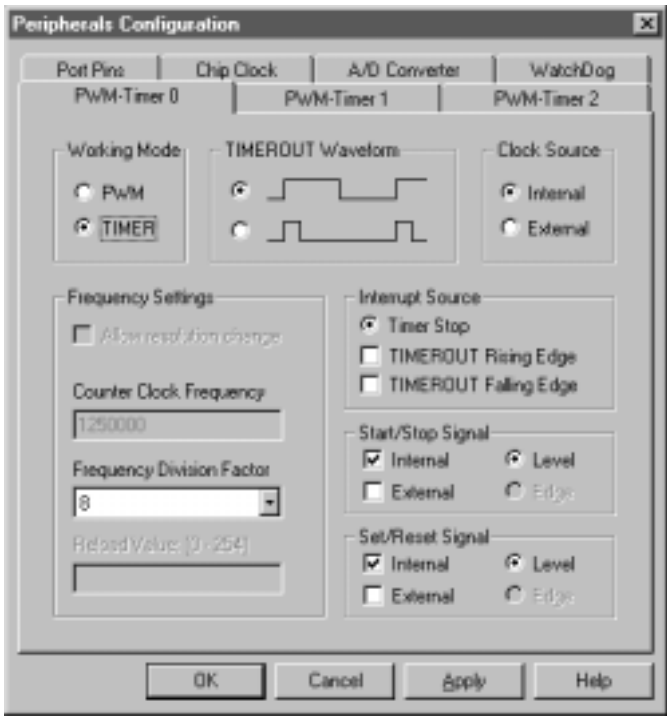
- Select PWM if you want the peripheral to work as PWM controller
- Select TIMER if you want the peripheral to work as Timer or events counter

When the peripheral is set in PWM mode, the output signal has a fixed period whose duty-cycle can be modified. The period is fixed by the Prescaler and by the contents of the Reload register; the duty-cycle is modified run-time by changing the Timer Counter register. Setting a value in the Reload register you can adjust the desired working frequency, but you may loose resolution because the counting range starts from the Reload value up to 255 (see data-sheet for further information). You can fix the control period by specifying the Working Frequency in the “PWM-Timer frequency setting” section.

- Check the box “Allow resolution change” if you want to use the Reload value
- Uncheck it if you don’t want to loose resolution.

If you allow the resolution change, you can specify directly the Working Frequency in Hz, writing the value in the apposite text-box. The values of the Prescaler Factor and of the Reload register are automatically computed and shown in the relative boxes. The specified frequency is approximated to the nearest value computed with integer values of the Prescaler and the Reload register.

Fig. A.4 - PWM-Timer 0 sheet



If you are using an external clock to drive the peripheral, the Working Frequency cannot be set directly by the user, because the clock frequency is not known. In this case the Working frequency text box is disabled and the Frequency Division factor list and the Reload Value text-boxes are enabled, in order to allow you to specify by yourself the two parameters.
If you don't allow the resolution change, you can only modify the Prescaler Factor, selecting it in the apposite drop-down list. The allowed factors are powers of two, up to 65536; doing so you can get 17 different Working Frequencies.

When the peripheral is set in Timer Mode, the signal has fixed the duty-cycle and the frequency depends on the Prescaler and Timer Counter register contents. The Reload register is not used. The frequency of the signal is equal to the clock (internal or external) frequency divided by the Prescaler and Timer Counter register values. You can fix the Division Factor of the Prescaler by specifying it in the "PWM-Timer frequency setting" section.

TIMEROUT Waveform section

The TIMEROUT Waveform section is disabled when the PWM mode is selected. This section allows to select the Timer output modality: square wave or impulse; select one of them by clicking the radiobutton next to the waveform figure.

When the square wave is selected, the Timer output will have a 50% duty-cycle. Otherwise the Timer output will have an impulse as long as the clock period multiplied by the Prescaler factor (see data-sheet for further information).

Clock Source section

This section allows to choose the input of the Prescaler: the internal clock, whose frequency is equal to the oscillator frequency, or the external clock applied on the T0CLK pin. Click the radiobutton next to the desired option to select one of them.

Note: *When in PWM mode, if external clock is selected, the PWM-Timer frequency setting section changes as described previously*

Interrupt Source section

This section allows you to select the source of the PWM-Timer 0 interrupt. The available choices are:

- *Timer Stop:* when the peripheral is stopped by the program or by an external signal, in PWM or Timer mode.
- *TIMEROUT rising edge*: when the signal generated by the peripheral has the transition from high to low. The interrupt request is generated when the counter starts and when the end of count is reached.
- *TIMEROUT falling edge*: when the signal generated by the peripheral has the transition from low to high. The interrupt request is generated when the half-count is reached.
- *TIMEROUT falling and rising edge*: selecting both previous sources, the interrupt request is generated on both edges of the TIMEROUT signal.

To select one or more sources, check the corresponding check-boxes in the Interrupt Source section.

Note: *Selecting Timer Stop source, the other sources cannot be activated and vice-versa, so the related check-boxes are disabled. If no interrupt source has been selected, the interrupt on Timer Stop is automatically assumed: when you open again the sheet, you will find this option automatically selected.*

Start Signal and Reset Signal sections

These sections are organized in the same way. They are used to set the source of the Start/Stop and Set/Reset signals of the PWM-Timer 0.

A couple of check-boxes (Internal and/or external) and a couple of radiobuttons (level or edge) compose each section.

The Start/Stop and the Set/Reset signals can be generated by the program by using the peripheral block (Internal), or by signals applied on the external pins (T0STRT and T0RES).

- Check the “Internal” check-box if you want to apply the signals from the program.
- Check the “External” check-box if you want to apply the signals to the external pins.
- Check both check-boxes if you want to apply the signals in both ways.

When the External mode is selected, the signal can be specified as edge sensitive or level sensitive. When the Internal mode is selected, only the level sensitive mode can be set, so the peripheral block acts in on/off way.

- Click the “Level” radiobutton if you want to toggle the peripheral according to the level of the signal.
- Click the “Edge” radiobutton if you want to toggle the peripheral on the rising edges of the signal (when only External mode is selected).

PWM-Timer 1 & 2 sheets

The configuration sheets for the PWM-Timer 1 and 2 are the same as the PWM-Timer 0 one. Because these PWM-Timers cannot be driven from the external pins, all the sections related to the configuration of the external functions are disabled. The disabled sections are:

- Clock Source
- Start Signal
- Reset Signal

The other section works normally as described previously for the PWM-Timer 0.

A/D Converter sheet

To configure the A/D Converter you have to specify the following characteristics:

- The conversion mode
- The channel conversion sequence
- The channel or the sequence of the channels to be converted.

In the “Conversion” section click:

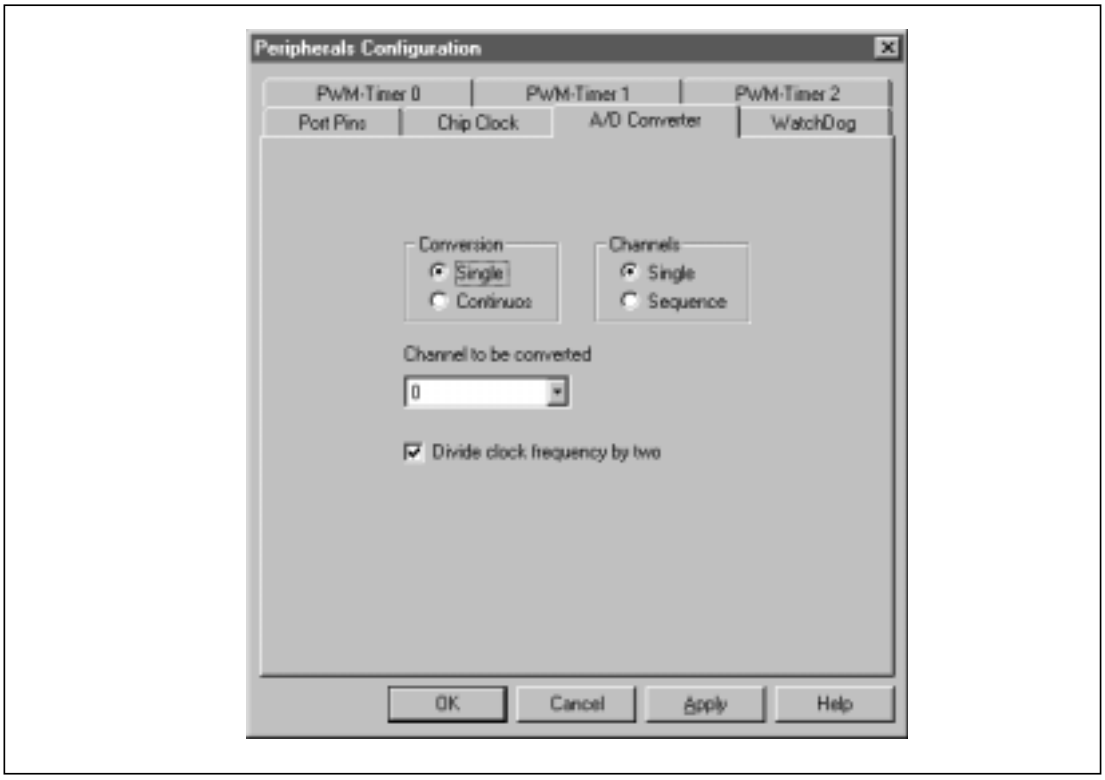
- “Single” radiobutton if you want to perform a single conversion cycle
- “Continuous” radiobutton if you want the peripheral to work continuously until it is stopped.

Note: *When “Single” mode is selected, after the conversion, you must start again the peripheral, with the peripheral block, if you want another conversion cycle.*

In the “Channel” section click:

- “Single” radiobutton if you want the conversion of the channel specified in the “Channel to be converted” section.
- “Sequence” radiobutton if you want the conversion of the channels sequence from the first one up to the one specified in the “Last Channel to be converted” section.

Fig. A.5 - A/D Converter sheet



The next section is called in two ways, according with the previous settings:

- “Channel to be converted” if the “Single” radiobutton has been selected in the “Channel” section.
- “Last Channel to be converted” if the “Sequence” radiobutton has been selected in the “Channel” section.

In both cases, you have to select the channel number from the drop-down list-box available in this section.

Note: *The contents of the channels numbers list depends on the pin configured as analog input: some A/D configuration are not allowed if the pins have not been configured correctly. If no pin has been configured as analog input, the entire sheet is disabled.*

Peripherals Setting Blocks

The Peripherals Setting Blocks for ST52x420/420Gx set the following peripherals:

- A/D Converter
- Watchdog
- PWM-Timer 0
- PWM-Timer 1
- PWM-Timer 2

In the following you can find the description of each block.

A/D Converter setting block

The A/D setting block is the most complex, because it allows to change run-time the whole configuration of the peripheral. The sections that you can find in the dialog-box allows to perform the following actions:

- Start or Stop the peripheral
- Set or Reset the peripheral
- Change the conversion mode
- Change the channel conversion sequence
- Choose the channel or the sequence of the channels to be converted
- Change the A/D working frequency

To Start/Stop the peripheral, click the corresponding radiobutton at the top of the dialog box.

If you intend to change run-time the A/D converter configuration, uncheck the “Use Default Configuration” check-box and follow the instructions described below:

In the “Conversion” section click:

- “Single” radiobutton if you want to perform a single conversion cycle
- “Continuous” radiobutton if you want the peripheral to work continuously until it is stopped.

Note: *When “Single” mode is selected, after the conversion, you must start again the peripheral, with the peripheral block, if you want another conversion cycle.*

In the “Channel” section click:

- “Single” radiobutton if you want the conversion of the channel specified in the “Channel to be converted” section.
- “Sequence” radiobutton if you want the conversion of the channels sequence from the first one up to the one specified in the “Last Channel to be converted” section.

The next section is called in two ways, according with the previous settings:

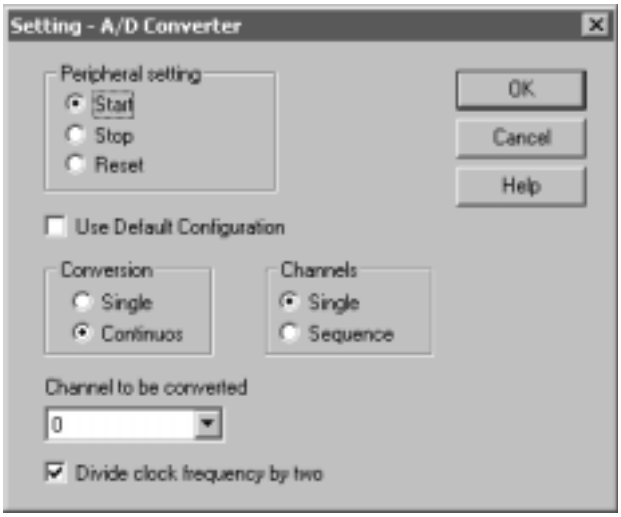
- “Channel to be converted” if the “Single” radiobutton has been selected in the “Channel” section.
- “Last Channel to be converted” if the “Sequence” radiobutton has been selected in the “Channel” section.

In both cases, you have to select the channel number from the drop-down list-box available in this section.

Note: *The contents of the channels’ numbers list depends on the pin configured as analog input: some A/D configurations are not allowed if the pins have not been configured correctly. If no pin has been configured as analog input, the entire sheet is disabled.*

The A/D Converter can work both with the chip clock frequency or with half of this frequency. Check the apposite check-box if you want to slower to the half the A/D frequency and conversion time.

Fig. A.6 - A/D Converter Setting Block



Watchdog Setting block

The Watchdog setting block is used to Start/Refresh the Watchdog counter or to disable the peripheral.

- Click the “Start/Refresh” radiobutton if you want to start the peripheral of refresh the counter to avoid the device reset.
- Click the “Disable” radiobutton to disable the peripheral.

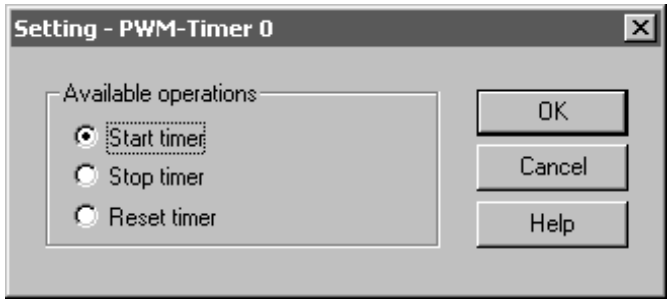
Fig. A.7 - Watchdog Setting block

PWM-Timer 0, 1 & 2 setting blocks

All the PWM-Timer setting blocks are the same; each of them is used to Start/Stop or Set/Reset the corresponding peripheral.

- To Start/Stop or Reset the peripheral, click the corresponding radiobutton.

Fig. A.8 - PWM-Timer Setting block

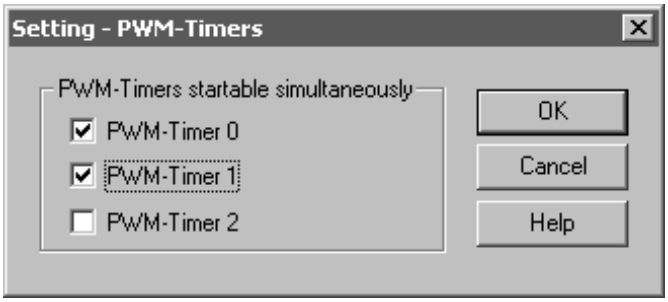


Note: *The available radiobuttons are the only allowed to the peripheral settings. For instance, if the Set/Reset and Start/Stop signal of the PWM/Timer0 are set bothe as external, none of the radiobuttons are made available. Moreover, when the PWM mode is set for the peripheral, the only available radiobuttons are the Start and Reset radiobuttons.*

In addition, with the PWM-Timers setting block you can start two or all the PWM-Timers at the same time by checking the corresponding check-box.

Note: *If the PWM/Timer0 is set with the Start/Stop or Set/reset signals configured as external, the corresponding check-box is disabled.*

Fig. A.9 - PWM-Timers Setting Block



Blocks Related to the Interrupts

The interrupts sources managed in the ST52x420/420Gx are the following:

- External Interrupt
- A/D Converter Interrupt
- PWM-Timer 0 Interrupt
- PWM-Timer 1 Interrupt
- PWM-Timer 2 Interrupt

Each interrupt can be recognized by its name in the list-boxes (see related chapters in this manual).

All interrupts pending can be reset and all sources can be masked. The priority level of the External Interrupt is fixed to the maximum and cannot be changed. The default priority order is the following:

- External Interrupt
- PWM-Timer 0
- PWM-Timer 1
- PWM-Timer 2
- A/D Converter

Note: *All interrupts are disabled by default.*

Memory Spaces

The available memory spaces in the ST52x420/420Gx devices are the following:

	ST52x420	ST52x420G0	ST52x420G1	ST52x420G2
Program Memory (EPROM)	4 Kbytes	1 Kbyte	2 kbytes	4 Kbytes
Program Memory available for MF	1 Kbyte	1 Kbyte	1 Kbyte	1 Kbyte
Data Memory	128 bytes	128 bytes	128 bytes	128 bytes

These resources allow you to define:

- programs whose length should be up to 4 Kbytes
- up to 335 Membership Functions

The Program Memory shares data tables, Membership Functions data and the program instruction; for this reason, the effective available space for program instruction depends on the memory space used by data and vice-versa.

The number of the user variables depends not only on the available RAM location, but also on the stack levels used by the CALL instructions and by the nested interrupt requests. Actually, each jump to a subroutine or interrupt service routine uses as system stack two RAM locations, starting from the last ones in the RAM memory space. To avoid overlapping between data and system stack, the number of defined variables should be less than the remaining memory locations (see ST52x420 datasheet for further details).

In addition, the first two RAM locations are used by the Compiler macros as working register, so the total variables number that can be defined is lowered by two.

Example: if your program has max 5 levels of nested calls to subroutine and interrupts and the Compiler reserves two RAM locations. The available variables number is:

128 – (5 * 2) – 2 = 116 variables available.

Pin names to be used in the Stimulus file

In the following you can find the list of the ST52x420/420Gx pins names used with the stimulus file.

PA0	Port A pin 0; also used for driving T0RES signal
PA1	Port A pin 1; also used for driving T0OUTN signal
PA2	Port A pin 2; also used for driving T1OUTN signal
PA3	Port A pin 3; also used for driving T2OUTN signal
PA4	Port A pin 4; also used for driving T0STRT signal
PA5	Port A pin 5; also used for driving T0CLK signal
PA6	Port A pin 6
PA7	Port A pin 7
PB0	Port B pin 0
PB1	Port B pin 1
PB2	Port B pin 2
PB3	Port B pin 3
PB4	Port B pin 4
PB5	Port B pin 5
PB6	Port B pin 6
PC0	Port C pin 0
PC1	Port C pin 1
PC2	Port C pin 2
PC3	Port C pin 3
Ain0	Analog input channel 0
Ain1	Analog input channel 1
Ain2	Analog input channel 2
Ain3	Analog input channel 3
Ain4	Analog input channel 4
Ain5	Analog input channel 5
Ain6	Analog input channel 6
Ain7	Analog input channel 7

Note: *The signal names are case-sensitive: Ain0 is different from ain0 or AIN0.*

Debugger Exceptions list

In the following you can find the list of the Exceptions message and codes related to ST52x420/420Gx emulator.

001	RAM address out of range [0-127]
002	EPROM address out of range [0-4095]
003	Input register address out of range [0-17]
004	Output register address out of range [0-8]
005	Configuration register address out of range [0-16]
006	Fuzzy input register address out of range [0-7]
007	EPROM location pointed by PC does not correspond to any instruction code
008	The reset of a not pending interrupt has been tried
009	The Watchdog has been disabled when already disabled
010	Division by zero
011	Pin is set in input mode but there is no signal that drives it
012	Pin is set in output mode but it is forced by a signal defined for it
013	Inconsistent binary file
014	Timer 0 clock is set as external but the pin PA5 is set in output mode
015	Timer 0 reset is set as external but the pin PA0 is set in output mode
016	Timer 0 start/stop is set as external but the pin PA4 is set in output mode
017	Timer 0 clock is set as external but the signal defined for pin PA5 is not a clock
018	Pin is configured as PWM/Timer output but it is set in input mode
019	The A/D Converter tried to convert the i-th analog signal but the pin PBi is set in output mode
020	The A/D Converter tried to convert the i-th analog signal but the pin PBi is set as Digital I/O
021	The A/D Converter tried to convert the seventh analog signal but the pin Pin 18 (PA7 or PB7) is set to belong to the port A
022	WAIT instruction has been executed but all interrupt sources are disabled
023	The use of an EPROM location has been tried but the PGSET instruction was not specified before.
024	The use of an EPROM location has been tried but an instruction different from PGSET or an interrupt asserviment has modified the previously set page.

ST52x430Kx Features

Predefined Variables

Read only Variables			
CHAN0	Channel 0 A/D converter	Address: 1	Read
CHAN1	Channel 1 A/D converter	Address: 2	Read
CHAN2	Channel 2 A/D converter	Address: 3	Read
CHAN3	Channel 3 A/D converter	Address: 4	Read
CHAN4	Channel 4 A/D converter	Address: 5	Read
CHAN5	Channel 5 A/D converter	Address: 6	Read
CHAN6	Channel 6 A/D converter	Address: 7	Read
CHAN7	Channel 7 A/D converter	Address: 8	Read
PWM_0_STATUS	Timer-PWM 0 Status Register	Address: 13	Read
PWM_1_STATUS	Timer-PWM 1 Status Register	Address: 15	Read
PWM_2_STATUS	Timer-PWM 2 Status Register	Address: 17	Read
SCI_RX_DATA	SCI received data	Address: 18	Read
SCI_STATUS	SCI Status Register	Address: 19	Read

Write only Variables			
PWM_0_RELOAD	Timer-PWM 0 Reload Register	Address: 4	Write
PWM_1_RELOAD	Timer-PWM 1 Reload Register	Address: 6	Write
PWM_2_RELOAD	Timer-PWM 2 Reload Register	Address: 8	Write
SCI_TX_DATA	SCI transmitted data	Address: 9	Write

Read-Write Variables			
PORT_A	Port A Input Register	Address 9	Read
	Port A Output Register	Address 0	Write
PORT_B	Port B Input Register	Address 10	Read
	Port B Output Register	Address 1	Write
PORT_C	Port C Input Register	Address 11	Read
	Port C Output Register	Address 2	Write
PWM_0_COUNT	Timer-PWM 0 Counter	Address 12	Read
	Timer-PWM 0 Counter	Address3	Write
PWM_1_COUNT	Timer-PWM 1 Counter	Address 14	Read
	Timer-PWM 1 Counter	Address 5	Write
PWM_2_COUNT	Timer-PWM 2 Counter	Address 16	Read
	Timer-PWM 2 Counter	Address 7	Write

Other Predefined variables:

The following Predefined Variable are write-only variables to be used only in Assembler Block with the instructions LDCE and LDCR and in Arithmetic Block with the assignation operator (=) where the variable must be written only on the left side. They are used to the chip configuration.

REG_CONF0	Configuration Register 0	Address 0 (Write)
REG_CONF1	Configuration Register 1	Address 1 (Write)
REG_CONF2	Configuration Register 2	Address 2 (Write)
REG_CONF3	Configuration Register 3	Address 3 (Write)
REG_CONF4	Configuration Register 4	Address 4 (Write)
REG_CONF5	Configuration Register 5	Address 5 (Write)
REG_CONF6	Configuration Register 6	Address 6 (Write)
REG_CONF7	Configuration Register 7	Address 7 (Write)
REG_CONF8	Configuration Register 8	Address 8 (Write)
REG_CONF9	Configuration Register 9	Address 9 (Write)
REG_CONF10	Configuration Register 10	Address 10 (Write)
REG_CONF11	Configuration Register 11	Address 11 (Write)
REG_CONF12	Configuration Register 12	Address 12 (Write)
REG_CONF13	Configuration Register 13	Address 13 (Write)
REG_CONF14	Configuration Register 14	Address 14 (Write)
REG_CONF15	Configuration Register 15	Address 15 (Write)
REG_CONF16	Configuration Register 16	Address 16 (Write)
REG_CONF17	Configuration Register 17	Address 17 (Write)
REG_CONF18	Configuration Register 18	Address 18 (Write)
REG_CONF19	Configuration Register 19	Address 19 (Write)
REG_CONF20	Configuration Register 20	Address 20 (Write)

DeviceStatus() Function Parameters

The DeviceStatus library function arguments are characterized by a peripheral identifier and one parameter:

DeviceStatus(*periph, param);*

These parameters depend on the selected target device. In the ST52x430Kx they can be the following:

periph:

PWM_0	identifies the PWM/Timer 0
PWM_1	identifies the PWM/Timer 1
PWM_2	identifies the PWM/Timer 2
SCI	identifies the SCI peripheral

param:

SET	identifies the Set status of the PWM/Timer
RESET	identifies the Reset status of the PWM/Timer
START	identifies the Start status of the PWM/Timer
STOP	identifies the Stop status of the PWM/Timer

Related to the SCI first parameter:

TX_END	identifies the transmission end flag
TX_EMPTY	identifies the transmission buffer empty flag
NINTH_BIT	identifies the received ninth data bit
OVERRUN	identifies the overrun error condition flag
RX_FULL	identifies the reception register full flag
FRAME_ERR	identifies the frame error flag
NOISE_ERR	identifies the noise error flag

Note: *The parameter must be expressed in capital letters.*

DeviceSet() function parameters

The DeviceSet library function arguments are characterized by a peripheral identifier and one parameter:

DeviceSet(*periph, param1, param2, param3, param4);*

These parameters depend on the selected target device. In the ST52x430Kx they can be the following:

***periph*:**

ADC	identifies the A/D Converter
PWM_0	identifies the PWM/Timer 0
PWM_1	identifies the PWM/Timer 1
PWM_2	identifies the PWM/Timer 2
START_PWM_TIMER	identifies all the PWM in order to start them simulaneously

***param1*:**

Related to the the ADC, PWM_0, PWM_1, PWM_2 first parameter:

START	resets the PWM/Timer or A/D Converter
STOP	stops the PWM/Timer or A/D Converter
SET	sets the PWM/Timers
RESET	resets the PWM/Timer or A/D Converter

Related to the START_PWM_TIMERS first parameter:

1	identifies the PWM/Timer 0
2	identifies the PWM/Timer 1
4	identifies the PWM/Timer 2

The PWM/Timers to be started are identified by the sum of parameters or separating them with | (or)

Related to the SCI first parameter:

TX_START	starts serial transmission
TX_STOP	stops serial transmission
NONE	indicates no action in transmission

***Param2*:**

Related to the SCI first parameter:

RX_START	starts serial reception
RX_STOP	stops serial reception
NONE	indicates no action in reception

Related to the the ADC first parameter:

0-n	identifies the AD channel

***Param3*:**

Related to the SCI first parameter:

NINTH_BIT_0	sets the ninth data bit to 0
NINTH_BIT_1	sets the ninth data bit to 1

Related to the the ADC first parameter:

SINGLE	sets the conversion mode to a single conversion
CONTINUOUS	sets the conversion mode as continuous conversion

param4:

Related to the the ADC first parameter:

SINGLE	sets the channel mode to the single channel conversion
SEQUENCE	sets the channel mode in the sequence of channel conversion

param5:

Related to the the ADC first parameter:

FULL	sets the A/D converter frequency at the full speed
DIVIDED	sets the A/D converter frequency at the half speed

In PWM mode the SET parameter is equivalent to the START one and the RESET parameter to the STOP one. In Timer when the Start command is configured as external the SET parameter allows to put the peripheral in Set mode waiting the external start signal.

Note: *Some parameters can be omitted according to the other arguments and the action to perform. The parameters must be expressed in capital letters.*

Interrupt Related Functions

The Standard Library supplies the following functions:

IrqEnable();	enables globally the interrupts
IrqDisable();	disables globally the interrupts
IrqReset(*int1, int2,…..*);	resets the pending interrupts
IrqEnableMask(*int1, int2,…..*);	enables the interrupts selectively
IrqPriority(int*1, int2,…..*);	sets the interrupts priority order

The interrupts identifiers int1, int2,……, can be the following:

EXTERNAL	identifies the External interrupt
AD_CONVERTER	identifies the A/D Converter interrupt
PWMTIMER0	identifies the PWM/Timer 0 interrupt
PWMTIMER1	identifies the PWM/Timer 1 interrupt
PWMTIMER2	identifies the PWM/Timer 2 interrupt
SCI	identifies the SCI interrupt

The IrqReset function arguments are the identifiers of the pending interrupts to be reset: missing interrupts are not reset.

The IrqEnableMask function arguments are the identifiers of the interrupts to be enabled: missing interrupts are disabled. In addition it is possible to further specify one of these identifiers to set the polarity of the external interrupt (if missing default is assumed):

RISING	sets the interrupt polarity on the rising edge of the applied signal
FALLING	sets the interrupt polarity on the falling edge of the applied signal

The IrqPriority function arguments are the identifiers of all the interrupt (except the external interrupt that have fixed top level priority) written with the priority order.

Note: *The identifiers must be expressed in capital letters.*

Peripherals Configuration Sheets

The Peripherals Configuration property-sheet for ST52x430Kx is composed by the following pages:

- Chip Clock
- Port Pins
- A/D Converter
- Watchdog
- PWM-Timer 0
- PWM-Timer 1
- PWM-Timer 2
- SCI

The setting of the chip clock and of the port pins involves some changes in the configuration parameters of the other pages. For example, changing the device frequency, the available counting times for Watchdog are modified as well as some parameters of the PWM-Timers. Another example: the availability of the A/D channels number depends on the pins configured as analog input. For this reason, it is better to set the chip frequency and the port pins before the peripherals configuration.

Chip Clock sheet

In this page you can find the controls for the setting of the device clock frequency. The available frequencies in the ST52x430Kx device are in the range 1 MHz up to 20 MHz.

To set the frequency:

- select from the drop-down list the desired frequency

or

- write the desired frequency in the apposite text box

Fig.A.10- Chip Clock sheet

Default clock frequency is 5 MHz.

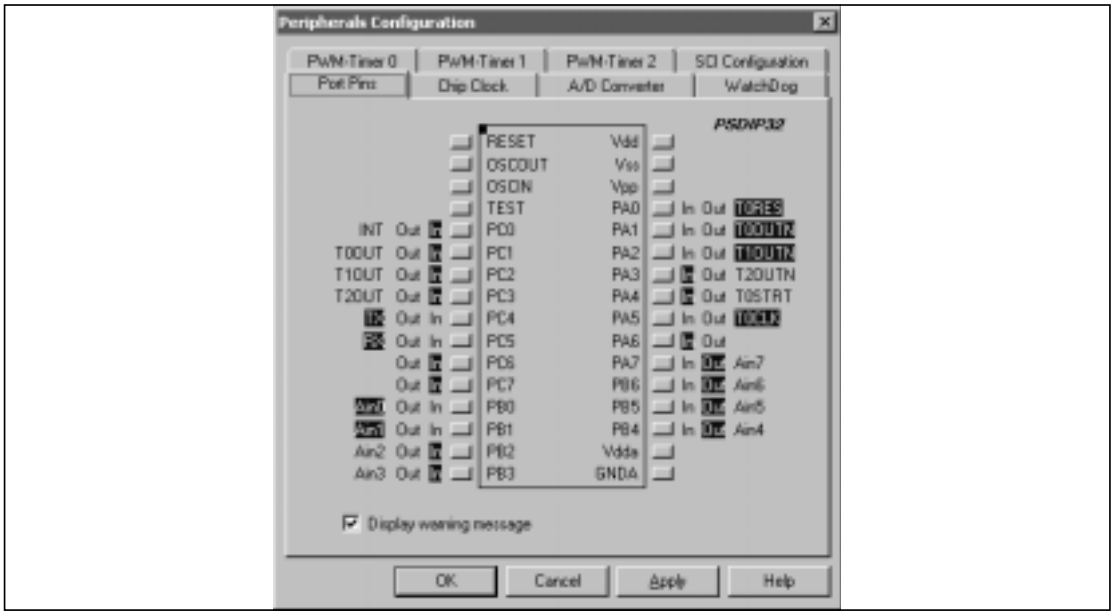
Changing the frequency will affect the configuration of the PWM-Timers Prescaler, Reload registers (if used) and the Watchdog counting time.

Port Pins sheet

The Port Pin sheet shows the layout of the ST52x430Kx device indicating the pins and the associated functionalities (input, output or alternate function).

Each pin is a button to change the configuration of the pin itself just clicking on them: the cur-

Fig. A.11 - Port pins sheet



rent configured function is highlighted and rotates through the available ones. Moreover, you can directly choose the pin function by clicking on the function name next to the pin.

Some pins cannot be configured (such as TEST or Vdd) and are shown just for having a complete look of the device: actually the pin position is the same as the real device.

The check-box “Display warning message” allows to disable the warning messages when configuring the pins. An example of warning messages is the one that warns, when you pass from the configuration of the pin as analog input to digital input or output, that the predefined variable related to the A/D channel will not be available.

Watchdog sheet

The only setting to perform is the choice of the counting time of the Watchdog. This can be achieved selecting the time in milliseconds nearest to the desired counting time from the available drop-down list-box.

Note: *The contents of the list-box depends on the selected chip clock frequency.*

Fig.A.12 - Watchdog sheet

PWM-Timer 0 sheet

The PWM-Timer sheet is composed by several sections, each allowing the configuration of a peripheral feature.

Working Mode and Frequency Setting sections

The first choice you should perform is the Working Mode:

- Select PWM if you want the peripheral to work as PWM controller
- Select TIMER if you want the peripheral to work as Timer or events counter

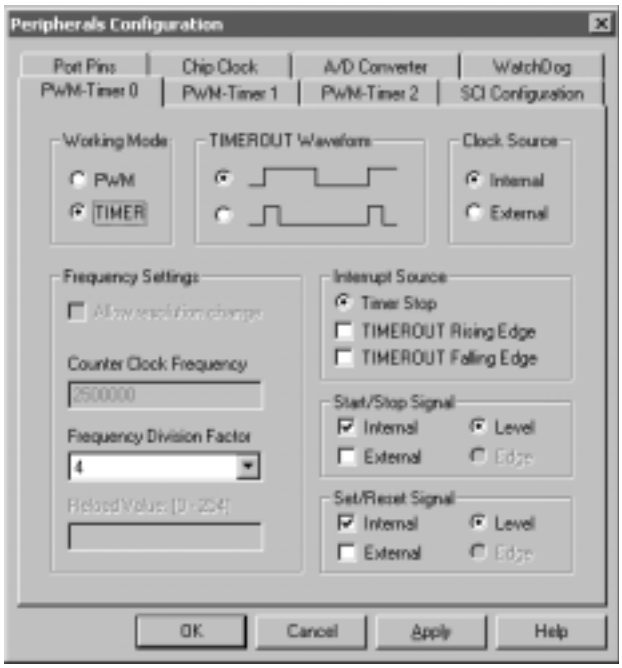
When the peripheral is set in PWM mode, the output signal has a fixed period whose duty-cycle can be modified. The period is fixed by the Prescaler and by the contents of the Reload register; the duty-cycle is modified run-time by changing the Timer Counter register. Setting a value in the Reload register you can adjust the desired working frequency, but you may loose resolution because the counting range starts from the Reload value up to 255 (see data-sheet for further information). You can fix the control period by specifying the Working Frequency in the “PWM-Timer frequency setting” section.

- Check the box “Allow resolution change” if you want to use the Reload value
- Uncheck it if you don’t want to loose resolution.

If you allow the resolution change, you can specify directly the Working Frequency in Hz, writing the value in the apposite text-box. The values of the Prescaler Factor and of the Reload register are automatically computed and shown in the relative boxes. The specified frequency is approximated to the nearest value computed with integer values of the Prescaler and the Reload register.

If you are using an external clock to drive the peripheral, the Working Frequency cannot be set directly by the user, because the clock frequency is not known. In this case the Working frequency text box is disabled and the Frequency Division factor list and the Reload Value text-boxes are enabled, in order to allow you to specify by yourself the two parameters.

Fig. A.13 - PWM-Timer 0 sheet



If you don't allow the resolution change, you can only modify the Prescaler Factor, selecting it in the apposite drop-down list. The allowed factors are powers of two, up to 65536; doing so you can get 17 different Working Frequencies.

When the peripheral is set in Timer Mode, the signal has fixed the duty-cycle and the frequency depends on the Prescaler and Timer Counter register contents. The Reload register is not used. The frequency of the signal is equal to the clock (internal or external) frequency divided by the Prescaler and Timer Counter register values. You can fix the Division Factor of the Prescaler by specifying it in the "PWM-Timer frequency setting" section.

TIMEROUT Waveform section

The TIMEROUT Waveform section is disabled when the PWM mode is selected. This section allows to select the Timer output modality: square wave or impulse; select one of them by clicking the radiobutton next to the waveform figure.

When the square wave is selected, the Timer output will have a 50% duty-cycle. Otherwise the Timer output will have an impulse as long as the clock period multiplied by the Prescaler factor (see ST52x430 data-sheet for further information).

Clock Source section

This section allows to choose the input of the Prescaler: the internal clock, whose frequency is equal to the oscillator frequency, or the external clock applied on the T0CLK pin. Click the radiobutton next to the desired option to select one of them.

Note: *When in PWM mode, if external clock is selected, the PWM-Timer frequency setting section changes as described previously.*

Interrupt Source section

This section allows you to select the source of the PWM-Timer 0 interrupt. The available choices are:

- *Timer Stop:* when the peripheral is stopped by the program or by an external signal, in PWM or Timer mode.
- *TIMEROUT rising edge*: when the signal generated by the peripheral has the transition from high to low. The interrupt request is generated when the counter starts and when the end of count is reached.
- *TIMEROUT falling edge*: when the signal generated by the peripheral has the transition from low to high. The interrupt request is generated when the half-count is reached.
- *TIMEROUT falling and rising edge*: selecting both previous sources, the interrupt request is generated on both edges of the TIMEROUT signal.

To select one or more sources, check the corresponding check-boxes in the Interrupt Source section.

Note: *Selecting Timer Stop source, the other sources cannot be activated and vice-versa, so the related check-boxes are disabled. If no interrupt source has been selected, the interrupt on Timer Stop is automatically assumed: when you open again the sheet, you will find this option automatically selected.*

Start Signal and Reset Signal sections

These sections are organized in the same way. They are used to set the source of the Start/Stop and Set/Reset signals of the PWM-Timer 0.

A couple of check-boxes (Internal and/or external) and a couple of radiobuttons (level or edge) compose each section.

The Start/Stop and the Set/Reset signals can be generated by the program by using the peripheral block (Internal), or by signals applied on the external pins (T0STRT and T0RES).

- Check the “Internal” check-box if you want to apply the signals from the program.
- Check the “External” check-box if you want to apply the signals to the external pins.
- Check both check-boxes if you want to apply the signals in both ways.

When the External mode is selected, the signal can be specified as edge sensitive or level sensitive. When the Internal mode is selected, only the level sensitive mode can be set, so the peripheral block acts in on/off way.

- Click the “Level” radiobutton if you want to toggle the peripheral according to the level of the signal.
- Click the “Edge” radiobutton if you want to toggle the peripheral on the rising edges of the signal (when only External mode is selected).

PWM-Timer 1 & 2 sheets

The configuration sheets for the PWM-Timer 1 and 2 are the same as the PWM-Timer 0 one. Because these PWM-Timers cannot be driven from the external pins, all the sections related to the configuration of the external functions are disabled. The disabled sections are:

- Clock Source
- Start Signal
- Reset Signal

The other section works normally as described previously for the PWM-Timer 0.

A/D Converter sheet

To configure the A/D Converter you have to specify the following characteristics:

- The conversion mode
- The channel conversion sequence
- The channel or the sequence of the channels to be converted.

In the “Conversion” section click:

- “Single” radiobutton if you want to perform a single conversion cycle
- “Continuous” radiobutton if you want the peripheral to work continuously until it is stopped.

Note: *When “Single” mode is selected, after the conversion, you must start again the peripheral, with the peripheral block, if you want another conversion cycle.*

In the “Channel” section click:

- “Single” radiobutton if you want the conversion of the channel specified in the “Channel to be converted” section.
- “Sequence” radiobutton if you want the conversion of the channels sequence from the first one up to the one specified in the “Last Channel to be converted” section.

The last section is called in two ways, according with the previous settings:

- “Channel to be converted” if the “Single” radiobutton has been selected in the “Channel” section.
- “Last Channel to be converted” if the “Sequence” radiobutton has been selected in the “Channel” section.

In both cases, you have to select the channel number from the drop-down list-box available in this section.

Note: *The contents of the channels numbers list depends on the pin configured as analog input: some A/D configuration are not allowed if the pins have not been configured correctly. If no pin has been configured as analog input, the entire sheet is disabled.*

Fig.A.14- A/D Converter sheet

SCI Sheet

To configure the Serial Communication Interface you have to specify the following characteristics:

- Communication speed expressed in baud rate
- Data frame format
- Interrupt sources

Baud Rate section

Determines the communication protocol speed. The followings are the acceptable values: 600,1200, 2400, 4800, 9600, 38400 baud. Select the desired speed from the drop-down list.

Data Bits section

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 section

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 section

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 section

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:

The communication break code (10 bits at low level) has been recognized.

***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 DeviceStatus 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 the "DeviceStatus() function parameters" paragraph in Appendix A.*

Warning: the SCI peripheral works properly only when the Chip Clock frequency is set to 5, 10 or 20 MHz. If another frequency has been specified, the SCI configuration sheet is disabled and the peripheral cannot be used.

Fig.A.15 - Serial Communication Interface sheet

Peripherals Setting Blocks

The Peripherals Setting Blocks for ST52x430Kx set the following peripherals:

- A/D Converter
- Watchdog
- PWM-Timer 0
- PWM-Timer 1
- PWM-Timer 2
- SCI

In the following you can find the description of each block.

A/D Converter setting block

The A/D setting block is the most complex, because it allows to change run-time the whole configuration of the peripheral. The sections that you can find in the dialog-box allows to perform the following actions:

- Start or Stop the peripheral
- Set or Reset the peripheral
- Change the conversion mode
- Change the channel conversion sequence
- Choose the channel or the sequence of the channels to be converted
- Change the A/D working frequency

To Start/Stop the peripheral, click the corresponding radiobutton at the top of the dialog box.

If you intend to change run-time the A/D converter configuration, uncheck the “Use Default Configuration” check-box and follow the instructions described below:

In the “Conversion” section click:

- “Single” radiobutton if you want to perform a single conversion cycle
- “Continuous” radiobutton if you want the peripheral to work continuously until it is stopped.

Note: *When “Single” mode is selected, after the conversion, you must start again the peripheral, with the peripheral block, if you want another conversion cycle.*

In the “Channel” section click:

- “Single” radiobutton if you want the conversion of the channel specified in the “Channel to be converted” section.
- “Sequence” radiobutton if you want the conversion of the channels sequence from the first one up to the one specified in the “Last Channel to be converted” section.

The next section is called in two ways, according with the previous settings:

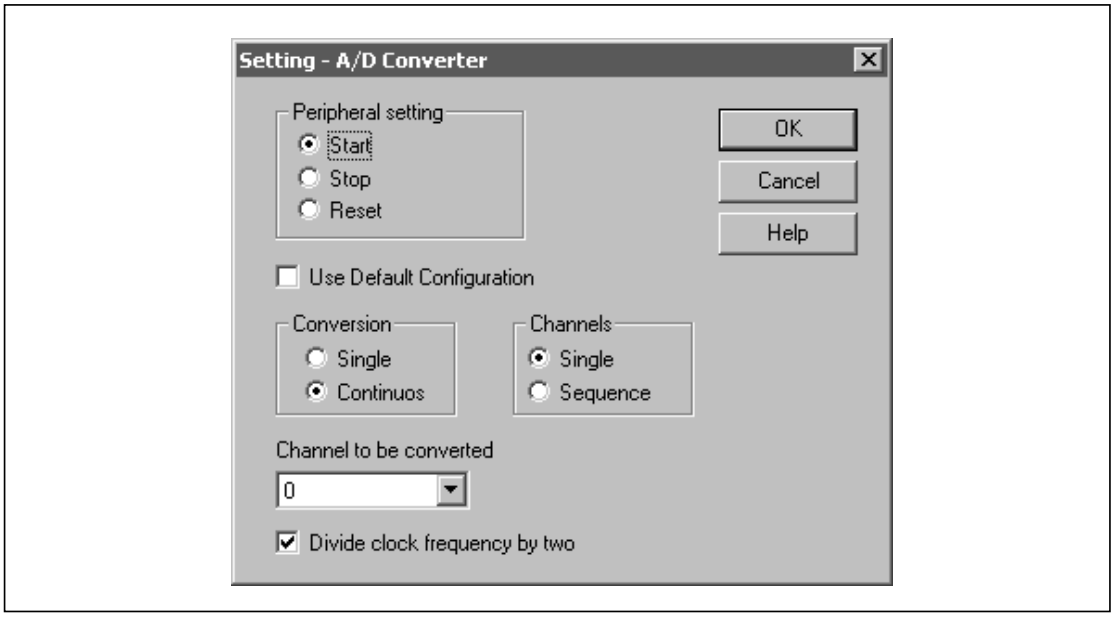
- “Channel to be converted” if the “Single” radiobutton has been selected in the “Channel” section.
- “Last Channel to be converted” if the “Sequence” radiobutton has been selected in the “Channel” section.

In both cases, you have to select the channel number from the drop-down list-box available in this section.

Note: *The contents of the channels' numbers list depends on the pin configured as analog input: some A/D configurations are not allowed if the pins have not been configured correctly. If no pin has been configured as analog input, the entire sheet is disabled.*

The A/D Converter can work both with the chip clock frequency or with half of this frequency. Check the apposite check-box if you want to slower to the half the A/D frequency and conversion time.

Fig. A.16 - A/D Converter Setting Block



Watchdog Setting block

The Watchdog setting block is used to Start/Refresh the Watchdog counter or to disable the peripheral.

- Click the “Start/Refresh” radiobutton if you want to start the peripheral of refresh the counter to avoid the device reset.
- Click the “Disable” radiobutton to disable the peripheral.

Fig. A.17 - Watchdog Setting block

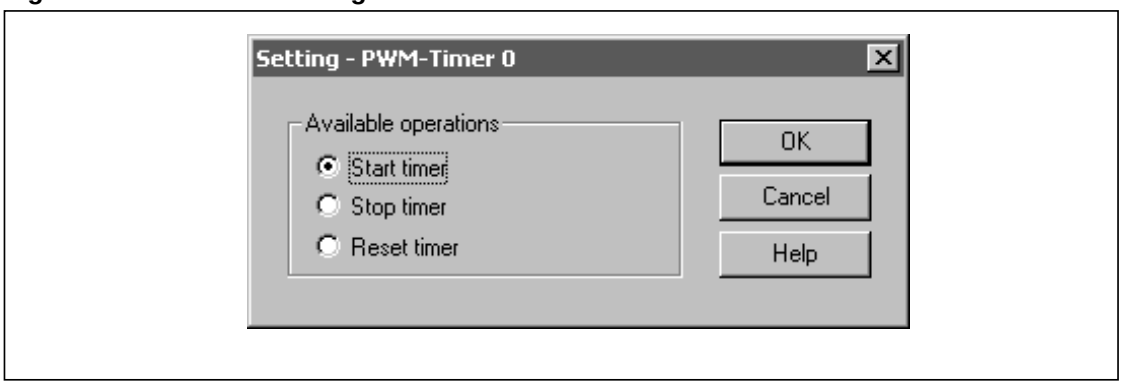
PWM-Timer 0, 1 & 2 setting blocks

All the PWM-Timer setting blocks are the same; each of them is used to Start/Stop or Set/Reset the corresponding peripheral.

- To Start/Stop or Reset the peripheral, click the corresponding radiobutton.

Note: *The available radiobuttons are the only allowed to the peripheral settings. For instance, if the Set/Reset and Start/Stop signal of the PWM/Timer0 are set bothe as external, none of the radiobuttons are made available. Moreover, when the PWM mode is set for the peripheral, the only available radiobuttons are the Start and Reset radiobuttons.*

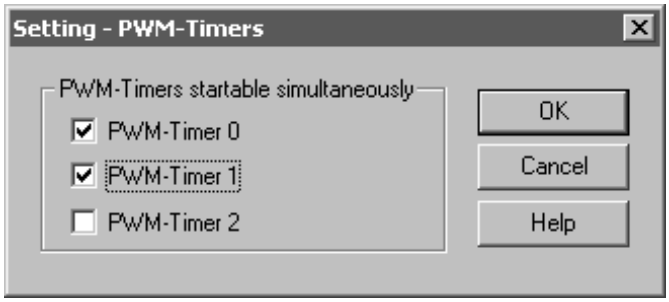
Fig. A.18 - PWM-Timer Settign Block



In addition, with the PWM-Timers setting block you can start two or all the PWM-Timers at the same time by checking the corresponding check-box.

Note: *If the PWM/Timer0 is set with the Start/Stop or Set/reset signals configured as external, the corresponding check-box is disabled.*

Fig. A.19 - PWM-Timers Setting Block



SCI Setting Block

The SCI setting block is used to Start/Stop both the serial trasmission (Tx) and/or reception (Rx).

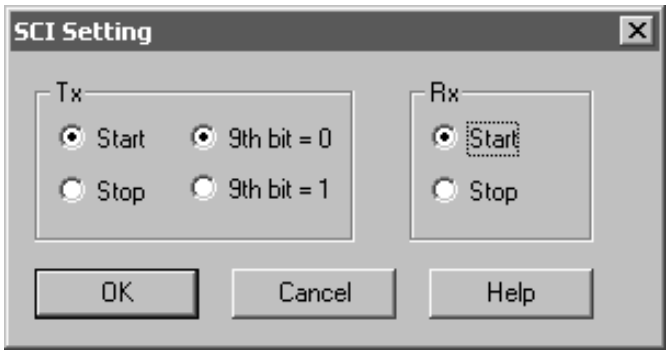
To Start or Stop the serial trasmission click the corresponding radiobutton in the Tx section.

To Start or Stop the serial reception click the corresponding radiobutton in the Rx section.

Note: the radiobuttons are disabled if the corresponding pins Rx and Tx are not configured in these Alternate Functions

Moreover, if the data frame has been configured as 9 bit data, you can us this setting block to specify the value of the 9th bit (0 or 1). Check the corresponding radiobutton in the Tx section. The radiobuttons are disabled if the peripheral has not been configured with 9 bit data.

Fig. A.20 - SCI Setting Block



B - PROGRAMMER BOARD

- Programs EPROM and OTP versions
- PC driven
- Fast parallel code transfer
- Read/write capability
- Memory blank-check
- Piracy chip protection

General Description

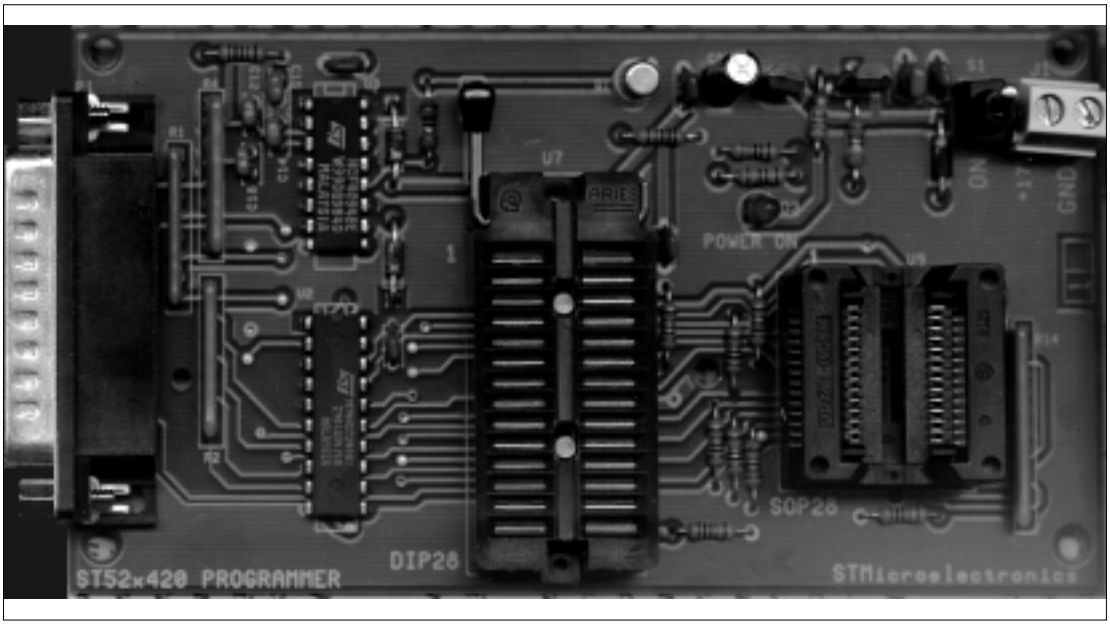
The Programmer board is designed to allow a fast programming of EPROM and OTP version of the *DualLogic* ™ microcontroller.

The Programmer board is managed by means of a dedicated printer driver under control of FUZZYSTUDIO™4 software.

The Programmer Kit is composed by:

- 1 Programming Board
- 1 Communication 25 wire Flat Cable
- FUZZYSTUDIO™4 User Manual
- 3 Floppy disks

Fig. 1 - ST52x420 Programmer Board



Software Installation

The Programmer board is directly managed by FUZZYSTUDIO™4 software then no specific driver installation is required. Refer to FUZZYSTUDIO™4 installation software note to install the whole environment.

Hardware Installation

Because of the Read/Write programmer capabilities, your PC must be equipped with a **bi-directional** parallel port. If your PC-motherboard is an old version, check if this port configuration is available in the BIOS setup.

To install the Programmer board, follow these steps:

1. Power off PC

2. Connect the 25 wires cable connector to the LPT1 parallel port

3. Power on PC

4. Run the BIOS setup and check parallel port configuration (port address and direction mode). Standard configuration should be: port address=378Hex or 278Hex and bi-directional mode or ECP/EPP mode.

If a different direction configuration is enabled, please change in BI-DIRECTIONAL mode.

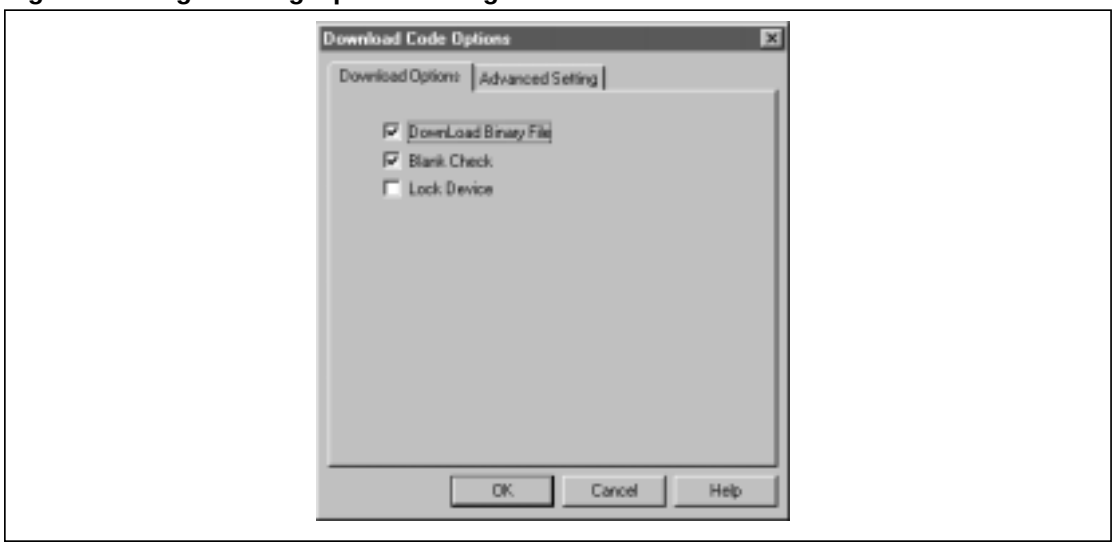
Programming Phase

After the project editing and compilation phase, FUZZYSTUDIO™4 will generate a binary file with the same name of fuzzy project in the current working directory.

This binary code, contained in the “filename.bin” file, is written into the device memory in the downloading phase.

Before writing the binary code in the device, it is suggested to open the option in the “download option” dialog box.

Fig. B. 2 - Programming Options dialog box



Device Programming

After opening the download options, insert the device on the apposite socket, turn the programmer board on and choose the PROGRAMMER > RUN command from the TOOLS menu.

Note: *Take care during the insertion of the device. Insert the device with pin 1 aligned to the printed "1" in the board then turn the Programmer Board on (15Vdc to 18Vdc).*

Hardware Description

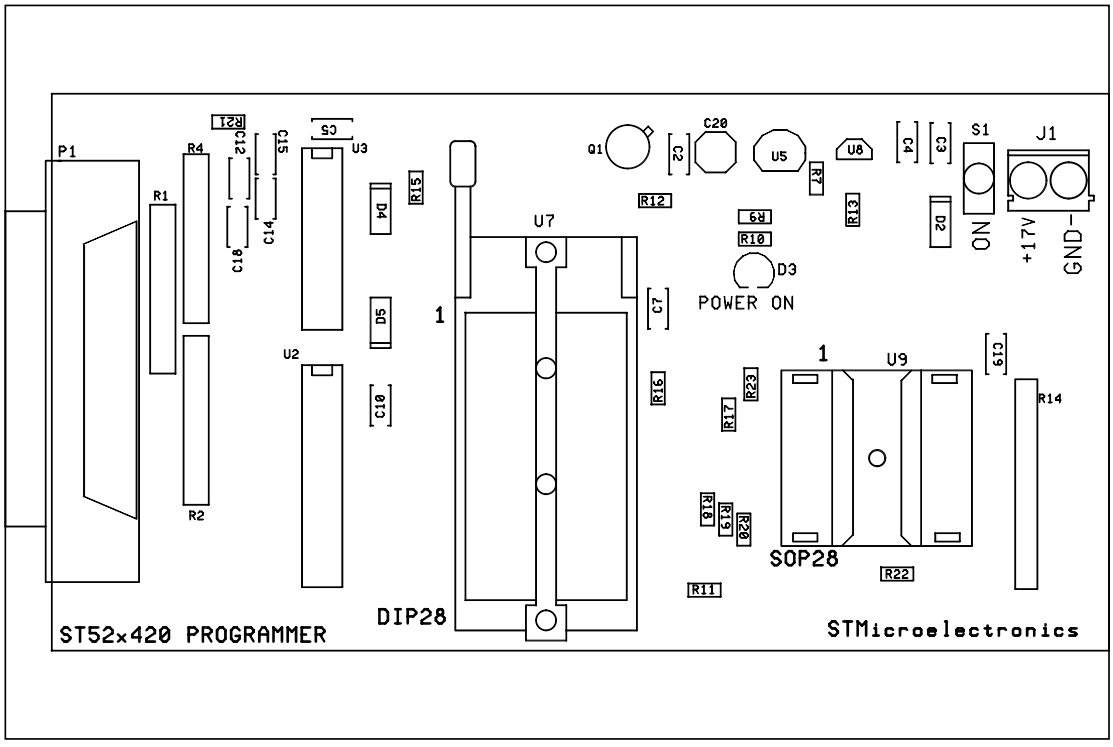
The Programmer board is a simple interface between PC and the device. A software driver allows a direct control of board signals as a particular printer.

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).

Note: *A red LED is used to monitor the power and the short circuits during a possible wrong insertion of device.*

Note: *Although a logic buffer is used to isolate PC signals, it is suggested to turn off the board or disconnect the cable when not used. Other user programs could use the PC parallel port without warnings.*

Fig. 3 - ST52x420 Schematics



C - *FSASM* ASSEMBLER PROGRAMMING TOOL

Introduction

FSAsm is a powerful tool allowing to program the ST52 family products in Assembler.

FSAsm combines the features of a text editor, for the writing and editing of the assembler program, with an easy-to-use machine code generation. The Assembler program can be tested by using the Debugger tool that supplies a graphical environment to insert the result of the chip's simulation. Then, the devices can be quickly programmed by means of an apposite programming board connected to the PC.

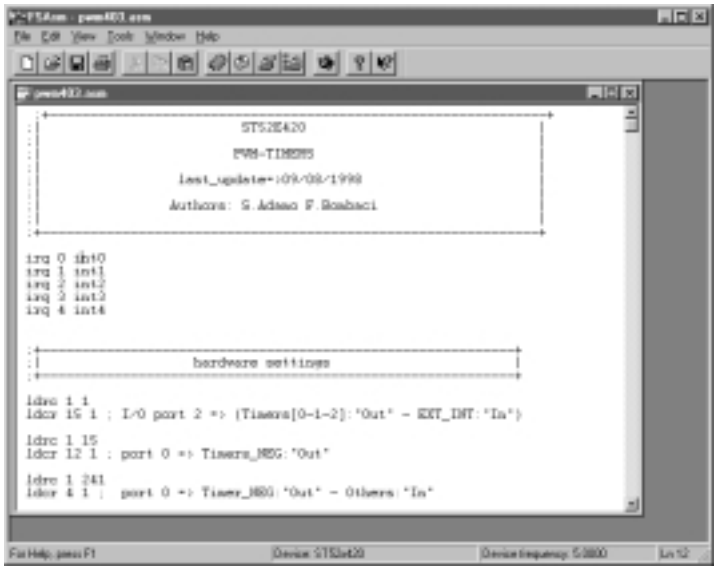
Moreover, it is possible to generate the Assembler listing by creating a new program or by loading a file with the extension .ASM previously generated with FUZZYSTUDIO™4.1 and then modify it with *FSAsm*.

System Requirements

Before you install *FSAsm*, make sure you have all the hardware and software you need to run the program:

- Intel type 80386 processor or higher.
- 8 Mb RAM memory.
- Hard Disk with at least 3 Mbytes of free space.
- VGA or higher graphics card.
- Mouse.
- Windows 95/ 98/ NT operating systems.

Fig. C.1 - FSAsm Main Window



Installing *FSAsm*

The FSAsm Program is installed during the FUZZYSTUDIO™4 installation. Please refer to the FUZZYSTUDIO™4 installation in chapter 1.

FSAsm Main Window

This section provides an overview of the major elements of the *FSAsm* Main Window that you see at a first glance when you start a new project or open an existing one.

FSAsm menus

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

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, font and tab settings commands.

Tools Provides commands for the machine code generation debugging and the device programming by means of the programming board.

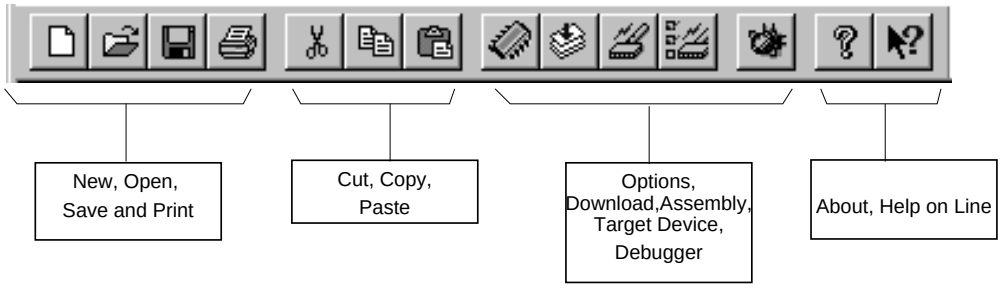
Windows Contains commands related to windows management.

Help Contains help commands.

Note: *When the FSAsm Debugger is active and in foreground the menu item DEBUGGER is added containing the related commands.*

FSAsm toolbar

The *FSASM* toolbar allows 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.

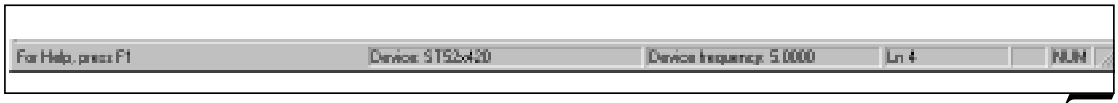


FSAsm status bar

The status bar contains information about the current selected target chip, the selected frequency, the current line number and the position of the cursor.

It is possible to display/hide the status bar by using the apposite command from the View menu.

Fig. C.2 - *FSAsm* status bar



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. By default, the file name is *Untitledx* where *x* is a progressive number according to the already open files "Untitled".

- The file can be saved by means of the command SAVE (CTRL + S) or, with the possibility to modify its name and directory, 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 from *FSAsm* tool (ALT + F4).

Editing Commands

FSAsm provides standard editing commands:

Command	Description
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…(CTRL + F)	Searches a selected text.
REPLACE…(CTRL + H)	Allows to search for and replace text items.

The described commands can be accessed from Edit menu or from the pop-up menu that opens by clicking the right mouse button on the document.

It is also possible to set fonts and tabs by selecting the following commands on the View menu:

- TAB … allows to specify the number of blank characters which form a single tab stop.
- FONT… allows you to set the fonts.

Target Device Selection

To generate the current project's machine code, it is necessary to specify the target device to which the code refers to, if not already specified during previous working sessions.

To specify the target device do the following:

1. Select the command DEVICE from the menu TOOLS or click from the toolbar).
2. In the Target Device dialog box select the device from the scroll-down list.
3. Click OK.

In addition, the text-box to specify the working frequency is supplied. This data item is used only by the Debugger tool to simulate with the correct timing the chip's elaboration. The chosen device is indicated in the status bar; if no device has been selected yet, only the word "Device" will appear.

Fig. C.3 - Target Device dialog box

Note: *The Target Device dialog box automatically appears anytime you choose a command that requires to know the target device, if not yet previously specified .*

Machine Code Generation

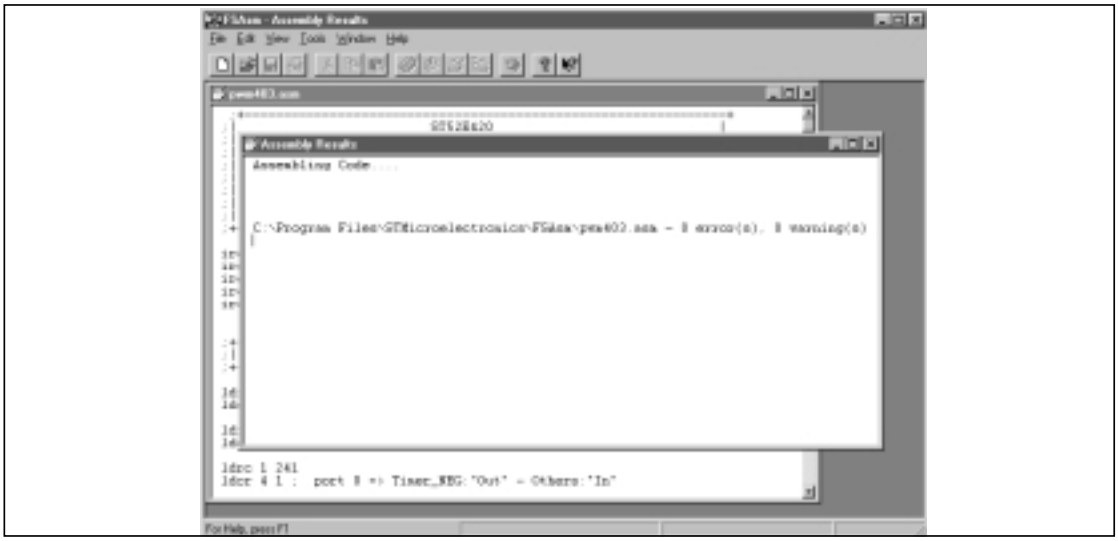
Once the target device has been selected, it is possible to generate the machine code relative to the current program, e.g. the one appearing in the active window if more than one project is open.

To generate the machine code select the ASSEMBLY command from TOOLS menu or click on the apposite toolbar button.

This Assembly command determines the opening of the Output Window :in case of unsuccessful compilation, the list of errors is displayed. Refer to the complete list of errors during the generation of the machine code.

In the Assembly results window, double-clicking on the error message the source code window pops up showing the line where the error occurred.

Fig.C.4 - Assembly Results are shown in the output window



Debugger

FSAsm supplies the Debugger tool that allows to test the developed program by means of the chip's simulation. The Debugger graphical environment allows to choose and visualize the signals to be observed in their time evolution. Then, you can test your program before implementing the application.
The FSAsm Debugger is the same environment of the FS4 Debugger with the same functionalities with the exclusion of the tools and functionalities related to the high-level program definition. In particular the following parts have been excluded:

- FSCode Window
- Block Trace
- Variables Dump

The only high-level feature supplied is the use of the Predefined Variables to address Input and Output Registers. Anyway the Predefined Variables cannot be used in the Assembler program definition.

For all the details on using the Debugger refer to the Chapter 14 in the FUZZYSTUDIO™ 4 User Manual.

Device Programming

After the machine code has been successfully completed, it is possible to program the target device inserted in the apposite board.

Make sure the device has been inserted in the correct way, that the board is turned on and check if it is connected to the computer's parallel port. Moreover, make sure the program has been compiled for the correct device and that the programming board is the suitable one for that device.

It is possible to choose and customize the operations to carry out during the downloading phase, specifying the options using the tab-sheet accessible by means of the Download Options command. Refer to next paragraph for a detailed description of the available options.

To start the programming of the device inserted in the socket select the command DOWNLOAD from the TOOLS menu or click on the apposite toolbar button. The output window opens displaying the evolution of the downloading phases and the eventual error messages.

Device programming status messages

The list of possible messages displayed in the output window is the following one:

Blank Check
The protocol has started to verify the device is not programmed.

Blank Checking Device
The device is being verified that it is not programmed.

Device Lock
The protocol has started the lock of the device memory.

Done
The download has been successfully completed.

Done with Errors
The download has not been successfully completed because an error occurred.

ID Code Read
The protocol has started to read the ID Code.

ID Code Write
The protocol has started to write the ID Code file into the device memory.

Locking Device
The device memory is being locked .

Memory Read
The protocol has started the dumping phase of the device memory.

Memory Write
The protocol has started the writing phase into the device memory.

Reading ID Code
The ID Code is being read in the device memory.

Reading File
The binary file, containing the program to be written into the device memory, is being read.

Reading Memory
The device memory to generate the dump file is being read.

Testing Lock Bit
The device is being verified that it is not locked.

Writing File
The dump file of the device memory is being generated.

Writing ID Code
The ID Code is being written in the memory device.

Writing Memory
The device memory is being programmed.

Device programming error messages

Device is Locked

It has been attempted an operation in a locked device. The only operation allowed on a locked device is the reading of the ID Code.

Device not Blank

The inserted device has not been canceled properly or it is badly placed in the socket.

Error Reading File filename

An error has occurred during the reading of the file containing the binary code to be loaded into the device memory or during the reading of the ID Code. Check if the file exists or if it is corrupted; in the case of the ID Code, check if the name and the path have been specified correctly.

Error Writing File filename

An error has occurred during the reading of the dump file or during the generation of the ID Code. Check if there is enough space on the disk or if the file name has been correctly specified.

Out of Memory

A memory error has occurred. Try closing some open programs.

Unable to Lock Device

The device locking has not been successfully completed. The device could be either already protected or damaged.

Unable to use I/O Ports

You are trying to use the Programmer under Windows NT Operating System or a wrong parallel port address has been specified or the port is out of order.

Write Memory Error

An error has occurred during the programming of the memory device. Check if the device has been correctly inserted or if it is already programmed enabling the Black Check control. Or the device could be damaged.

Wrong Binary File

The binary code file to be loaded into the device memory is corrupted or the file format is not correct. Try to generate again the binary code file.

Programming Options

Before starting the downloading phase of the device, it is possible to specify the operations to carry out during this phase.

To open the Download Code Options Tab-sheet select the item DOWNLOAD OPTIONS from TOOLS menu or click the apposite toolbar button. Choose the “Download Options” sheet to set the most common actions to be performed during the downloading phase; select the sheet “Advanced” to perform more advanced settings.

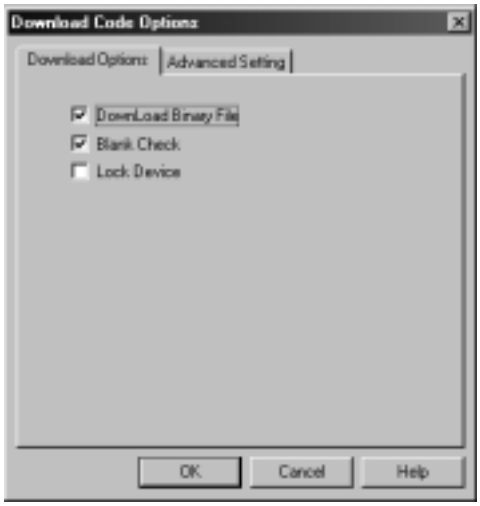
Download Options Settings

In this tab it is possible to enable/disable the following actions:

Download Binary File
Memory device programming.

Blank Check
Verify if the device has been erased.

Fig. C.5 - Download Options



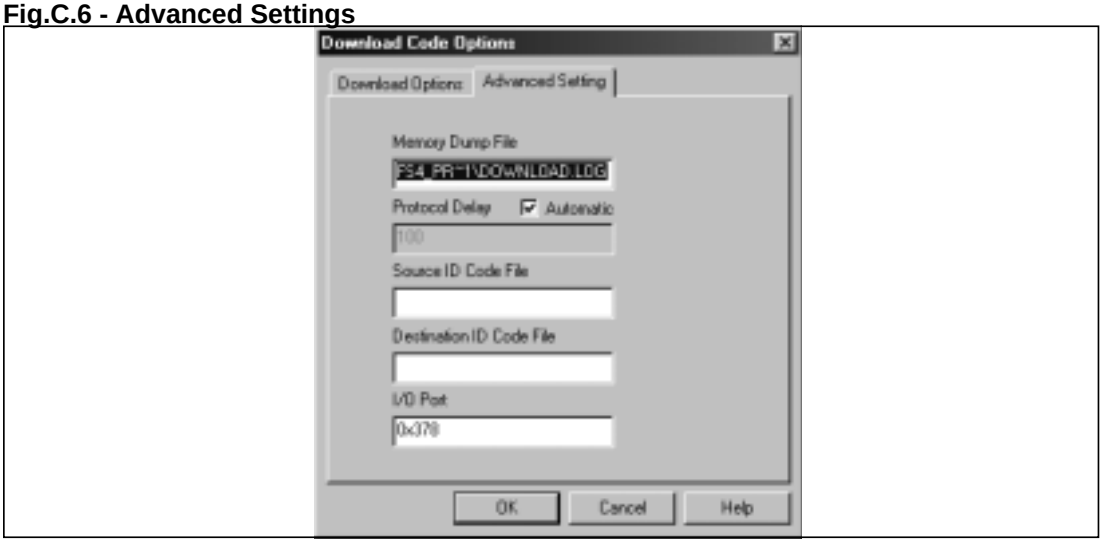
Lock Device
Device read protection

Note : *The device reading protection prevents the reading of the Memory Program content. A device protected in reading can be unprotected only erasing the EPROM memory by exposure to the UV rays. The only operation that is possible to carry out on a reading-protected device is the reading of the ID code.*

Advanced Settings

In this sheet it is possible to specify the advanced options related to the actions started during the programming phase. It is suggested to maintain unchanged the default parameters in case you are not sure about the changes to carry out.

Fig.C.6 - Advanced Settings



Memory Dump File
In this edit-box specify the file name including the file path in which the data read from the program memory during the downloading phase are to be found. The default name is *download.log*; if no name is specified the dump file is not created. The data are shown in hexadecimal format.

Protocol Delay
This parameter (suitable values are between the range 80 400) indicates the speed of data transmission through the parallel port. The default value is 100; specify a greater value if you want to speed up the transmission. The maximum speed you can specify depends from the computer's speed. In case of too high speed the downloading could fail. If you prefer it, it is possible to calibrate automatically the speed factor by checking the Automatic check-box.

Source ID Code File
In this edit-box you can specify the file name including the path containing the ID Code, in text format, to write in the apposite memory space of the device. The data are written during the programming phase of the device but can be written in an already programmed device if that area has not already been programmed. If you do not want to write the memory containing the ID Code do not specify any source file. The total space available is 64 bytes; the file must be formed by 64 characters: further characters will be ignored.

Destination ID Code File:
In this edit-box you can specify the file name including the path containing the ID Code, read from the apposite memory space of the device. In order not to carry out the reading of the ID Code file, do not specify any destination file. The file format is textual and is shown both in numeric and ASCII format.

I/O Port
This edit-box is used to specify the address of the parallel port connected to the board. The default address is 0X378 (LPT1).

Assembler 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.

bad option “xxx” in command line !
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

badly placed or missing “(”
The left parenthesis has been found in a misplaced point or the right parenthesis number is greater than the left one.

badly placed or missing “)”
The right parenthesis has been found in a misplaced point or the left parenthesis number is greater than the right one.

badly placed or missing parenthesis
A parenthesis has been found in a misplaced point or it is missing where necessary.

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 close output file
A file generated during the code generation is corrupted or the disk space is full.

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.

cannot write on binary file
The binary code file is corrupted or the disk space is full.

code will overwrite user data
User data have been placed in memory locations where the program code should be allocated. Change the allocation of the data or check the placement of SETMEM instructions.

data will overwrite previous data
User data have been placed in memory locations where other user data have already been allocated.

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.

insufficient arguments in command line !
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

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

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 one or more file names !
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

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.

new address “address” < current “address”
The address specified with the SETMEM instruction refers to a location before to the current one and causes program code overwriting.

no more memory available
There is not enough memory to run Assembler. Try closing some open programs.

no output format specified !
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

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 optimizing 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 arguments on command line !
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

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.

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.

undefined membership “value”
The specified membership function in fuzzy instruction has not been previously defined.

undefined starting jump to executable code
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

unexpected end of source
The source code ended in a not correct way. Check if the source file is corrupted or for syntax errors.

unrecognized command “name”
The specified command “name” is not a valid command. Check for syntax errors.

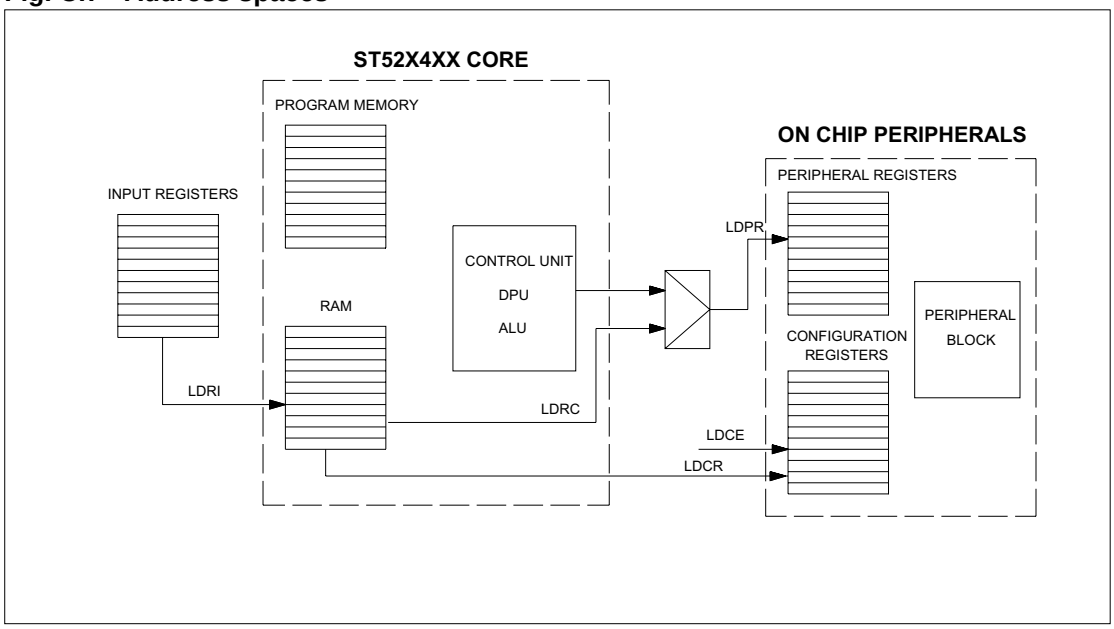
unsupported function “name” for command “command”
Internal error: contact STMicroelectronics - Fuzzy Logic B.U.

wrong membership number “value” (should be “value”)
The value specified as first argument in the MBF instruction is not valid.

Program Memory and Registers' Architecture

To program in Assembler, it is important to consider the architecture of the processor and in particular the address spaces: Program Memory, RAM and registers.

Fig. C.7 - Address spaces



Program Memory

The Program Memory is the EPROM memory where the program is stored. This is made up by three main sections:

- Interrupt Vectors
- Membership Functions (MF) Data Memory
- Program/Data Space

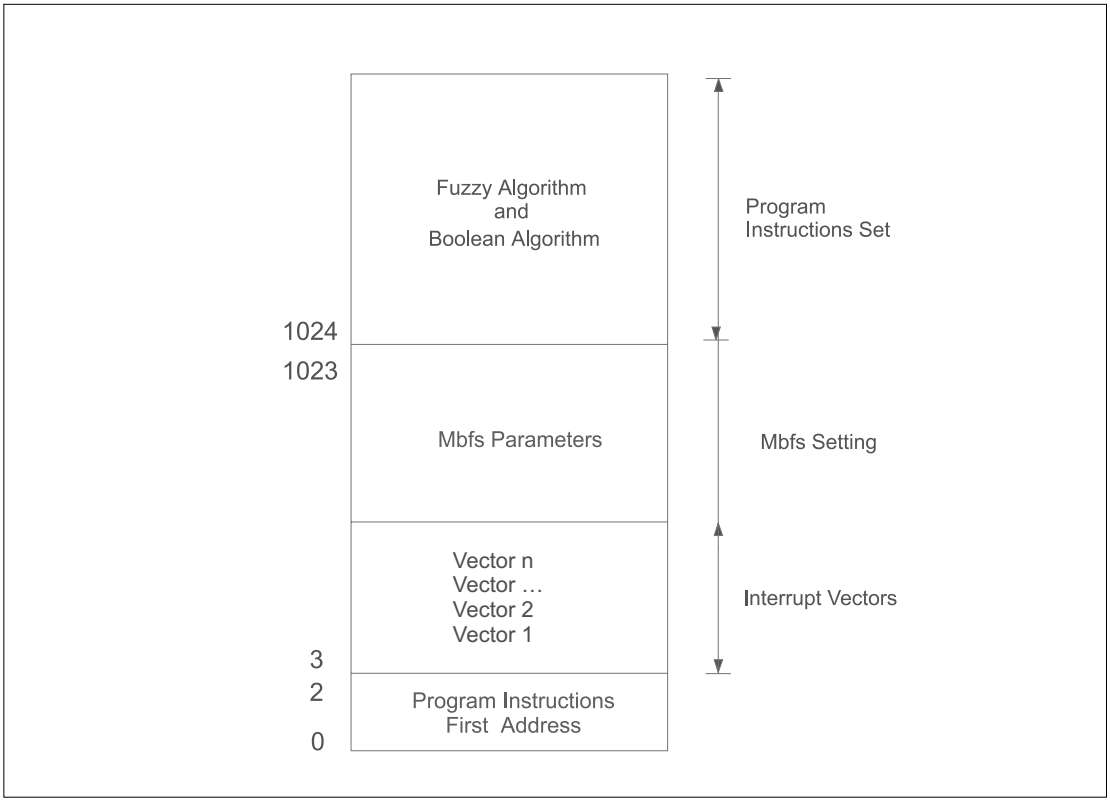
The first three memory locations should contain the jump instruction to the first program instruction address. The Assembler automatically inserts this jump instruction during the code generation.

The memory locations about the Interrupt Vectors are organized in blocks of three bytes where the Interrupt Service routines addresses are contained(the first byte is the jump instruction op code).

After this section, both data type (MF or program data) and program instruction can be inserted. To avoid undesired overlap between data and instructions, it is recommended to write the first program instruction after the data section. The MF data must precede the Program data and/or instructions.

The MF Data describe 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. MF data are stored in the Program Memory by using the instruction MBF. They can be stored up to the address 1023.

Fig. C.8 - Program Memory map



The Program Memory 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 the FUZZY instruction.

WARNING: The fuzzy computation can be interrupted at the end of the computation of a single rule. Be careful when using fuzzy computation in interrupt service routine because it may corrupt an interrupted fuzzy computation section in the main program.

The Program Memory can also contain program data stored with the instruction DATA. These locations can be addressed by the use of the following Assembler instructions:

Instruction	Description
LDCE	Configuration register loading from Program Memory location.
LDPE	Output Register loading from Program Memory location indirect,
LDRE	RAM location loading from Program Memory location.
LDRE	RAM location indirect loading from Program Memory indirect.

These instruction can address Program Memory location by pages because they can address only 256 location. The current Memory page can be set by the PGSET instruction.

RAM Memory

The RAM Memory is composed by locations that can be used either as read and write. These memory locations contain the data which you can perform arithmetic and boolean operations. In addition, starting from the bottom, the RAM space is used as system stack to store the program counter after a CALL instruction or after an interrupt acknowledgment. For this reason, care should be taken using high address location to avoid overlapping with the stack.

Several assembler instructions can address these locations:

Instruction	Description
ADD	Addition
ADDO	Addition with offset
AND	Logic AND
ASL	Shift left
ASR	Shift right
DEC	Decrement
DIV	Division
INC	Increment
LDCR	Configuration register loading with location contents
LDFR	Fuzzy input register loading with location contents
LDPR	Output Register loading with location contents
LDRC	Location loading with a constant
LDRE	Location loading from Program Memory location
LDRE	Location indirect loading from Program Memory location indirect
LDRI	Location loading with the input register contents
LDRR	Location loading with another location contents
MIRROR	Mirroring of location contents
MULT	Multiplication
NOT	Logic NOT
OR	Logic OR
SUB	Subtraction
SUBO	Subtraction with Offset

Configuration Registers

The Configuration Registers file consists of write-only registers. These registers have the task to contain the internal peripherals' configuration of the processor.

It is possible to load these registers through the following instructions:

Instruction	Description
LDCE	Configuration register loading from Program Memory location.
LDCR	Configuration register loading with a RAM location contents.

Constant values should be loaded before in Program Memory or Ram Memory location and then loaded in Configuration register with the above mentioned instructions.

For a detailed description of the Configuration Registers, refer to the device data-sheet.

Input Registers

The Input Registers file consists of read-only registers. 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 location of the RAM Memory.

Output Registers

The Output Registers File consists of write-only registers. Their function is to write particular data to be used by some of the peripherals. They can be addressed only with the following instructions:

Instruction	Description
LDPE	Output Register loading from Program Memory location indirect.
LDPR	Output Register loading with RAM location contents.

Flags

The ST52 family core owns three flag bits with 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 flag bits are the following:

Flag	Description
S	Sign flag: it is set in case of underflow.
C	Carry flag: it is set in case of overflow
Z	Zero flag: it is set when the result of an operation is zero.

See the Assembler Instructions description to know what are the instructions that affects the flag and how they do it.

The flags are taken into consideration by the conditional jump instructions that are the following:

Instruction	Description
JPC	Jumps if carry flag is set.
JPNC	Jumps if carry flag is not set.
JPNS	Jumps if sign flag is not set.
JPNZ	Jumps if zero flag is not set.
JPS	Jumps if sign flag is set.
JPZ	Jumps if zero flag is set.

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™.

The fuzzy programming in Assembler is divided into two phases: the first to define the Mbfs and the second to define the rules.

Membership Functions definition

The assembler instruction to define the Membership Functions is MBF. It indicates to the Compiler which data have to be loaded according to the programmer's specifications.

The MBF instruction syntax is the following one:

MBF mbf_num lvd vtx rvd

where:

mbf_num the order number of the mbf that you are defining.

lvd distance of the left vertex Mbf from the central vertex.

vtx position of the central vertex in the Universe of Discourse.

rvd distance of the right vertex Mbf from the central vertex.

Fig. C. 9 - Triangular Mbf

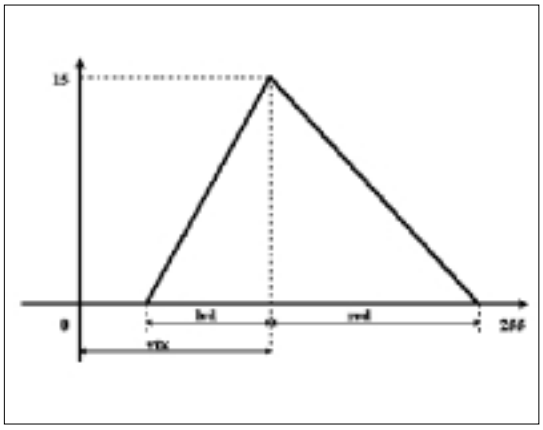
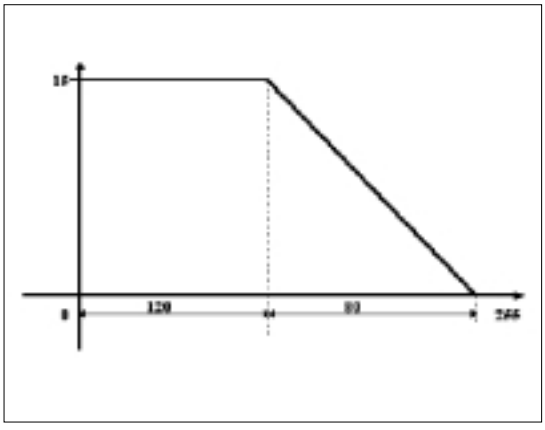


Fig. C. 10 - Trapezoidal Mbf



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 Membership Function showed in the figure on the left is stored as third Membership Function with the instruction: MBF 2 0 120 80 .*

The Membership Functions data should be inserted to the lower addresses of the Program Memory, before program instructions, starting from address 18 and up to the address 1023. MBF data placed over this address cannot be processed. Each MBF instruction stores three bytes of data: so up to 335 Membership Functions can be stored. The Membership Functions are identified with the progressive number *mbf_num* specified in the instruction MBF that is not related to any input variable. So the same Membership Function can be shared by many fuzzy input variable.

Rule Inference

The fuzzy computation unit is made up by:

- Multiplier block for the 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 defuzzyfication.
- Two registers to hold the partial results of the defuzzyfication.
- Divider for the defuzzyfication and the calculation of the output.

The fuzzy instructions affect these block in the following way:

Instruction	Description
CON	activates the multiplier to execute consequent inference
FZAND	executes the AND (minimum) fuzzy operation
FZOR	executes the OR (maximum) fuzzy operation
LDK	loads stack with the result of the last fuzzy operation
LDM	loads stack with the temporary buffer contents.
LDN	executes the fuzzyfication of an input and loads the stack
LDP	executes the fuzzyfication of an input and loads the stack
OUT	activates divider to compute the fuzzy output
SKM	stores last fuzzy operation result on the temporary buffer
LDP	executes the fuzzyfication of an input and loads the stack.

In the ST52 family, each fuzzy rule is consists of up to 8 antecedents and up to as many consequent as the RAM memory locations. Each antecedent term 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.

Fuzzy Variables are initialized, with the crisp values contained in the RAM locations, by means of the LDFR instruction. The arguments of this instruction are the fuzzy destination variable address (0 to 7) and the source RAM location.

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 Instruction Set list for the difference of the two instructions). In particular, the first argument of these instructions indicates the fuzzy variable in question and the second the Membership Function number.

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.

The instruction CON multiplies 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 defuzzyfication of a fuzzy output using the results of all the previous CON instructions.

The Assembler instructions operate as in the following example: let us suppose you have previously defined the Mbf with MBF instructions, then the rule:

```
IF Inp0 is NOT MF01 AND Inp2 is MF21 OR Inp3 is MF33 THEN CRISP1
```

is therefore codified as:

LDN 0 1 Loads in the stack the NOT value relative to the first term of the rule (supposing that *mbf_num* for *MF01* is 1).

LDP 2 21 Loads in the stack the NOT value relative to the second term of the rule(supposing that *mbf_num* for *MF21* is 21) .

FZAND Calculates the min between two values in the stack.

LDK Stores the result of the previous operation in the stack.

LDP 3 33 Loads in the stack the value relative to the third term of the rule (supposing that *mbf_num* for *MF33* is 33).

FZOR Calculates the max between the two values in the stack.

CON 58 Performs the product between the values calculated and the value CRISP1 = 58 (consequent calculus)

Now, let us suppose you have the following rule:

```
IF (Inp2 is MF21 AND Inp3 is NOT MF35) OR (Inp0 is MF03 OR Inp1 is NOT
MF16) THEN CRISP2
```

It is codified with the following instructions:

LDP 2 21 Loads in the stack the value relative to the first term of the rule (supposing that *mbf_num* for *MF21* is 21).

LDN 3 35 Loads in the stack the NOT value relative to the second term of the rule (supposing that *mbf_num* for *MF35* is 35).

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 relative to the third term of the rule (supposing that *mbf_num* for *MF03* is 3).

LDN 1 16 Loads in the stack the NOT value relative to the fourth term of the rule (supposing that *mbf_num* for *MF16* is 16).

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 of each output always start with a FUZZY instruction so, after the first instruction OUT, it is necessary to specify again the FUZZY instruction before starting the instruction for the computation of the next fuzzy output.

The rules that can be 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. As example, the rules having more than one consequent term can be split in as many rules having the same antecedent part and one of each consequent term.

Note: *All rules relative to an output have to be consecutive to calculate the output. Refer to pseudo-instruction list and ST52 data-sheet for further detailed explanations.*

THE STRUCTURE OF A PROGRAM

The ST52 programs in assembler language follows the rigid structure listed below:

- Interrupt Vector Definition
- Membership Function data
- Arithmetic Instructions and/or Program Memory data
- Fuzzy Instructions
- Arithmetic Instructions and/or Program Memory data

The program must end always with an Arithmetic Instruction sections. Couples of arithmetic and fuzzy sections can be repeated to the user discretion. It is not mandatory to insert fuzzy sections, it is necessary to insert at least one arithmetic instructions sections.

The easiest program for a chip of ST52 family consists of the following arithmetic instructions as shown in following:

loop: jp loop

This indicates that a program should end with a loop to another part of the program: the programmer must be 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.

Each section is made up by an assembler code line sequence, with the eventual insertion of blank lines. The fuzzy section for the computation of a single fuzzy output must always start with the instruction **FUZZY**. If other outputs have to be computed, the instruction **FUZZY** must be inserted again for each output.

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 lower case letters for instructions is mandatory because the ST52 Assembler is case sensitive.

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 label 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.

ST52 family Assembler language allows to define line labels only within 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.

Interrupt Vectors Definition

It is a program's section that allows to define the starting address for the interrupt service routine that will be defined in the program. The interrupts service routines 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 directive to define the interrupt vectors is IRQ. The parameters supplied are the interrupt number and the label that indicates the associated service routine starting address. See pseudo-instruction list to further details.

Note The interrupt vectors definition sector must precede all the program instruction and cannot be insert after any other command or Assembler directive.

Program Memory Organization

The ST52 family Program memory is organized by pages of 256 bytes. To access to data and addresses with the Assembler instruction and directives, the user have to specify both the page and the address inside the page.

Actually, instructions can address memory location with 8-bit, so only addresses in the range 0-255 can be managed. The page number is progressively incremented starting from 0 for the first 256 locations up to the last available memory page.

WARNING: The currently set page number is affected and modified after a jump or call instruction or after servicing an interrupt routine. For this reason it is strongly recommended to insert a PGSET instruction each time data in the Program Memory should be accessed. In addition, it is recommended to disable globally the interrupts, with the instruction UDGI, before the PGSET instruction and enable them, with the UEGI instruction, after completing the data access.

To set the current page address the instruction PGSET is used. The only argument of this instruction is an integer value representing the number of the page to be set as current.

Data Management

Data section or look-up table can be inserted in the EPROM memory to be used with the instructions that addresses the location of the Program Memory. The Assembler directive to store data in the Program Memory is **DATA**. The arguments in the order are: the page, the address inside the page (refer to ”Program Memory Organization” paragraph) and the 8-bit data.

Data directives can be inserted everywhere in the program but after the Interrupt Vectors section and the Membership Functions data section.

Current Program Address Management

The first program instruction is allocated automatically in the device memory after the Interrupt Vectors and the Membership Functions data. The first three Program Memory location are programmed automatically with a jump instruction to this address so, after the reset, the Program Counter jumps directly to the first program instruction. After that, the current program address is incremented sequentially as many byte as the current instruction length.

To modify the current memory address, in order to better allocate data and program instructions, it is available the directive **SETMEM**. The argument of this directive are the page and the memory address inside the specified page (refer to the “Program Memory Organization” paragraph).

ASSEMBLER INSTRUCTION SET

ADD

Addition with Offset

Format : **add** dst, src

Operation: dst dst + src

Description: The content of the RAM location specified as source is added to the content of the destination location, leaving the result in the destination.

Flags:

Z	sets if result is zero , cleared otherwise.
C	sets if overflow, cleared otherwise.
S	not affected.

Bytes 3

Cycles 17

Example If the RAM location 20 contains the value 45 and the RAM location 11 contains the value 15, then the instruction:

add 20, 11 0010000 000010100 00001011

causes the location 20 of the RAM to be loaded with the value 60.

If the location 20 contains the value 200 and the location 11 contains the value 100, the instruction causes the location 20 to be loaded with the value 44 (result-256) and the C flag to be set.

ADDO

Addition with Offset

Format : **addo** dst, src

Operation: dst ⇒ dst + src - 128

Description: The content of the RAM location specified as source is added to the content of the destination location, the values 128 is subtracted from the result that is stored in the destination. This operation allows the use of the signed byte considering the values between 0 and 127 as negative, 128 as 0 and the values between 129 and 255 as positive.

Flags:

Z sets if result is zero , cleared otherwise.
C sets if overflow, cleared otherwise.
S sets if underflow, cleared otherwise.

Bytes 3

Cycles 20

Example

If the RAM location 20 contains the value 100 and the RAM location 11 contains the value 40, then the instruction:

```
add   20,   11   00100001   00010100
                 00001011
```

causes the location 20 of the RAM to be loaded with the value 12.

If the location 20 contains the value 100 and the location 11 contains the value 10, the instruction causes the location 20 to be loaded with the value 238 (result + 256) and the S flag to be set. If the location 20 contains the value 200 and the location 11 contains the value 228, the instruction causes the location 20 to be loaded with the value 44 (result - 256) and the C flag to be set.

AND

Logical AND

Format: **and** dst, src

Operation: Dst ⇐ dst AND src

Description: The instruction logically ANDs the content of the RAM locations specified as source and as destination, leaving the result in the destination.

Flags:

Z	sets if result is zero, cleared otherwise.
C	not affected
S	not affected

Bytes: 3

Cycles: 17

Example: If the RAM location 20 contains the value 240 (11110000b) and the RAM location 11contains the value 85 (01010101b), then the instruction

and 20, 11 001000100 0010100 00001011

causes the location 20 of the RAM to be loaded with the value 80 (01010000b).

ASL

Arithmetic Shift Left

Format: **asl** dst

Operation: **C dst (7)**
dst (0) ⇐ 0
dst (n+1) ⇐ dst(n) where n = 0-6

Description: The instruction shifts one bit left the content of the RAM location specified as destination. The most significative bit is placed in the C flag and the less significative bit is loaded with 0.

Flags: Z sets if result is zero, cleared otherwise.
C sets if MSB is set, cleared otherwise.
S not affected.

Bytes: 2
Cycles: 15

Example: if the RAM location 20 contains the value 85 (01010101b), then the instruction:

asl 20 001010010 0010100

causes the location 20 of the RAM to be loaded with the value 170 (10101010b).

If the RAM location 20 contains the value 150 (10010110b), then the instruction causes the location 20 of the RAM to be loaded with the value 44 (00101100b) and the C flag to be set.

ASR

Arithmetic Shift Right

Format: **asr** dst

Operation: **S ⇐ dst (0)**
dst (7) ⇐ 0
dst (n) ⇐ dst (n+1) where n = 0-6

Description: The instruction shifts one bit right the content of the RAM location specified as destination. The less significative bit is placed in the S flag and the most significative bit is loaded with 0.

Flags: Z sets if result is zero, cleared otherwise.
C not affected.
S sets if LSB is set, cleared otherwise.

Bytes: 2
Cycles: 15

Example: If the RAM location 20 contains the value 170 (10101010b), then the instruction:

asr 20 001010100 0010100

causes the location 20 of the RAM to be loaded with the value 85 (01010101b).

If the RAM location 20 contains the value 85 (01010101b), then the instruction causes the location 20 of the RAM to be loaded with the value 42 (00101010b) and the S flag to be set.

CALL

Subroutine Call

Format: **call** label

Operation: **SP ⇐ SP -2** **(SP = Stack Pointer)**
(SP) ⇐PC **(PC = Program Counter)**
PC label

Description: The content of the Program Counter (PC) is pushed to the top of the System Stack and the location address specified by the symbol label is loaded into the PC in order to point to the first instruction of the subroutine.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 17

Example: If the label “subx”, that indicates the first location of a subroutine, is located to the address 2500 (00001001 11000100), then the instruction:

call subx	01000111	00001001	11000100

causes the PC to be loaded with the value 2500 and the program to jump to the subroutine labelled “subx”.

DEC

Decrement

Format: **dec** dst

Operation: **dst ⇐ dst - 1**

Description: The content of the specified RAM location is decremented by 1.

Flags:
Z sets if result is zero, cleared otherwise.
C not affected.
S sets if underflow, cleared otherwise.

Bytes: 2
Cycles: 15

Example: If the RAM location 20 contains the value 50, then the instruction:

dec 20 001011000 0010100

causes the location 20 of the RAM to be loaded with the value 49.

If the RAM location 20 contains the value 0, then the instruction causes the location 20 to be loaded with the value 255 and the S flag to be set.

DIV

Division (16/8)

Format: **dec** dst, src

Operation: **[dst dst+1] / src :**
dst ⇐ remainder
dst + 1 ⇐ result

Description: The content of the destination RAM location pair (the 16 bit dividend is composed by the dst (MSB) and dst+1 (LSB) locations) is divided by the source. The LSB of the destination location pair (dst+1) is loaded with the result, the MSB (dst) is loaded with the remainder. In case of overflow the MSB and the LSB are loaded both with 255.

Flags:
Z sets if result is zero, cleared otherwise.
C sets if overflow, cleared otherwise.
S sets if remainder is zero, cleared otherwise.

Bytes: 3
Cycles: 26

Example: If the RAM location pair 20 and 21 contains the value 1523 and the location 40 contains the value 30, then the instruction:

div 20, 40	00100011	00010100	00101000

causes the location 21 of the RAM to be loaded with the value 50 and the location 20 with the value 23.

FUZZY

Fuzzy Computation

Format: **fuzzy**

Operation: **Start fuzzy output computation**

Description: This instruction transfers the control to the Fuzzy Computation Unit for the evaluation of a single fuzzy output. After this instruction, only fuzzy instructions can be inserted until the instruction OUT is specified. If more fuzzy output have to be computed, the instruction FUZZY should be specified again after the instruction OUT.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 6

Example: The following instruction:

fuzzy 10000000

starts a fuzzy computation section.

HALT

Halt

Format: **halt**

Operation: **Clock Master halted.**

Description: This instruction stops the clock master so that the CPU and the peripherals are turned-off. It is possible to exit from the halt mode by means of an external interrupt or a chip reset.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 - 13

Example: After the instruction:

halt 00110111

the device is put in halt mode and the program is stopped until an external interrupt or a chip resets.

INC

Increment

Format: **inc** dst

Operation: **dst ⇐ dst + 1**

Description: The content of the specified RAM location is incremented by 1.

Flags:
Z sets if result is zero, cleared otherwise.
C sets if overflow, cleared otherwise.
S not affected.

Bytes: 2
Cycles: 15

Example: If the RAM location 20 contains the value 50, then the instruction:

inc 20 00101101 00010100

causes the location 20 of the RAM to be loaded with the value 51.

If the RAM location 20 contains the value 255, then the instruction causes the location 20 to be loaded with the value 0 and the S and Z flags to be set.

JP

Unconditional Jump

Format: **jp** label

Operation: **PC ⇐ label** **(PC = Program Counter)**

Description: This instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, then the instruction:

jp labelx 01000000 00001001 11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPC

Jump if C Flag Set

Format: **jpc** label

Operation: **if C=1, PC ⇐ label (PC = Program Counter)**

Description: If C flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the C flag is set then the instruction:

jpc labelx 01000101 00001001 11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNC

Jump if C Flag Not Set

Format: **jpnc** label

Operation: **if C=0, PC label (PC = Program Counter)**

Description: If C flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the C flag is not set then the instruction:

jpnc labelx	01000110	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNS

Jump if S Flag Not Set

Format: **jpns** label

Operation: **if S=0, PC label (PC = Program Counter)**

Description: If S flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the S flag is not set then the instruction:

jpns labelx 01000010 00001001 11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNZ

Jump if Z Flag Not Set

Format: **jpnz** label

Operation: **if Z=0, PC ⇐ label** **(PC = Program Counter)**

Description: If Z flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the Z flag is not set then the instruction:

jpnz labelx	01000100	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPS

Jump if S Flag Set

Format: **jps** label

Operation: **if S=1, PC ⇐ label** **(PC = Program Counter)**

Description: If S flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3

Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (000 01001b 11000100b) is labeled with “labelx”, and the S flag is set then the instruction:

jps labelx	01000001	00001001	1000100

loads the value 2500 into the PC and transfers the program control to that location.

JPZ

Jump if Z Flag Set

Format: **jpz** label

Operation: **if Z=1, PC ⇐ label** **(PC = Program Counter)**

Description: If Z flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 11 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the Z flag is set then the instruction:

jpz labelx 01000011 00001001 11000100

loads the value 2500 into the PC and transfers the program control to that location.

LDCE

Load Configuration, EPROM

Format: **ldce** dst, src

Operation: **dst ⇐ src**

Description: The instruction loads into the configuration register specified as destination the data contained in the Program Memory source location in the current page, specified with the PGSET instruction.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: if the Program Memory location 300 contains the value 240 and the current page is set to 1 (256+44=300), then the instruction:

ldce 12, 44 00011011 00001100 00101100

causes the configuration register 12 to be loaded with the value 240.

LDCR

Load Configuration, RAM

Format **ldcr dst, src**

Operation: **dst ⇐ src**

Description: The instruction loads into the configuration register specified as destination the data contained in the RAM location specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: If the RAM location 80 contains the value 64 then the instruction

ldcr 12, 80 00010100 00001100 01010000

causes the configuration register 12 to be loaded with the value 64.

LDFR

Load Fuzzy, RAM

Format: **ldfr** dst, src

Operation: **dst ⇐ src**

Description: The instruction loads into Fuzzy input registers (0 to 7) specified as destination the data contained in the RAM location specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: If the RAM location 80 contains the value 64 then the instruction

ldfr 2, 80 00011000 00000010 01010000

causes the fuzzy input register 2 to be loaded with the value 64, that is used as crisp input value of the third fuzzy variable.

LDPE

Load Peripheral, EPROM Indirect

Format: **ldpe** dst, (src)

Operation: **dst ⇐ (src)**

Description: The instruction loads into the Output Peripheral Register specified as destination the data contained in the EPROM location which address (in the page set with the PGSET instruction) is contained in the location specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 17

Example: If the currently EPROM page set is 2, the RAM location 30 contains the value 10 and the EPROM location 522 (256*2+10) contains the value 100, then the instruction:

ldpe 2, (30) 00010110 00000010 00011110

causes the Output Peripheral Register 2 to be loaded with the value 100.

LDPR

Load Peripheral, RAM

Format: **ldpr** dst, src

Operation: **dst ⇐ src**

Description: The instruction loads into the Output Peripheral Register specified as destination the data contained in the RAM location specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: If the RAM location 30 contains the value 100, then the instruction:

ldpr 2, 30 00010101 00000010 00011110

causes the Output Peripheral Register 2 to be loaded with the value 100.

LDRC

Load RAM, Constant

Format: **ldrc** dst, const

Operation: **dst ⇐ const**

Description: The instruction loads into the RAM location specified as destination the constant specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: The following instruction:

ldrc 24, 130 00010000 00011000 10000010

causes the RAM location 24 to be loaded with the value 130.

LDRE

Load RAM, EPROM

Format: **ldre** dst, src

Operation: **dst ⇐ src**

Description: The instruction loads into the RAM location specified as destination the contents of the EPROM location specified as source (in the page set with the PGSET instruction).

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 16

Example: If the currently set EPROM page is 2 and the address 522 (256*2+10) contains the value 100, then the following instruction:

ldre 24, 10 00010001 00011000 00001010

causes the RAM location 24 to be loaded with the value 100.

(LDRE)

Load RAM Indirect, EPROM Indirect

Format: **ldre** (dst), (src)

Operation: **(dst) ⇐ (src)**

Description: The instruction loads into the RAM location, which address is contained in the RAM location specified as destination, the contents of the EPROM location, which address is contained in the RAM location specified as source (in the page set with the PGSET instruction).

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 18

Example: If the currently set EPROM page is 2, the RAM location 20 contains the value 10, the address 522 (256*2+10) contains the value 100 and the RAM location 24 contains the value 50, then the following instruction:

ldre (24), (10)	00010010	00011000	00001010

causes the RAM location 50 to be loaded with the value 100.

LDRI

Load RAM, Peripheral Input

Format: **ldri** dst, src

Operation: **dst ⇐ src**

Description: The instruction loads into the RAM location specified as destination the contents of the Input Peripheral Register specified as source.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 14

Example: If the Input Peripheral Register 10 contains the value 100, then the following instruction:

ldri 24, 10 0 0010011 00011000 00001010

causes the RAM location 24 to be loaded with the value 100.

LDRR

Load RAM, RAM

Format: **ldrr** dst, src

Operation: **dst ⇐src**

Description: The instruction loads into the RAM location specified as destination the contents of RAM location specified as source.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 3
Cycles: 16

Example: if the RAM location 10 contains the value 100, then the following instruction:

ldrr 24, 10	00010111	00011000	00001010

causes the RAM location 24 to be loaded with the value 100.

MDGI

Macro Disable Global Interrupts

Format: **mdgi**

Operation: **all interrupts disabled**

Description: This instruction is used by the FUZZYSTUDIO Compiler in order to disable the interrupts at the beginning of a Compiler Macro.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 if GI already disabled, 16 otherwise

Example: After the instruction:

mdgi 00110100

interrupts cannot be serviced until the Global Interrupt Mask (GI) is again enabled with a MEGI instruction.

MEGI

Macro Enable Global Interrupts

Format: **megi**

Operation: **not masked interrupts enabled**

Description: This instruction is used by the FUZZYSTUDIO Compiler in order to enable not masked interrupts after the end of a Compiler Macro. Interrupts cannot be enabled if a UDGI instruction, not followed by a UEGI instruction, has been specified.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 if GI already enabled, 16 otherwise

Example: If a UDGI instruction, not followed by a UEGI instruction, has not been specified, after the instruction:

megi 00110101

not masked interrupts are enabled.

MIRROR

Byte Mirror

Format: **mirror dst**

Operation: **dst(n) ⇐dst(7-n)**

Description: This instruction modifies the content of the specified RAM location, inverting the order of the bits.

Flags:
Z set if result is zero, cleared otherwise.
C not affected.
S not affected.

Bytes: 2
Cycles: 15

Example: If the RAM location 24 contains the value 142 (10001110b), after the instruction:

mirror 24 00101011 00011000

the RAM locations will contain the value 113 (01110001b).

MULT

Multiplication (8 X 8)

Format: **mult dst, src**

Operation: **[dst dst+1] ⇐ dst * src**

Description: The instruction computes the product between the values contained in the RAM locations specified as destination and as source. The result is a 16 bit number which the most significative byte is stored in the destination location and the least significative is stored in the location after the destination.

Flags:
Z set if result is zero, cleared otherwise.
C not affected.
S not affected.

Bytes: 3
Cycles: 19

Example: If the RAM location 20 contains the value 100 and the location 40 contains the value 30, then the instruction:

mult 20, 40 00100100 00010100 00101000

causes the location 20 of the RAM to be loaded with the value 11 (MSB) and the location 21 with the value 184 (256*11+184=30*100=3000).

NOP

No Operation

Format: **nop**

Operation: **No operation.**

Description: No operation is carried out with this instruction. It is typically used for timing delay.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 6

Example: The instruction:

nop 10000001

causes the program control to pass to the next instruction after 6 clock cycles.

NOT

Logical NOT

Format: **not** dst

Operation: **dst ⇐ 255-dst**

Description: This instruction negates each bit of the location specified as destination.

Flags:
Z sets if result is zero, cleared otherwise.
C not affected.
S not affected.

Bytes: 2
Cycles: 15

Example: If the location 24 contains the value 100 (01100100b), the instruction:

not 24 00100101 00011000

causes the location 24 to be loaded with the value 155 (10011011b).

OR

Logical OR

Format: **or** dst, src

Operation: **dst ⇐ dst OR src**

Description: The instruction logically ORs the content of the RAM locations specified as source and as destination, leaving the result in the destination.

Flags:
Z sets if result is zero, cleared otherwise.
C not affected.
S not affected.

Bytes: 3
Cycles: 17

Example: If the location 24 contains the value 100 (01100100b), and the location 10 contains the value 15 (00001111b), then the instruction:

or 24, 10	00100110	00011000	00001010

causes the location 24 to be loaded with the value 111 (01101111b).

PGSET

Page Set

Format: **pgset** const

Operation: **Page pointer setting.**

Description: This instruction sets the current EPROM page to the const page, so that the locations that can be addressed are in the range [256*const , 256*cost+255]

WARNING: the page pointer is modified by the jump instructions (JP, CALL, JPC, etc.) so the PGSET instruction should be specified again before accessing an EPROM location if a jump instruction have been used after the last PGSET.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 9

Example: The instruction:

pgset 4	00011001	00000100

sets the current page to the fifth page (addresses 1024-1279).

RET

Return from Subroutine

Format: ret

Operation: **PC ⇐(SP)** (PC = Program Counter)
SP ⇐ SP+ 2 (SP = Stack Pointer)

Description: This instruction performs the return from a subroutine. It determines the jump of the program to the line after the subroutine call instruction.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 12

Example: If the value to the top of the stack is 0e4h, the instruction:

ret 01001000

determines the PC to be loaded with the value 0e4h and the previous value to be lost.

RETI

Return from Interrupt

Format: **reti**

Operation: **PC ⇐ (SP) (PC = Program Counter)**
SP ⇐ SP + 2 (SP = Stack Pointer)
flag ⇐ saved flags

Description: This instruction performs the return from a interrupt service routine. It determines the return of the device to the state it was before the interrupt. The value of the PC is popped from the top of the stack, together with the saved flags.

Flags: Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 12

Example: If the value to the top of the stack is 0e4h, the instruction:

reti 00110000

determinates the PC to be loaded with the value 0e4h, the previous value to be lost and the flags status before the interrupt to be restored.

RINT

Reset Interrupt

Format: **rint** const

Operation: **Interrupt No. const Pending bit ⇒0**

Description: This instruction resets the pending bit of the interrupt No.const. After this instruction the request of interrupt is cancelled and will not be acknowledged

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 8

Example: If the interrupt 3 source has generated an interrupt request remaining pending (being the interrupt masked or globally disabled), after the instruction

rint 3 00110001 00000011

the interrupt request is cancelled and will be serviced when enabled only if a successive request is sent.

SUB

Subtraction

Format: **sub** dst, src

Operation: **Dst ⇐ dst - src**

Description: The content of the RAM location specified as source is subtracted to the contents of destination location, leaving the result in the destination.

Flags: Z sets if result is zero, cleared otherwise.
C not affected.
S sets if underflow, cleared otherwise.

Bytes: 3
Cycles: 17

Example: if the RAM location 20 contains the value 45 and the RAM location 11 contains the value 15, then the instruction

sub 20, 11 00100111 00010100 00001011

causes the location 20 of the RAM to be loaded with the value 30.

If the location 20 contains the value 80 and the location 11 contains the value 100, the instruction causes the location 20 to be loaded with the value 236 (256 + result) and the S flag to be set.

SUBO

Subtraction with Offset

Format: **subo** dst, src

Operation: **Dst ⇐ dst + 128 - src**

Description: The value 128 is added to the content of the RAM location specified as destination, then the content of source location is subtracted to the result and stored into the destination location. This operation allows the use of the signed byte considering the values between 0 and 127 as negative, 128 as 0, and the values between 129 and 255 as positive.

Flags:
Z sets if result is zero, cleared otherwise.
C sets if overflow, cleared otherwise.
S sets if underflow, cleared otherwise.

Bytes: 3
Cycles: 20

Example: if the RAM location 20 contains the value 45 and the RAM location 11 contains the value 65, then the instruction

subo 20, 11 00101000 00010100 00001011

causes the location 20 of the RAM to be loaded with the value 108.

If the location 20 contains the value 200 and the location 11 contains the value 20, the instruction causes the location 20 to be loaded with the value 52 (result-256) and the C flag to be set.If the location 20 contains the value 20 and the location 11 contains the value 200, the instruction causes the location 20 to be loaded with the value 204 (256+result) and the S flag to be set.

UDGI

User Disable Global Interrupts

Format: **udgi**

Operation: **all interrupts disabled**

Description: This instruction can be used by the User in order to disable globally the interrupts.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 if GI already disabled, 16 otherwise

Example: After the instruction:

udgi 00110010

interrupts cannot be serviced until the Global Interrupt Mask (GI) is again enabled with a UEGI instruction.

UEGI

User Enable Global Interrupts

Format: **uegi**

Operation: **not masked interrupts enabled**

Description: This instruction can be used by the Compiler in order to enable not masked interrupts. Interrupts cannot be enabled if a MDGI instruction, not followed by a MEGI instruction, has been specified.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 if GI already enabled, 16 otherwise

Example: If a MDGI instruction, not followed by a MEGI instruction, has not been specified, after the instruction:

uegi 00110011

not masked interrupts are enabled.

WAITI

Wait for Interrupt

Format: **wait**

Operation: **wait for interrupt**

Description: This instruction stops the program execution until an interrupt from an active source is requested. During the wait state some functionalities of the device are turned off in order to lower the power consumption.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7 if GI already enabled, 16 otherwise

Example: The instruction:

waiti 00110110

puts the chip in wait mode and stops the program execution, waiting for an interrupt signal. If there are no active interrupt sources, the device can exit from the wait mode only with a reset.

WDTRFR

Watchdog Refresh

Format: **wdtrfr**

Operation: **Watchdog counter enabled or refreshed**

Description: If the Watchdog is disabled, this instruction enables the watchdog and the counter starts to count from the configured value. If the watchdog is already enabled, this instruction restarts the counting from the beginning.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 7

Example: After the instruction:

wtdrfr 10000010

the Watchdog is enabled and the value of counting stored in the Configuration Register 2 is loaded in the Watchdog counter.

WDTSLP

Watchdog Sleep

Format: **wdtslp**

Operation: **Watchdog disabled**

Description: This instruction disables the Watchdog, avoiding the chip reset.

Flags:
Z not affected.
C not affected.
S not affected.

Bytes: 1
Cycles: 6

Example: After the instruction:

wtdslp 10000011

the Watchdog is disabled stopping the counter .

ST52 Assembler Pseudo Instructions:

The Assembler pseudo instructions have not direct correspondence with the machine code; this is obtained after the elaboration of the supplied data by means of the Assembler.

The Assembler pseudo instructions are used to set the data for the Fuzzy Computation, the Assembler then optimizes these data considering the code format used from the Fuzzy Computation Unit.

There are also the pseudo instructions to set data and to set the current location in EPROM Memory.

CON

Consequent

Format: **con** const

Operation: **Dividend Register ⇐Dividend register + Teta * const**
Divisor Register ⇐Divisor Register + Teta

Description: This instruction computes the values to add in the defuzzyfication registers, at the end of the single rule. The specified constant is the crisp value representing the output crisp membership function: it is multiplied by the last fuzzy operation result.

DATA

EPROM Data

Format: **data** page, addr, value

Operation: **none**

Description: This pseudo instruction indicates to the Assembler to store data in the EPROM. The location in the address of the specified page is loaded with the specified value.

IRQ

Interrupt Request Vector

Format: **irq** int, label

Operation: **none**

Description: This pseudo-instruction indicates the interrupt vectors to the Assembler.
The argument represents respectively the interrupt and the relative interrupt service routine first address, pointed with a label.

FZAND

Fuzzy AND

Format: **fzand**

Operation: **K ⇐MIN(stack(0) , stack(1))**

Description: This instruction computes the Fuzzy AND operation (minimum) between the two values stored in the Fuzzy stack, previously loaded with LDP, LDN or LDK instructions, and stores the result in the register K.

FZOR

Fuzzy OR

Format: **fzor**

Operation: **K ⇐MAX(stack(0) , stack(1))**

Description: This instruction computes the Fuzzy OR operation (maximum) between the two values stored in the Fuzzy stack, previously loaded with LDP, LDN or LDK instructions, and stores the result in the register K.

LDK

Load Stack with K Register

Format: **ldk**

Operation: **stack(0) ⇐ K**

Description: The instruction loads in the Fuzzy stack the value temporarily stored in the Fuzzy register K that is the result of the last Fuzzy operation.

LDM

Load Stack with M Register

Format: **ldm**

Operation: **stack(0) ⇐M**

Description: The instruction loads in the Fuzzy stack the value temporarily stored in the Fuzzy register M with a previous SKM operation.

LDN

Load Negative Alpha Value

Format: **ldn** var, mbf

Operation: **stack ⇐ 15 - computed alpha value related to mbf M.F. of var Variable**

Description: The instruction performs the fuzzyfication and loads in the stack the negated alpha value of the mbf M.F. of the var Variable.

LDP

Load Positive Alpha Value

Format: **ldp** var, mbf

Operation: **stack ⇐ computed alpha value related to mbf M.F. of var Variable**

Description: The instruction performs the fuzzyfication and loads in the stack the alpha value of the mbf M.F. of the var Variable.

MBF

Membership Function

Format: **mbf** num, lvd, vtx, rvd

Operation: **none**

Description: This pseudo instruction indicates to the Assembler to store a Membership Function data in the EPROM Memory. The M.F. number is specified as first argument, followed by the left semibase width, the vertex position and the right semibase width. The first (of three) EPROM location where the data are stored is the current program line.

OUT

Fuzzy Output

Format: **out** dst

Operation: **dst ⇐current fuzzy output defuzzyfication result.**

Description: This instruction performs the defuzzyfication for the computation of the current fuzzy output and store the result in the destination RAM location.

SETMEM

Set Memory

Format: **setmem** page, addr

Operation: **none**

Description: This pseudo-instruction indicates that the next current program line must be the one in the specified address of the specified page.

SKM

Store K Register in M Register

Format: **skm**

Operation: **M ⇐K**

Description: This instruction loads the result of the last performed Fuzzy operation (stored in the temporary register K) in the temporary buffer M.

Appendix D - 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 I smedium, the crossing is not far
with the **Reference Term**	Red, Yellow, Green; Very Low, Low, Medium, High; Very far away, far away, far away, close, very close
we obtain 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 Functions

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 .

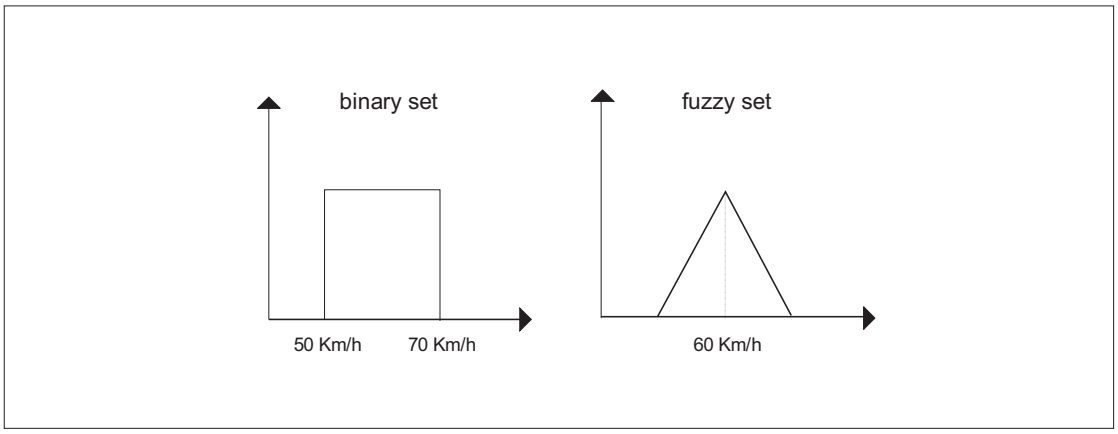


Fig.1

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*.

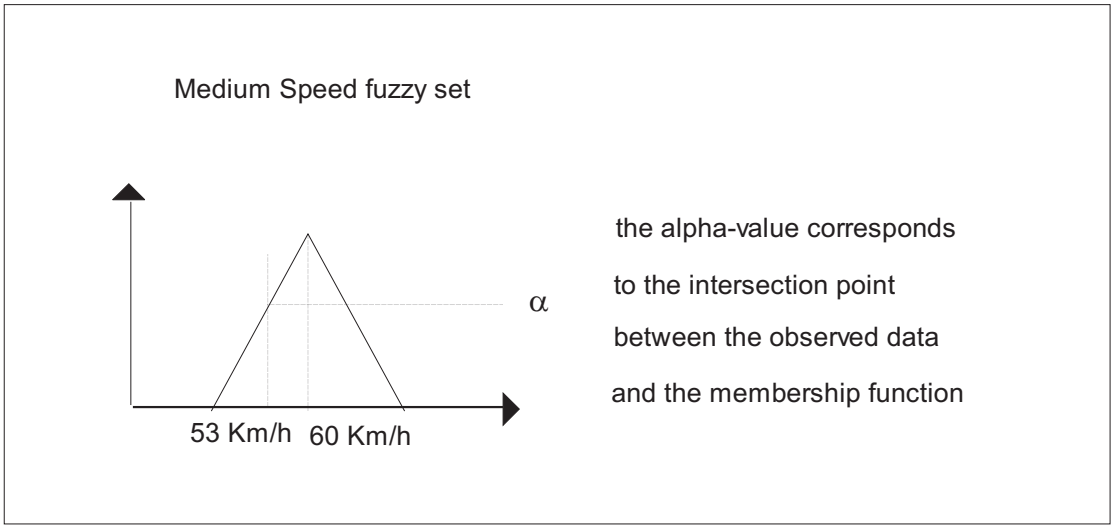


Fig. 2

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.

Fuzzyfication: 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.

Defuzzyfication: 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.

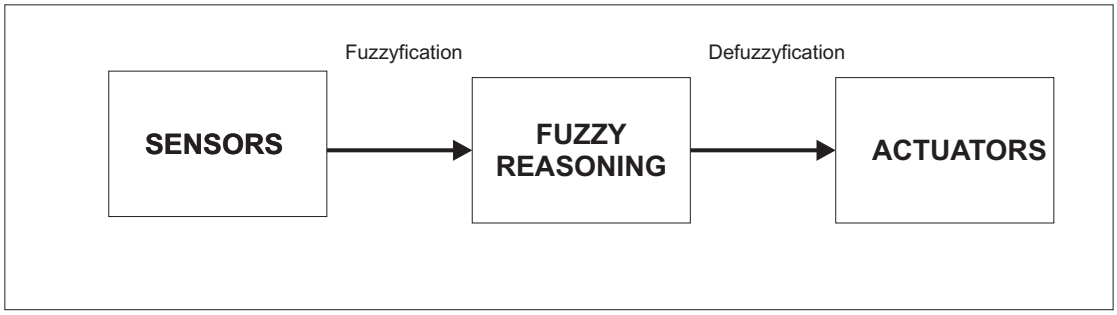


Fig. 3

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.

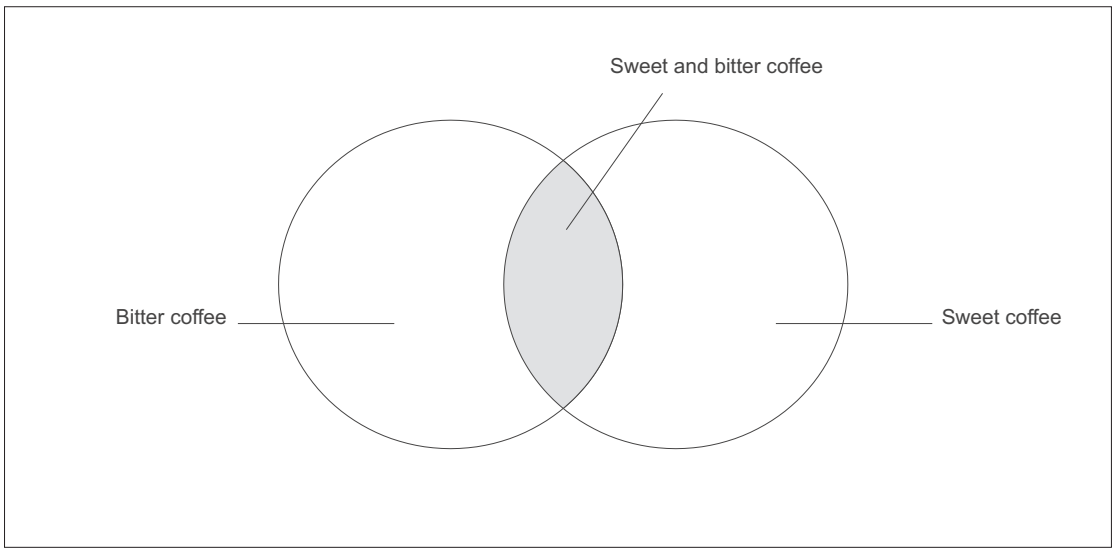


Fig. 4

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 $a \in U$, as:

$$A(a) = \begin{cases} 1 & if a \in A \\ 0 & if a \in A \end{cases}$$

$$A : U \rightarrow \{0,1\}.$$

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.

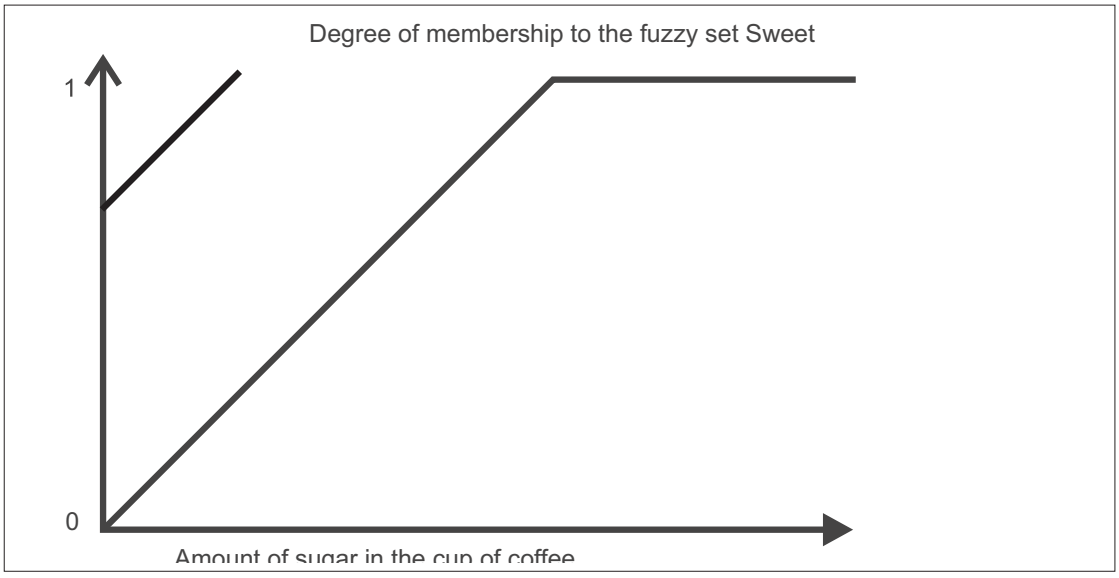


Fig.5

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 $S(x,0) = x, S(x,1) = 1 \quad \forall \; x \in I$

b $T(x,y) = 1- S(1-x,1-y) \quad \forall \; x, y \in I$

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:

$(A \cap B)(x) = T(A(x),B(x)) \qquad \forall x \in X$

$(A \cup B)(x) = S(A(x),B(x)) \qquad \forall x \in X$

The following step is the identification of T and S with some algebraic operators. In literature they are used to define:

$T(x,y) = \min(x,y) \qquad \forall x \in I$

$S(x,y) = \max(x,y) \qquad \forall x \in I$

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:

$$A \cup B = U$$

- 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:

$$A \cap B = \varnothing$$

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:

$$B(a)=1-A(a)$$

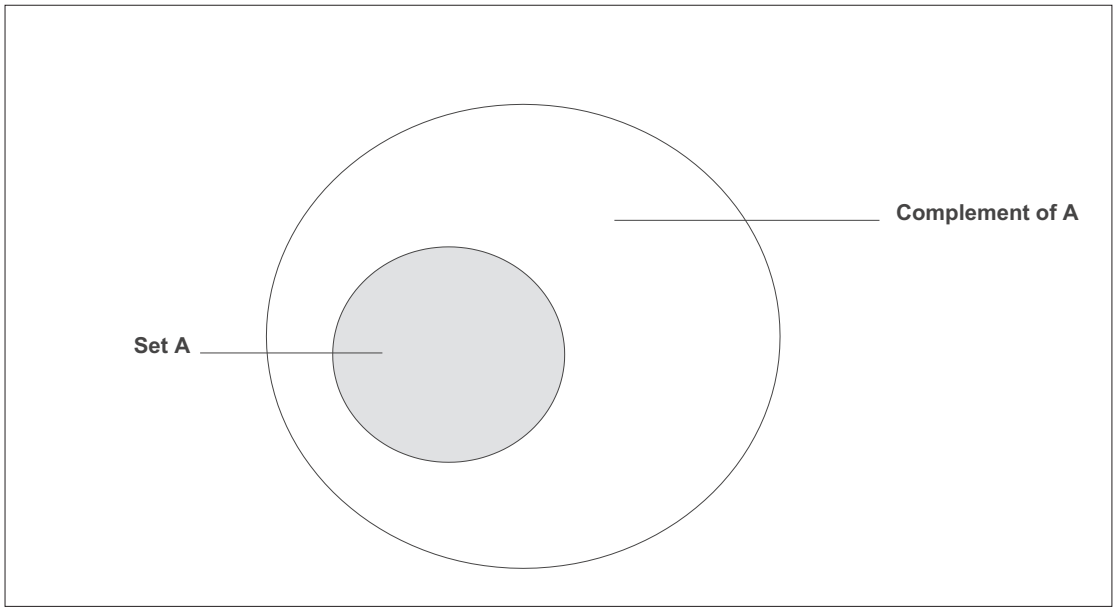


Fig. 6

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.

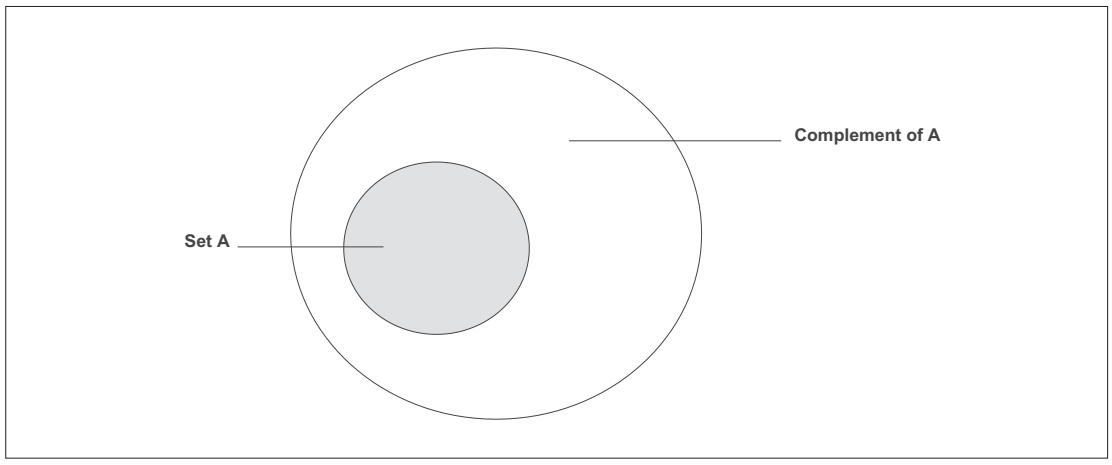


Fig.7

Set Union

The union of two sets is the set whose elements belong to any of the two sets, as shown in fig. 8.

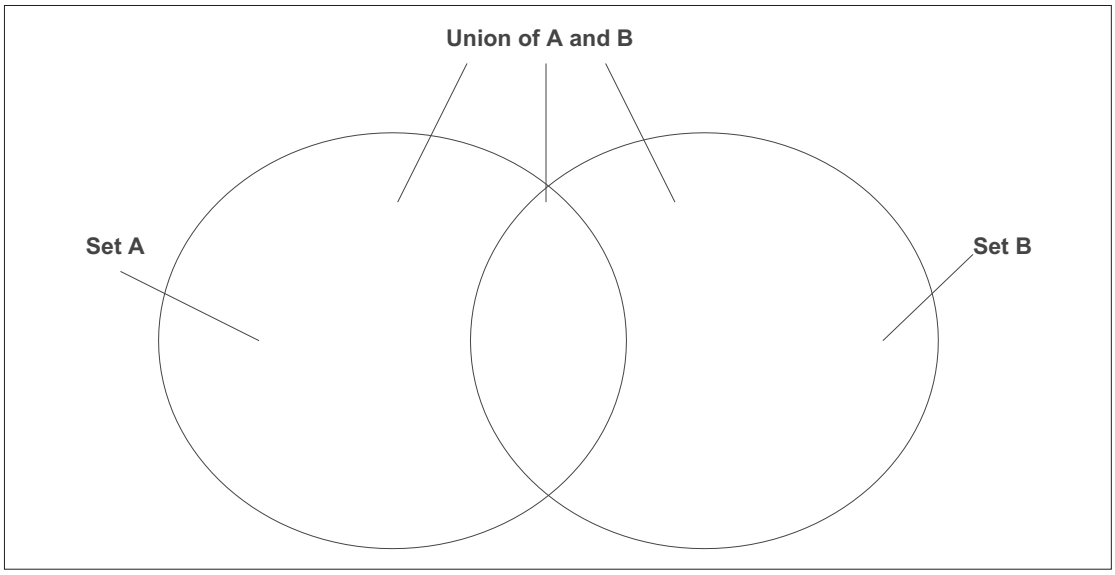


Fig.8

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.

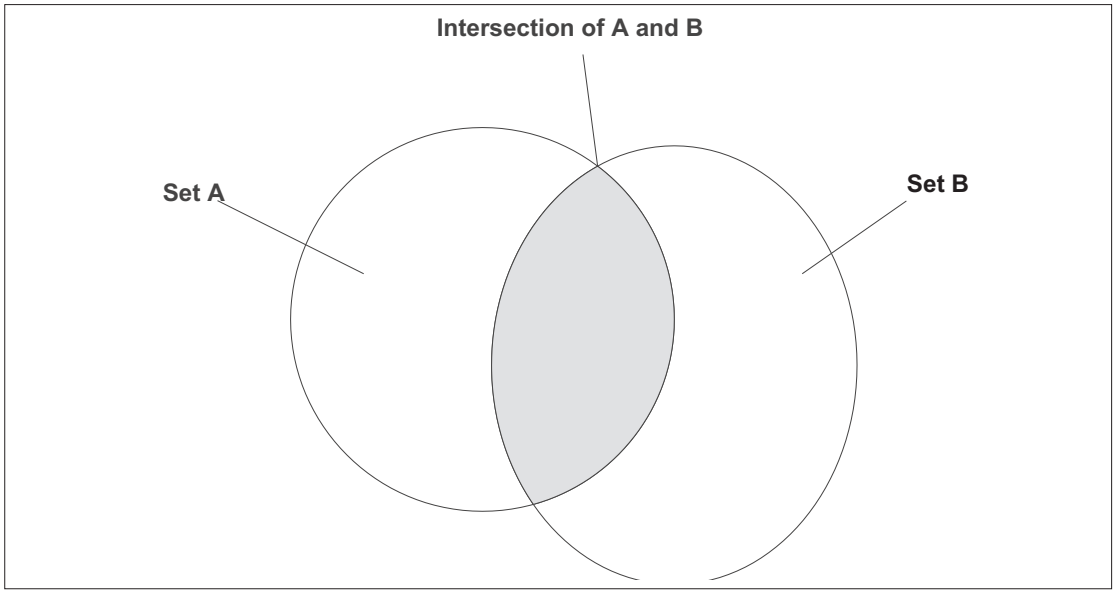


Fig.9

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.

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.

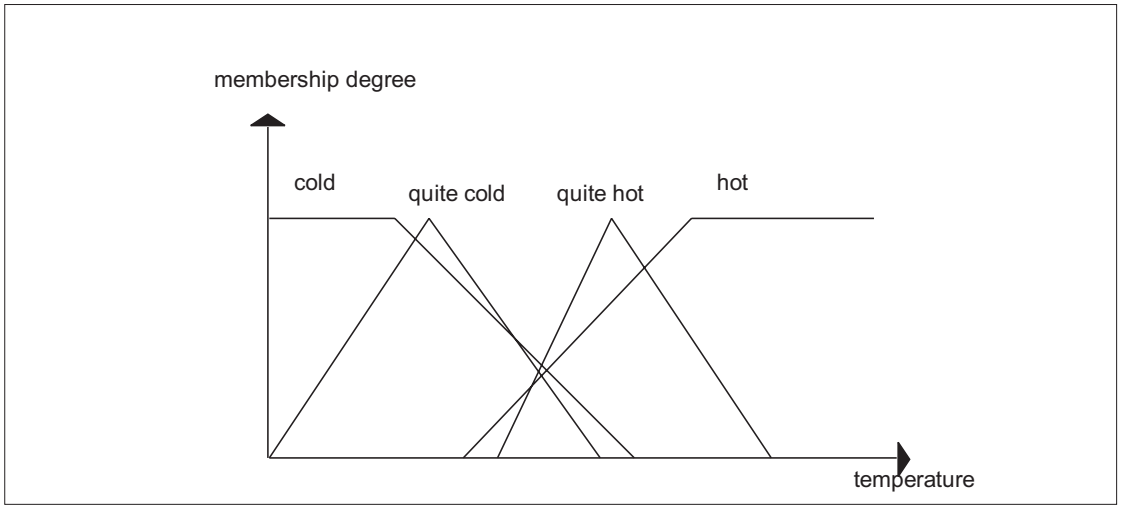


Figure 10.

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.

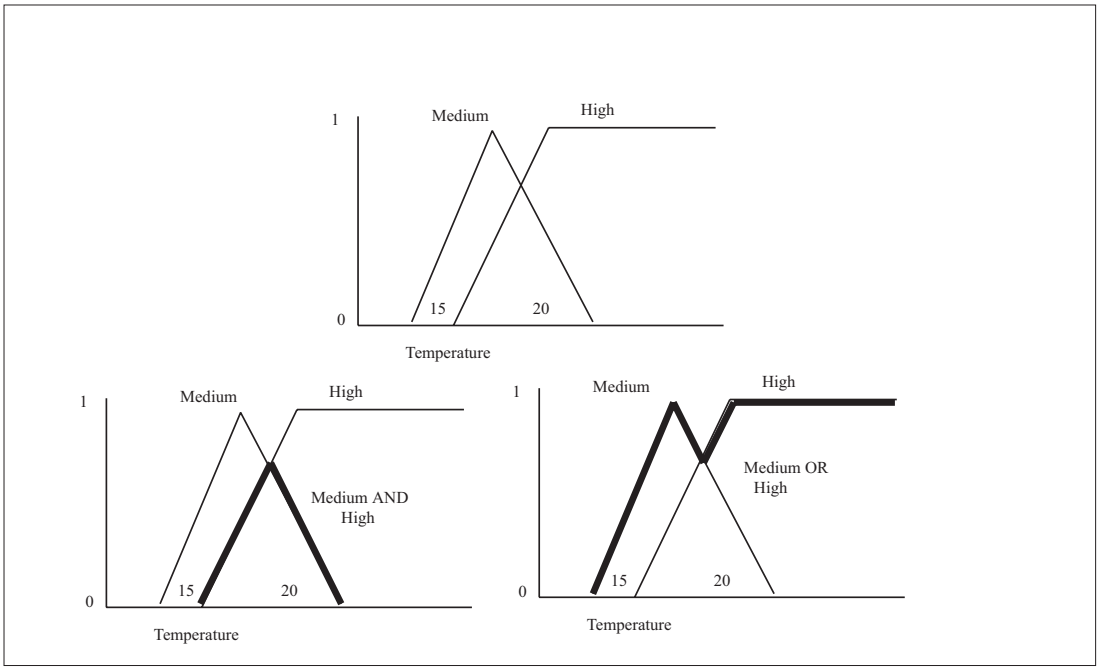


Fig.11

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	y is B
Premise 2:	IF x is A	
Conclusion:		y is 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	y is B
Premise 2:	IF x is A'	
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 X_1 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.

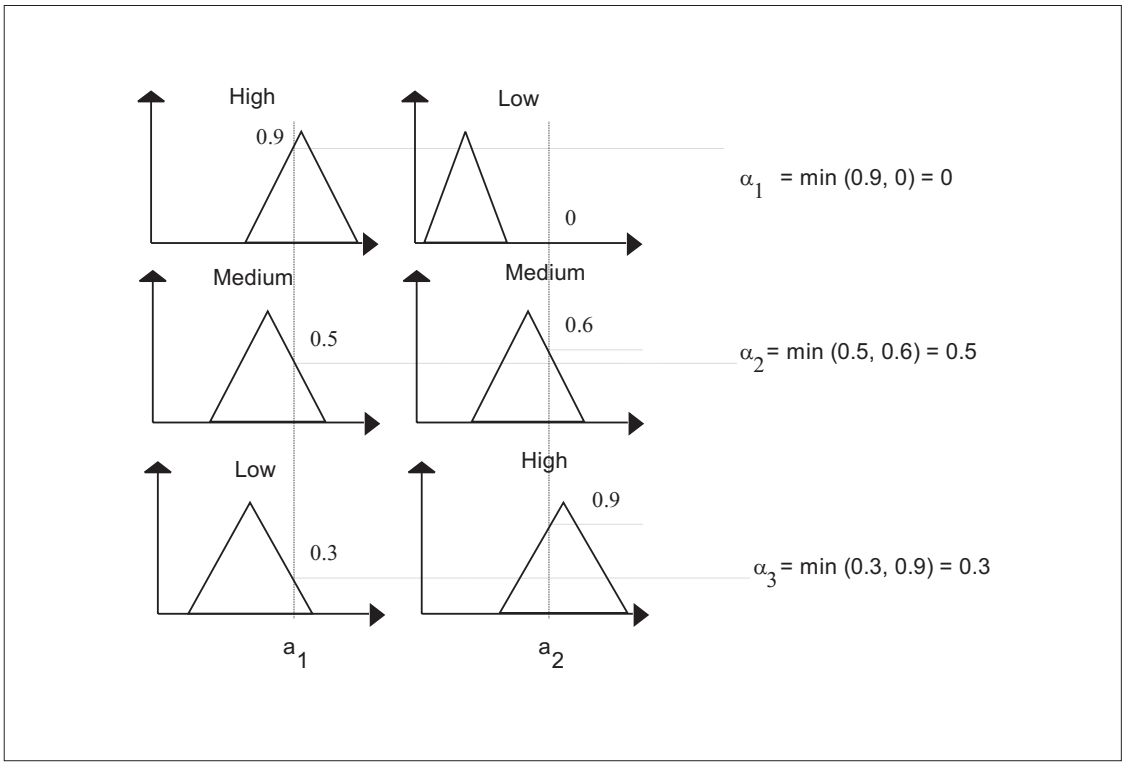


Figure 12.

The rules are used as follows.

Step 1: *fuzzyfication 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

$A'(y_i)=\min(A(y) \bullet \alpha_1)$, $B'(y_i)=\min(B(y_i) \bullet \alpha_2)$, $C'(y_i)=\min(C(y \bullet \alpha_3)$.

Fig. 13 gives a pictorial representation of the two most classical methods of inference.

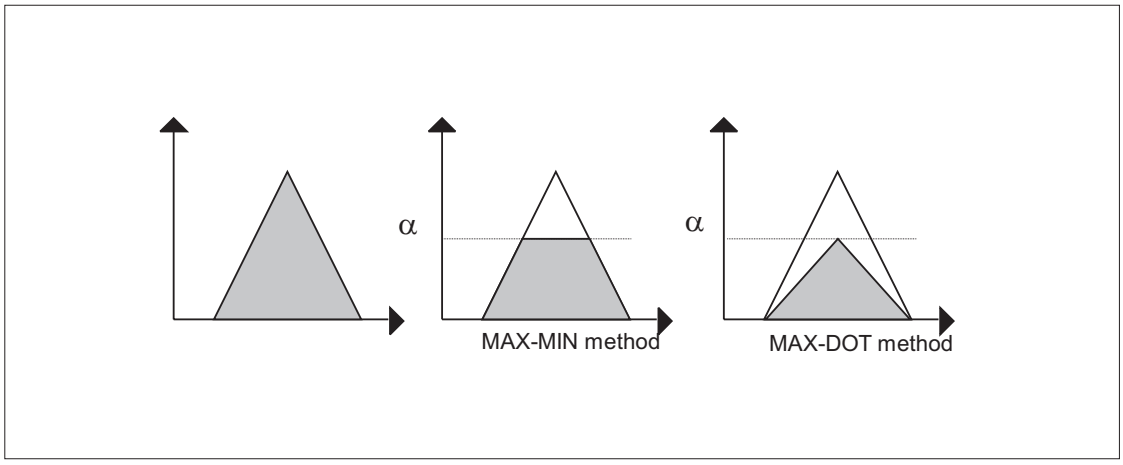


Fig.13

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.

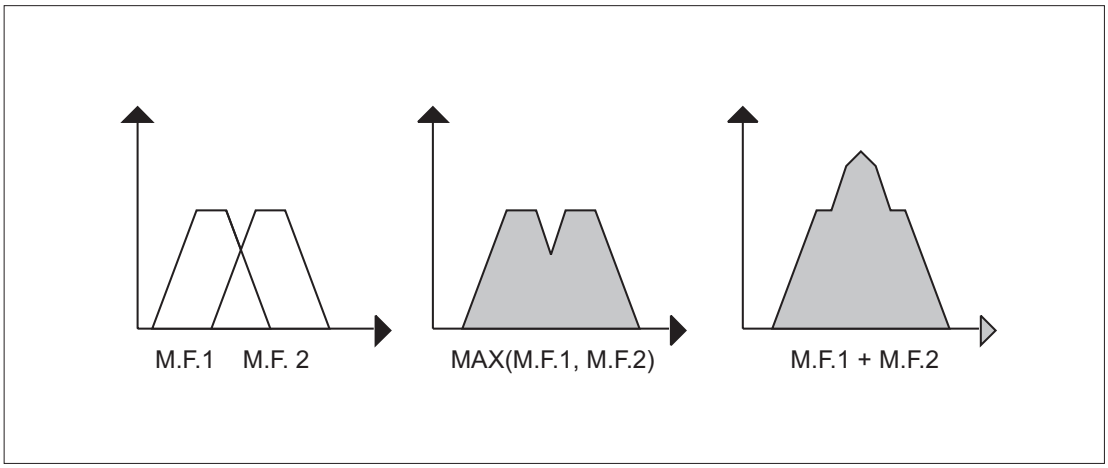


Figure 14.

The membership function G associated to the consequent is then used in the fourth step as follows.

Step 4: *Defuzzyfication 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 $U=\{u_1,U_2,\ldots,U_n\}$:

1 Center of Gravity: the output value y is given by the formula

$$y=S\ ui\ G(ui\ )\ /\ S\ G(ui\ )$$

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

$$y=S\ Ai\ bi\ /\ S\ Ai$$

where i varies over the inference rules. Ai 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 $m_1, m_2, \ldots, m_h$ are the h values where of maximum membership degree is G.

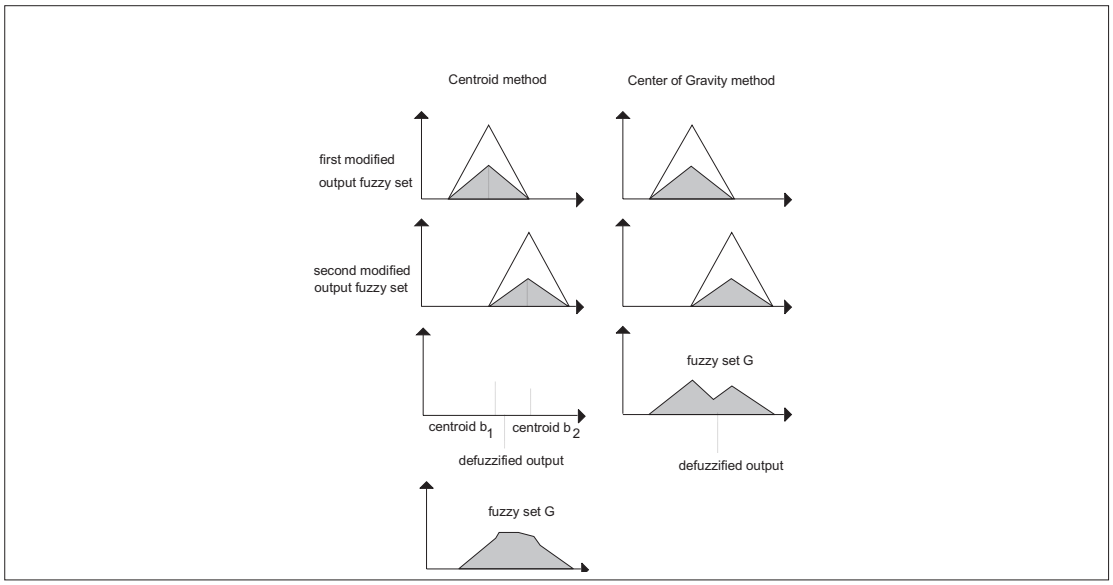


Fig.15

We conclude with a picture that summarizes the structure of a fuzzy computational model.

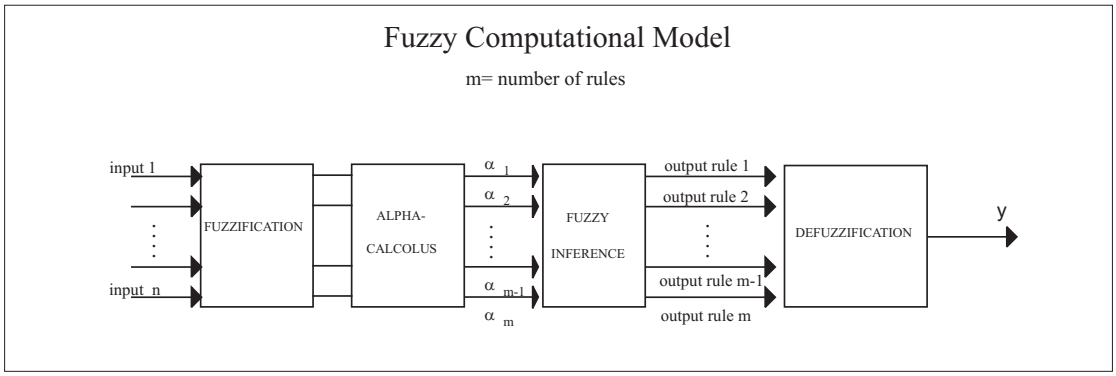


Fig. 16

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.

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 4 "
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
```

EUROPE

DENMARK
DK-2730 HERLEV
Gl. Klausdalsbrovej 491
Tel. (45-44) 94.85.33 Telefax: (45-44) 94.86.94

FINLAND
LOHJA SF-08150
Ratakatu, 26 Tel. (358-19) 32821
Telefax. (358-19) 3155.66

FRANCE
94253 GENTILLY Cedex
7 - Avenue Gallieni - BP. 93
Tel.: (33-1) 47.40.75.75
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG
20, Place des Halles
Tel. (33-3) 88.75.50.66
Telefax: (33-3) 88.22.29.32

GERMANY
D-85630 GRASBRUNN
Bretonischer Ring 4
Postfach 1122
Tel.: (49-89) 460060
Telefax: (49-89) 4605454

D-90449 NÜRNBERG
Sudwestpark, 92
Tel.: (49-911) 670408-0
Telefax: (49-911) 670408-99

D-70499 STUTTGART
31 Mittlerer Pfad 2-4
Tel. (49-711) 13968-0
Telefax: (49-711) 8661427

HUNGARY (Representative Office)
1139 Budapest Vaci UT 99
Tel.(36-1)350 5280

ITALY
20090 ASSAGO (MI)
South europe Commercial Headquarters
V.le Milanofiori - Palazzo E/5
Tel. (39) 0257546.1
Telefax: (39) 028250449

40033 CASALECCHIO DI RENO (BO)
Via R. Fucini, 12
Tel. (39) 051591914
Telefax: (39) 051591305

00161 ROMA
Via A. Torlonia, 15
Tel. (39) 064425941
Telefax: (39) 0685354438

THE NETHERLANDS
5652 AR EINDHOVEN
Meerenakkerweg 1
Tel.: (31-40) 2509600
Telefax: (31-40) 2528835

POLAND
WARSAW 00517
Ul. Marslzalkowska 82
Tel.: (0048-22) 622 0561
Telefax: (0048-22) 623 6437

SPAIN
E-08004 BARCELONA
Calle Gran Via Cortes Catalanes, 322
6th Floor, 2th Door
Tel. (34) 93 4251800
Telefax: (34) 93 4253674

E-28027 MADRID
Calle Albacete, 5
Tel. (34) 91 4051615
Telefax: (34) 91 4031134

SWEDEN
S-16421 KISTA
Borgarfjordsgatan, 13 - Box 1094
Tel.: (46-8) 58774400
Telefax: (46-8) 58774411

SWITZERLAND
1215 GENEVA 15
Route de PréBois, 20
Tel. (41-22) 9292929
Telefax: (41-22) 9292900

TURKEY
34630 FLORYA INSTAMBUL
Besyol Mah.
Florya Kavsagi Eski Londra Asfalti No.26B/10
Tel.(90) 212 624 32 64
Telefax: (90) 212 624 96 26

35030 BORNOVA IZMIR
295 SK. No:1 K:8 D:6
Tel: (90) 232 486 03 51
Fax: (90) 232 486 05 28

UNITED KINGDOM and EIRE
MARLOW, BUCKS, SL71Y
Planar House, Parkway
Globe Park
Tel.: (44-1628) 890800
Telefax: (44-1628) 890391

AMERICAS

BRAZIL
05413 SAO PAULO
R. Henrique Schaumann 286-CJ03
Tel.: (55-11) 883-5455
Telefax : (55-11) 282-2367

69050-010 MANAUS
Av. Djalma Batista, 2469 - sala 5B
Tel: +55 92 633-7303 (3 trunks)
Telefax: +55 92 633-7042

CANADA
CALGARY, Alberta T1Y 5R8
2723 37th Ave., N.E., Suite 206
Tel.: (403) 291-4001
Telefax: (403) 291-3948

NEPEAN ONTARIO K2H 9C4
301 Moodie Dr., Suite 307
Tel.: (613) 829-9944
Telefax: (613) 829-8996

MISSISAUGA, Ontario L4V 1R9
5945 Airport Rd., Suite 362
Tel.: (905) 678-9800
Telefax: (905) 678-1799

MEXICO
01070 MEXICO DF
Insurgentes Sur# 2376-604
Col. Chimalistac, San Angel
Tel.: (525) 616 4801
Telefax: (525) 616 4872

44550 Guadalajara
2347 Av. Mariano Otero
Piso 5, of. “B” Col. Verde Valle
Tel.: (52) 3-647 6081
Telefax: (52) 3-647 5231

U.S.A.
NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
Lexington Corporate Center
10 Maguire Road
Building 1, 3rd Floor
Lexington, MA 02421
Tel.: (781) 861-2650
Telefax: (781) 861-2678

ALABAMA
Huntsville - Tel.: (256) 895-9544
Fax: (256) 895-9114

ARIZONA
Phoenix - Tel.: (602) 485-6100
Fax: (602) 485-6330

CALIFORNIA
Agoura Hills - Tel.: (818) 865-6850
Fax: (818) 865-6861

Laguna Niguel - Tel.: (949) 347-0717
Fax: (949) 347-1224

San Jose - Tel.: (408) 452-8585
Fax: (408) 452-1549

COLORADO
LONGMONT
1625 S. Fordham Street, Suite 500
LONGMONT, CO 80503 - USA
Tel: +1 303 774-2530/2515/2537
Fax: +1 303 772 0720]

CONNECTICUT
Woodstock - Tel.: (860) 928-7700
Fax: (860) 928-2722

FLORIDA
Boca Raton - Tel.: (561) 997-7233
Fax: (561) 997-7554

GEORGIA
Norcross - Tel.: (770) 449-4610
Fax: (770) 449-4609

IDAHO
Boise - Tel.: (208) 376-9151
Fax: (208) 376-9109

ILLINOIS
Schaumburg - Tel.: (847) 517-1890
Fax: (847) 517-1899

INDIANA
Indianapolis - Tel.: (317) 575-5520
Fax: (317) 575-8271

Kokomo - Tel.: (765) 455-3500
Fax: (765) 455-3400

MICHIGAN
Livonia - Tel.: (734) 953-1700
Fax: (734) 462-4071

MINNESOTA
Edina - Tel.: (612) 835-3500
Fax: (612) 835-3555

MISSOURI
Kansas City - Tel.: (816) 468-6868
Fax: (816) 468-6561

NEW JERSEY
Basking Ridge - Tel.: (908) 766-7401
Fax: (908) 766-7738

Voorhees - Tel.: (609) 772-6222
Fax: (609) 772-6037

NEW YORK
Fishkill - Tel.: (914) 896-2926
Fax: (914) 897-3734

NORTH CAROLINA
Cary - Tel.: (919) 469-1311
Fax: (919) 469-4515

OREGON
Corvalis - Tel.: (541) 754 8192
Fax: (541) 754 8262

Lake Oswego - Tel.: (503) 635-7635
Fax: (503) 635-7677

PENSYLVANIA
Bensalem - Tel.: (215) 638 2958
Fax: (215) 638 2986
TEXAS
Carrollton - Tel.: (972) 466-8445
Fax: (972) 466-8387

Houston - Tel.: (281) 376-9939
Fax: (281) 376-9948

UTAH
Midvale - Tel.: (801) 256 3571
Fax: (801) 256 3578

ASIA / PACIFIC

AUSTRALIA

SYDNEY
Suite 3, Level 7, Otis House
43 Bridge Street
N.S.W. 2220 Hurtsville
Tel. (61-2) 9580 3811
Telefax: (61-2) 9580 6440

MELBOURNE
Suite 305 Level 3
3 Chester Street
Oakleigh Vic 3166
Tel. (61-3) 9568 1222
Telefax: (61-3) 9568 1999

CHINA (Liaison Offices)
BEIJING 100027 P.R.C.
Room 809, NCHK Manhattan Building,
6 Chanyangmen Beidajie,
Tel.: (86-10) 6554 4701
Telefax: (86-10) 6554 4705

SHANGHAI
Unit 1801, 18/F
Shui On Plaza
333 Huai Hai Zhong Road
Tel. (86-21) 5306 0898
Telefax: (86-21) 5306 0890

SHENZHEN 518048
52, Tao Hua Road
Futian Free Trade Zone Tel. (86-755) 359 0950
Telefax: (86-755) 359 1155

HONG KONG
Special Administrative Region
16/F., Tower 1, The Gateway 1,
25 Canton Road,
Tsim Sha Tsui, Kowloon
Tel. (852) 2861 5700
Telefax: (852) 2861 5044

INDIA(Liaison Offices)
BANGALORE 560052
Diners Business Service
26 Cunningham Road
Tel. (91-80) 267 272
Telefax: (91-80) 261 133

NOIDA 201301
Liaison Office
Plot N. 2 & 3, Sector 16A
Institutional Area
Distt Ghaziabad UP
Tel. (91-11) 9153 0965/8
Telefax: (91-11) 9154 1957

MALAYSIA
SELANGOR, PETALING JAYA 46050
Darul Ehsan
1205, Block A, Menara PJ
No 18, Jalan Persian Barat,
Tel.: (60-3) 758 1189
Telefax: (60-3) 758 1179

PENANG 11900
Unit 9-A, Lower Level 5
Hotel Equatorial
1 Jalan Bukit Jambul
Tel. (60-4) 642 8291
Telefax: (60-4) 642 8284

KOREA

SEOUL
19th Fl Kang Nam Building
1321-1 Seocho-dong, Seocho-Ku
Tel. (82-2) 3489-0114
Telefax: (82-2) 588-9030

TAEGU 701-023
18th Floor Youngnam Tower
111 Shinchun-3 Dong, Dong-Ku
Tel. (82-53) 756-9583
Telefax: (82-53) 756-4463

SINGAPORE

SINGAPORE 569508
28 Ang Mo Kio - Industrial Park 2
Tel. (65) 482 1411
Telefax: (65) 482 0240

TAIWAN
TAIPEI 106
20F, No. 207
Tun Hua South Road, Section 2
Tel. (886-2) 23788088
Telefax: (886-2) 23789188

THAILAND

BANGKOK 10110
Unit # 1315 54 Asoke Road
Sukhumvit 21
Tel.: (66-2) 260 7870
Telefax: (66-2) 260 7871

JAPAN

TOKYO
108 5 F Nisseki -
Takanawa Blg. 2-18-10 Takanawa Minato-Ku
Tel. (81-3) 3280-4120
Telefax: (81-3) 3280-4131

OSAKA 532
14 F Shin-Osaka Second Mori building
3-5-36 Miyahara Yodogawa-Ku
Tel. (81-0) 6397-4130
Telefax: (81-0) 6397-4131

ORDERING INFORMATION

INTERNAL CODE
ST52X420/KIT