WGDB7

WINDOWS DEBUGGER FOR THE ST7 EMULATOR,
SIMULATOR AND DEVELOPMENT KIT

PREFACE

Purpose of the Manual

This manual describes how to configure, start and operate the WGDB7 Windows Debugger, to debug programs written for the ST7 family of microcontrollers, in any of these development and test environments:

- Program debugging with ST7 emulators,
- Program debugging with ST7 simulators,
- Program debugging with ST7 development kits.

WGDB7 communicates with the user via the standard Windows Graphical User Interface.

It is based on the GNU Debugger (GDB) command set, and uses additional ST7-specific commands. Consequently, the manual discusses also basic GDB commands.

Audience

This book is intended for persons who have previous experience using assembler or C languages, but who are beginners in the ST7 microcontroller field and need to know how to handle related development and test tools.

Related Publications

The following publications contain useful complementary information:

- ***ST7-Family, Data Sheets***,
- ***ST7-Family, 8-bit MCUs, Product Overview***, Ref. BKST7/2
- ***Software Tools for the ST7 Family***, Ref. Doc-ST7ASMLK-SW
- ***ST7-Family, Programming Manual***,
- ***ST7-Family Development Kit, Getting Started***, Ref. Doc-ST7MDTx-DVP

Rev. 2.2

Table of Contents

Table of Contents

Organization of the Manual

This manual contains eight chapters:

Chapter 1, INTRODUCTION, discusses the concepts of simulation- and emulation-aided debugging.

Chapter 2, INSTALLING WGDB7, explains how to install WGDB7 with simulator or emulator.

Chapter 3, **PREPARING PROGRAMS FOR DEBUGGING** is a general discussion on source program origin.

Chapter 4, OPERATING WGDB7, explains how to use the debugger.

Chapter 5, CUSTOMISING WGDB7, explains how to tailor WGDB7 to you own needs.

Chapter 6, WORKING WITH WORKSPACES, is devoted to program workspace management.

Chapter 7, USING GDB7 COMMANDS, provides information on GDB7 commands that can be used with WGDB7.

Chapter 8, QUESTIONS AND ANSWERS, answers some general questions about WGDB7 operation.

1 INTRODUCTION

1.1 An ST7-Family Debugging Tool

WGDB7 is a debugger for programs developed for the ST7 family of microcontrollers and based on either the STMicroelectronics ST7 Macro-Assembler and linker ASM/LYN or the Hiware ST7 C Toolchain.

WGDB7 runs under Windows 3.x™, Windows 95™ or Windows NT ™.

1.2 Emulators, Simulators, Development Kits

You can use WGDB7 with either of the following device:

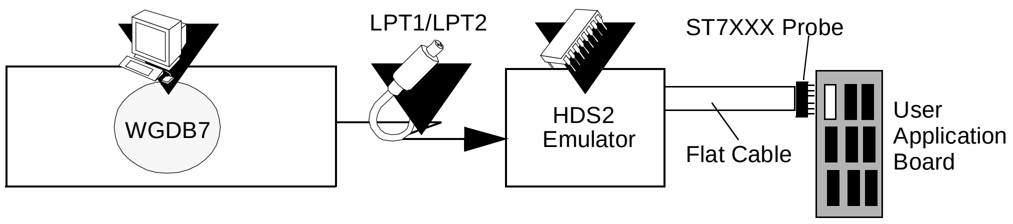
- ST7 Emulator,
- ST7 Simulator,
- ST7 Development Kit.

In the following, the icons , and will be used to point out topics related to features specific to emulators, simulators and development kits, respectively.

The **ST7 HDS2 Emulator** is a *hardware device* that **behaves like** an actual ST7 microcontroller and provides real-time execution of ST7 programs.

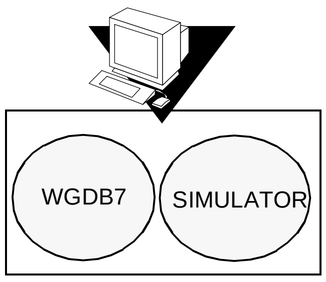
The emulator can be connected to your own board, enabling you to emulate the target application hardware and software.

The following diagram summarizes the ST7 HDS2 Emulator operating configurations:



The **ST7 Simulator** is a *program* that **simulates** the execution of ST7 programs, instruction by instruction. The behaviour of the peripherals is also simulated.

The following diagram shows the WGDB7 simulator operating configuration:

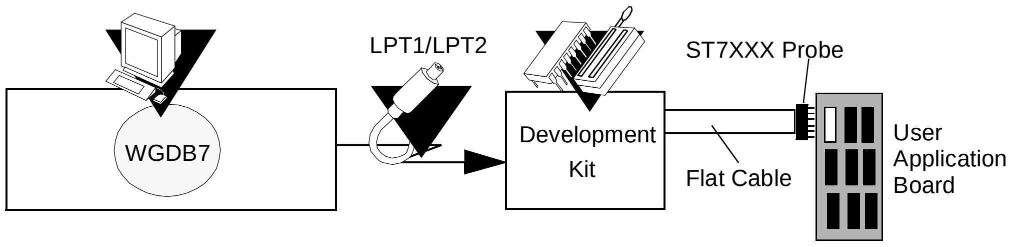


You may see that, in this case, no external hardware is required. This is the major difference between the debugging methods presently available: using a simulator or an emulator.

A **Development Kit** is an assembly of hardware and software components that combines development, debugging and chip-programming functions. It covers an extensive development and test field.

The Development Kit board is connected through a parallel port interface to a PC used to monitor debugging and device programming operations.

The following diagram summarizes the Development Kit operating configurations:



For more information on a particular Development Kit, see the appropriate ***ST7 Family, Development KIt, User Manual***.

Simulation-aided debugging lets you develop and test your application(s) long before your hardware is finished or available. This could save time and provide you with some comfort regarding your own application development plans. Also, this debugging method relieves you from any premature, undesirable, hardware dependency. Lastly, this is the cheapest solution.

Some real-time debugging features such as the tracing facility and the logic analyser function are not available, however. Nevertheless, you will benefit from new other features such as time measurement, and time-dependent break management.

Emulation-aided debugging lets you also develop and test your application(s) long before your hardware is finished or available. Although it is more hardware-dependent than the simulation option, because emulators have components specific to various microcontroller families, it presents the major advantage of supporting real-time testing and debugging in the user application board environment. Also, it supports the tracing and logic analyser debugging features that can be used in conjunction with the emulators.

The **Development Kit** and the **Emulator** debugging capabilities are very close. Compared with the Simulator and the Emulator, the Development Kit shows some differences, however, summarized in the table “WGDB7 Debugging Features for the Development Kit, Emulator and Simulator” on page 9

1.3 WGDB7 Main Features

1.3.1 Displaying Data

WGDB7 can display data in either ‘***normal***’ or ‘***hot***’ mode.

In ***normal display mode***, data is displayed as it was when you chose to view it, and it is not automatically updated. Data is displayed on a white background.

In ***hot display mode***, data is updated every time the execution of the program you are debugging is suspended, and is displayed on a yellow background.

1.3.2 Entering Data

When you operate WGDB7, you enter data in standard Windows dialog boxes. In some cases, you’ll notice that when you update fields, the field name is highlighted in red. This indicates that you have changed a value but the changes have not yet been implemented by the emulator or simulator. You must press the ***Enter*** key when the cursor is on that field to implement the changes you made.

1.3.3 Workspaces

WGDB7 enables you to save and load workspaces. Workspaces are snapshots of windows and option choices that are taken when you close a program. Each program you debug using WGDB7 has its own default workspace definition. When you load a program, the workspace that you were using when you last closed it is restored, thus you can continue working from where you left off. You can also save workspace definitions at any time, so that you can restore them at a later date. See “Working with Workspaces” on page 64 for further details.

1.3.4 Using GDB Commands

You can execute general purpose GNU and ST7-specific debugging commands through a command line interface, via the ***Console*** option in the main menu, so that you can use the commands that are not built into the WGDB7 Windows interface.

1.3.5 WGDB7 Debugging Capabilities

WGDB7 enables you to execute ST7 programs, and view the contents of the ST7 data and program memory as the program progresses. You can examine source code, as you would with a C-language source program, and assembler code. Program execution history can be viewed at source or instruction level. WGDB7 lets you read and write all ST7 registers and memory locations.

The debugging capabilities of WGDB7 depends on which version of the debugger is implemented: WGDB7 with Emulator or Simulator.

Differences are summarized below in Table 1, “WGDB7 Debugging Features for the Development Kit, Emulator and Simulator,” on page 9.

As a general rule, WGDB7:

- Enables you to execute source code and machine code line by line. A function call can optionally be considered as a single instruction, depending on the level of detail required.
- Lets you set software breakpoints on source or disassembled code, that stop the program running when a chosen instruction is reached.
- Lets you set hardware breakpoints, which either stop the program running when a predefined area of memory is accessed or send a signal to the output triggers (only with emulators and development kits in the later case).
- Enables you to view and modify the simulated/emulated ST7 memory and register contents.
- Enables you to view and modify assembly symbols or C-language variables in their type-

defined format with possible display options such as binary, decimal or hexadecimal.

- Enables you to view the stack contents.
- Enables you to write small programs or modify them at assembly mnemonic level, using the Online Assembler feature.
- Automatically executes GDB7 command batch files, with or without the WGDB7 graphical interface, at start-up.
- Enables you to quickly switch from a target debugging mode to the other, for example from the simulator to the emulator, maintaining all workspace context information such as windows position, software breakpoints, etc.

The features supported by WGDB7 when running with an emulator, a simulator or a development kit are summarized in the following table:

.

Table 1. WGDB7 Debugging Features for the Development Kit, Emulator and Simulator

Description	**Debugger +DEVKIT**	**Debugger +EMU**	**Debugger +SIMU**
Hardware Test	YES	YES	NO
Loading Programs, Viewing Program Details	YES	YES	YES
Executing Loaded Programs	YES	YES	YES
Managing Software Breakpoints	YES	YES	YES
Managing Hardware Breakpoints	YES	YES	YES
Viewing ST7 Resources	YES	YES	YES
Viewing Disassembled Program Code	YES	YES	YES
Viewing Register Contents	YES	YES	YES
Watching Symbols, Types of Variables, Expressions	YES	YES	YES

Table 1. WGDB7 Debugging Features for the Development Kit, Emulator and Simulator

Description	Debugger +DEVKIT	Debugger +EMU	Debugger +SIMU
Entering, Assembling Mnemonics to Emu /Simu ST7 Memory	YES	YES	YES
Using GDB7 Commands	YES	YES	YES
Choosing Emulated/Simulated ST7 Micro Name	YES	YES	YES
Changing ST7 Emulated/Simulated Memory Mapping	YES	YES	YES
Displaying and Modifying Configuration Options	YES	YES	YES
Customising the Debugger	YES	YES	YES
Working with Trace Buffer	NO	YES	NO
Accessing Logic Analyser Information	NO	YES	NO
Simulating Pin Input/Output Signals	n.a.	n.a.	YES
Time Management	NO	NO	YES
Output Trigger Management	YES	YES	NO
Stack Overflow/Underflow Stop	YES	NO	NO
Non-existent Memory Stop	YES	NO	NO
Write Protect Memory Stop	YES	NO	NO
Input Trigger Stop	YES	NO	NO

2 INSTALLING WGDB7

2.1 Hardware and Software Requirements

To run WGDB7, you must have the following hardware and software:

- A PC with a 386 or higher processor
- At least 5 Mbytes hard disk space
- At least 4 Mbytes RAM
- Microsoft Windows 3.1x, Windows 95 or Windows NT
- For Emulators and Development Kits: Your PC parallel port must have been configured as an output-only type port. WGDB7 does not support EPP, ECP or bi-directional parallel port configurations.

2.2 Compatibility

WGDB7 is compatible with all families of Development Kits and ST7 HDS2 emulators.

2.3 Basic Installation Procedure

To install WGDB7, follow these steps:

1 Insert the delivery CD-ROM into your CD-ROM drive.

2 Using Windows Explorer, open the root directory of your CDROM drive

3 Find the **`St7tools`** sub-directory in the CDROM and run

`St7tools\disk1\setup.exe`

to start the installation program.

4 Follow the installation instructions. You may:

- change the destination folder (**`c:\St7tools`** by default),
- choose to install only a selection of the software components, to spare disk space,
- specify the connection port name, LPT1 or LPT2 (LPT1 by default) for emulators and development kits.

2.4 Installing WGDB7 in the Windows NT Environment: Completing the Installation

If you are installing WGDB7 on a Windows NT platform, and you want to use WGDB7 with an HDS Emulator or a Development Kit over a parallel port, you must install the **`genport.sys`** parallel port driver that is supplied in the diskette labelled “NT Driver Installation” included in the delivery package.

For this, you must have the Windows NT system administrator’s rights.

To start the installation of the driver, run the **`setup.exe`** installation program in the supplied diskette and follow the instructions.

2.5 Powering Up the Emulator

2.5.1 Connecting the Emulator

To connect your emulator to your PC.

1 Connect the parallel cable of the emulator to the LPT1 or LPT2 parallel port of your PC,

2 Switch on the emulator,

3 Start WGDB7 by following the instructions given in “Starting WGDB7” on page 18.

Look at the emulator front panel: a red LED should be turned on indicating that WGDB7 is communicating with the emulator.

By default, when you start WGDB7 in this environment, the emulator program memory is set to undefined values.

2.5.2 Running the Hardware Test

The hardware test lets you check that your emulator is correctly connected, configured and working.

You can test components of the emulator individually or all at the same time.

To run the hardware test:

1 Run WGDB7.

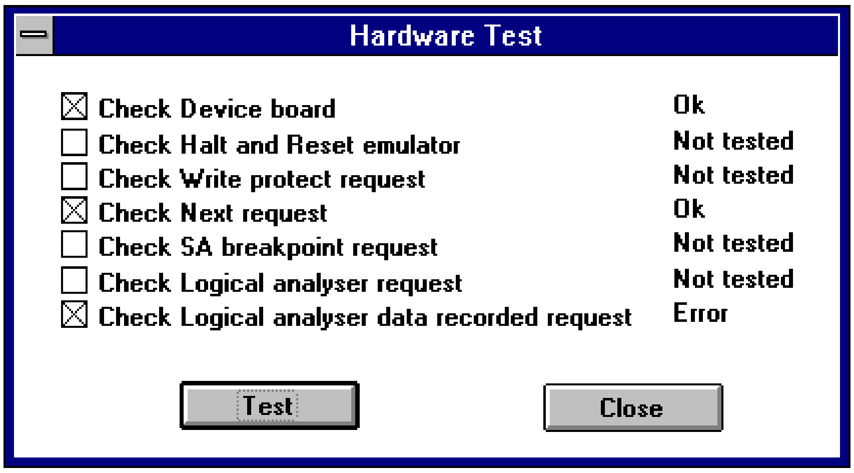
2 On the **Commands** menu, click **Hardware Test**.

The Hardware Test dialog box opens.

3 Select the components you want to test from the list, by clicking the appropriate check boxes. When a box is checked, its component is tested.

4 Click the **Test** button to start the test.

Once the test has been performed, the results are displayed in the **Hardware Test** dialog box:



☒ Check Device board	Ok
☐ Check Halt and Reset emulator	Not tested
☐ Check Write protect request	Not tested
☒ Check Next request	Ok
☐ Check SA breakpoint request	Not tested
☐ Check Logical analyser request	Not tested
☒ Check Logical analyser data recorded request	Error

NOTE: In case of test failure, contact your local support representative.

Click the **Close** button to close the Hardware Test dialog box. You are directed to the WGDB7 main window.

2.5.3 Connection Troubleshooting

The following messages may be displayed when you start WGDB7 with an emulator:

Message:

```
Check that the emulator is switched on
and the parallel cable is properly connected
```

Meaning:

No connection has been established between the emulator and your PC.

Action:

1 Exit WGDB7

2 Check the following:

  i Your emulator is switched on,

  ii You switched on the emulator after switching on your PC. If you switched on the emulator before switching on your PC, switch the emulator off then on again.

  iii The cable is correctly connected to your PC.

Message:

```
No message received from the emulator
```

Meaning:

Configuration problem.

Actions:

- If you are running an HDS2 emulator on a Windows NT platform, check that you have correctly installed the Windows NT driver that is supplied on the diskette labelled “NT Driver Installation Diskette”.
- Check that your PC parallel port has been configured as an output-only parallel type port (BIOS settings). WGDB7 does not support EPP, ECP or bi-directional parallel port configurations.

2.6 Powering Up the Development Kit

2.6.1 Connecting the Development Kit

Proceed as follows:

1 Connect the evaluation board (P2 connector) to the parallel port (LPT1 or LPT2) of your PC via the appropriate cable.

NOTE: The supplied interface cable has been tested in order to operate properly on most PCs. Do not use any other cable, especially if it is longer than the one provided by STMicroelectronics: the board may not operate properly.

The cable should be connected directly to the DB-25 female connector of the PC parallel port. This connector is similar to the one installed on the board. Do not insert any additional cables or switchboxes between the PC and the board: a malfunctioning of the board may result.

2 If necessary, connect the probe (P3 connector) using the flat cable that has been supplied in the package.

3 Power up the board.

NOTE: To avoid risks of short circuits, first connect the plug-in power supply pack jack into the P1 female jack connector on the board, then plug the power supply pack into the mains AC outlet.

If the board is fed via the complementary two-point connector make sure that the right feeders lead to the right polarities.

The green POWER LED lights up.

2.6.2 Running the Hardware Test

The hardware test lets you check that the evaluation board is correctly connected, configured and working.

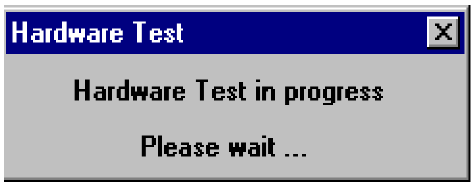
To run the hardware test:

1 Run WGDB7.

2 On the Commands menu, click **Hardware Test**.

The components of the development kit are tested one after the other

During the tests the following message box is displayed:

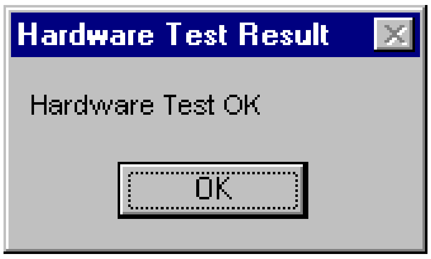


Hardware Test

Hardware Test in progress

Please wait ...

Upon successful completion of the tests the following message box appears:



Hardware Test Result

Hardware Test OK

NOTE: In case of test failure, contact your local support representative.

Click **OK** to close the Hardware Test Result message box.

3 PREPARING PROGRAMS FOR DEBUGGING

You can debug programs that were generated using either the STMicroelectronics ST7 Macro-Assembler or the Hiware ST7 toolchain.

In both cases, make sure that the ***Reset Vector*** is correctly initialised, so that the Run and Reset commands operate correctly.

Some example macro-assembler source programs are provided in the **Sample** subdirectory of the WGDB7 installation directory.

To be able to load a program, WGDB7 must have access to the ***.s19** file in the **Srecord** format or the **.abs** file for the Hiware ST7 toolchain.

From the ***.s19** file, WGDB7 can obtain the symbol files that were generated by the toolchain. For the STMicroelectronics ST7 Macro-Assembler, symbol files include **.MAP**. and **.LST** files.

To generate the correct files for WGDB7, when you assemble programs using the STMicroelectronics ST7 Macro-Assembler, you must use the following options:

```
asm -li macrost7.asm                      -- assemble with listing(*)
lyn macrost7.obj,macrost7                 -- link and generate map
asm macrost7.asm -sym -fi=macrost7.map -- update the listing with symbols(*)
obsend macrost7,f,macrost7.s19,srec     -- generate srec code
```

or:

```
obsend macrost7,f,macrost7.hex,intel    -- or generate intel hex code
```

(*) to be repeated for each module linked in the application.

Make sure that you use the STMicroelectronics ST7 Macro-assembler version pack 8 or later.

If no symbol file is provided with the **.s19** or **.hex** file, you can still perform some debugging tasks, such as viewing ST7 memory contents, viewing disassembled code, displaying and modifying register values, executing programs and viewing trace buffer contents (not applicable in the later case for simulators).

For the ST7 C compiler, refer to the compiler documentation for information on how to generate debug information.

For more information on Assembler directives, see “*Software Tools for the ST7 Family*”.

4 OPERATING WGDB7

4.1 Introduction

This section describes how to perform the basic tasks that you will typically carry out when you debug a program using WGDB7. It aims to get you quickly started with WGDB7. For detailed information about windows, menus and dialog boxes, and on how to perform all tasks, use the WGDB7 online help (see “Getting Help” on page 19).

4.2 Starting WGDB7

The WGDB7 debugger is installed in the installation root directory. By default, you will find it in:

```
C:\St7tools
```

To start WGDB7, just click the appropriate icon (or name in the cascading menus) in the Windows desktop.

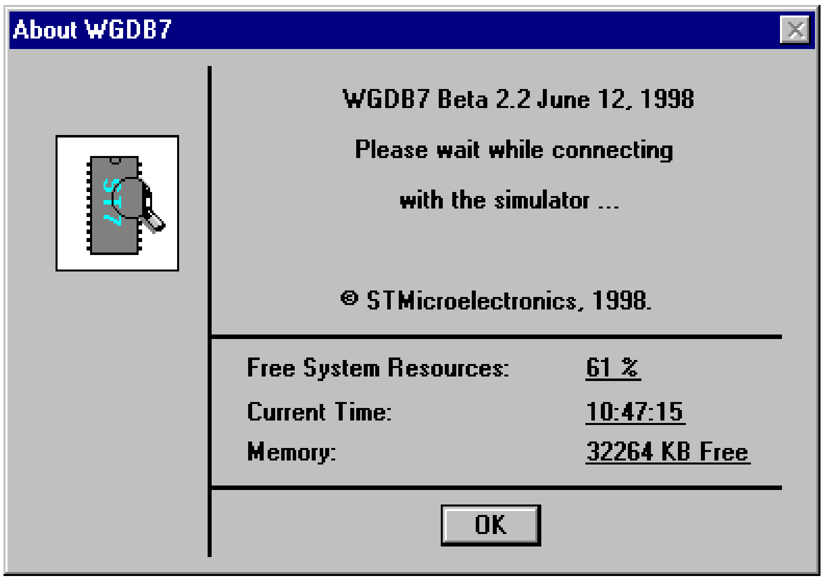
Depending on which tool you installed, you may find icons (or names) for:

WGDB7 Emulator

WGDB7 Development Kit

WGDB7 Simulator

The WGDB7 Introductory window appears (simulator example):



About WGDB7

WGDB7 Beta 2.2 June 12, 1998

Please wait while connecting

with the simulator ...

© STMicroelectronics, 1998.

Free System Resources: 61 %

Current Time: 10:47:15

Memory: 32264 KB Free

The debugging session starts a few seconds later.

To get accustomed to the product, you may start one of the sample programs provided with the package, via the ***File/Open*** debugger menu chain.

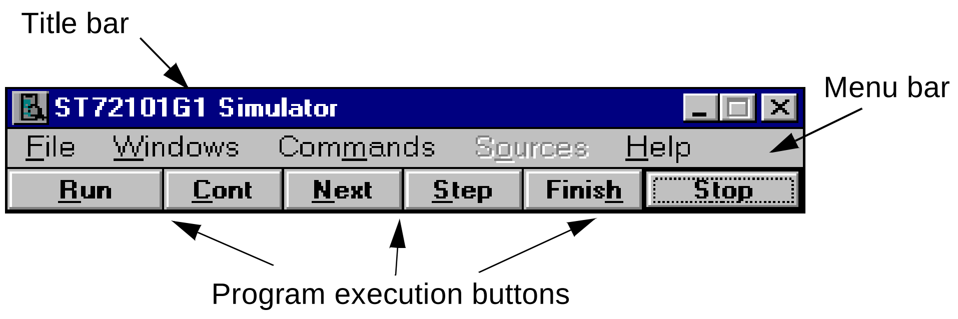
The sample programs can be found in:

`C:\<install folder>\sample`

where **`<install folder>`** is the installation root folder (directory), **`St7tools`** by default.

Once WGDB7 starts, the main window opens (simulator example):

The Title bar will later display the name of the program that is currently loaded (see "Loading



a Program" on page 29).

The Menu bar displays the available menus. Clicking a menu name opens it. Since no program is loaded, all program-related menus are greyed out and thus cannot be accessed.

The program execution buttons provide you with quick access to the program execution functions. See "Executing Loaded Programs" on page 35.

NOTE: All the task procedures described in this book start from the main window.

4.3 Getting Help

WGDB7 enables you to access two types of help messages:

- Task-based help, where you choose a task about which you want information.
- Context-sensitive help, that displays information about the window or dialog box that is currently active.

To access task-based help:

1 From the Help menu in the main window click Contents.

2 Click the task about which you want further information.

To access context-sensitive help on the window or dialog box that is currently active:

1 On the window or dialog box, click the **Help** button if there is one, otherwise click the [**F1**] key.

A help window opens displaying the window or dialog box that is currently active.

2 In the help window, click on the item, such as a menu name or field, about which you want further information.

4.4 Emulator/Simulator Configuration Options

Once you have started the debugger, even if you have not yet loaded a program, you can view and configure some emulator/simulator-specific settings.

These are:

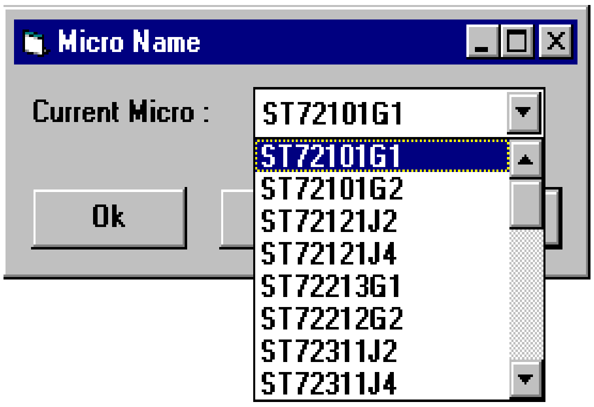
- The ST7 micro name that is being emulated or simulated.
- The emulated/simulated ST7 memory mapping.
- Other emulator configuration options, such as the clock speed. The configurable options depend on the emulator family you are using. Refer to the documentation provided with your ST7 HDS emulator for further details.

When you close an application, a configuration file (***<application>.CF7***) that contains memory mapping information is automatically generated. The definitions stored in this file are restored when reloading the application. This configuration file includes the micro name, the memory configuration, micro configuration and the trace analyser settings. When WGDB7 is used with a simulator, this information is saved in the application workspace.

4.4.1 Choosing the Emulated/Simulated ST7 Microcontroller

1 On the Commands menu, click **Micro Nam**e.

The name of the ST7 that is currently being emulated/simulated is displayed in the **Micro Name** dialog box



2 To choose another ST7 , select the appropriate name from the drop down list. Click **OK**.

The emulator/simulator is now configured for the selected ST7 microcontroller.

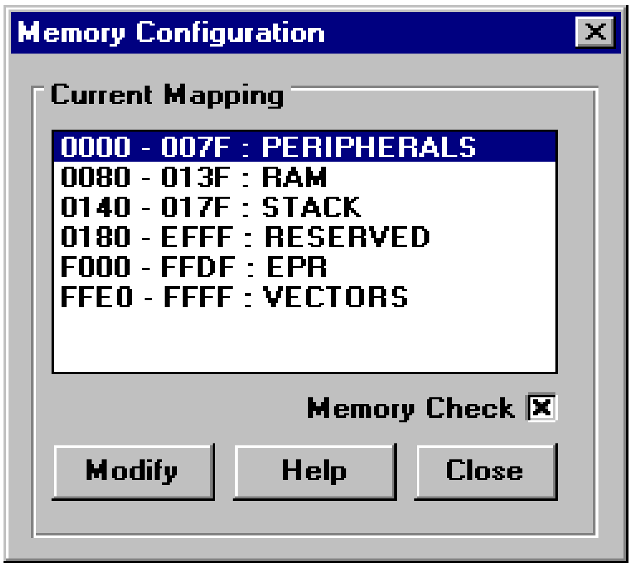
4.4.2 Changing the Emulated/Simulated ST7 Memory Mapping

1 On the Commands menu, click **Memory Configuration**.

The ***Memory Configuration*** dialog box opens, displaying the defined memory mapping for the emulated/simulated ST7:

WGDB7 with Simulator:

2To modify or create a mapped area, click the **Modify** button. The New/Modify

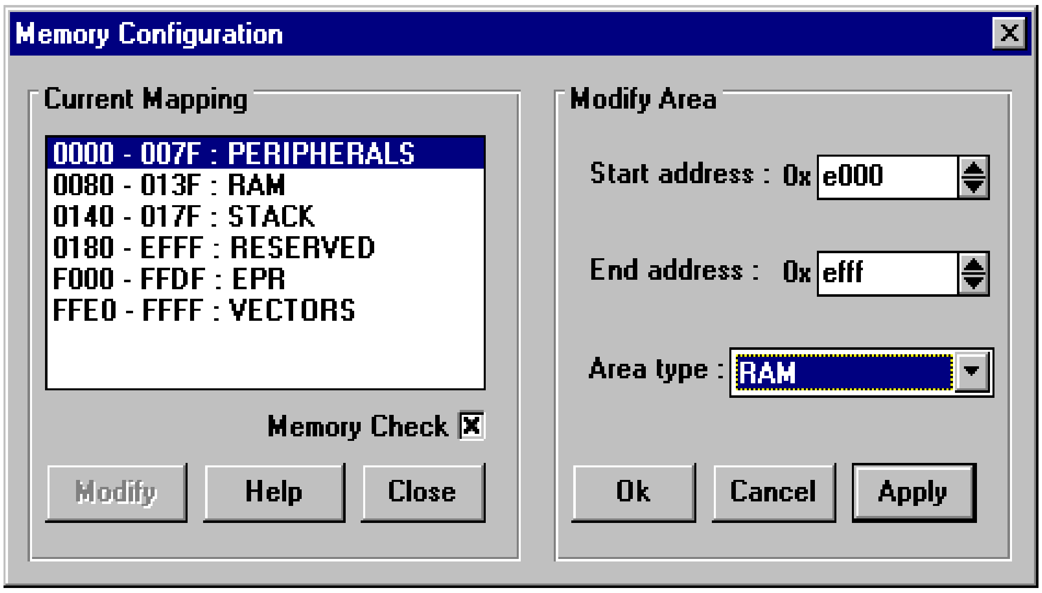


0000 - 007F : PERIPHERALS
0080 - 013F : RAM
0140 - 017F : STACK
0180 - EFFF : RESERVED
F000 - FFDF : EPR
FFE0 - FFFF : VECTORS

Memory Check [✓]

Map window opens:

3 Next, enter in the appropriate fields the **Start** and **End address** of the memory area you



Memory Configuration

Current Mapping

```
0000 - 007F : PERIPHERALS
0080 - 013F : RAM
0140 - 017F : STACK
0180 - EFFF : RESERVED
F000 - FFDF : EPR
FFE0 - FFFF : VECTORS
```

Memory Check [✓]

Modify Area

Start address : 0x e000

End address : 0x efff

Area type : RAM

want to define/modify.

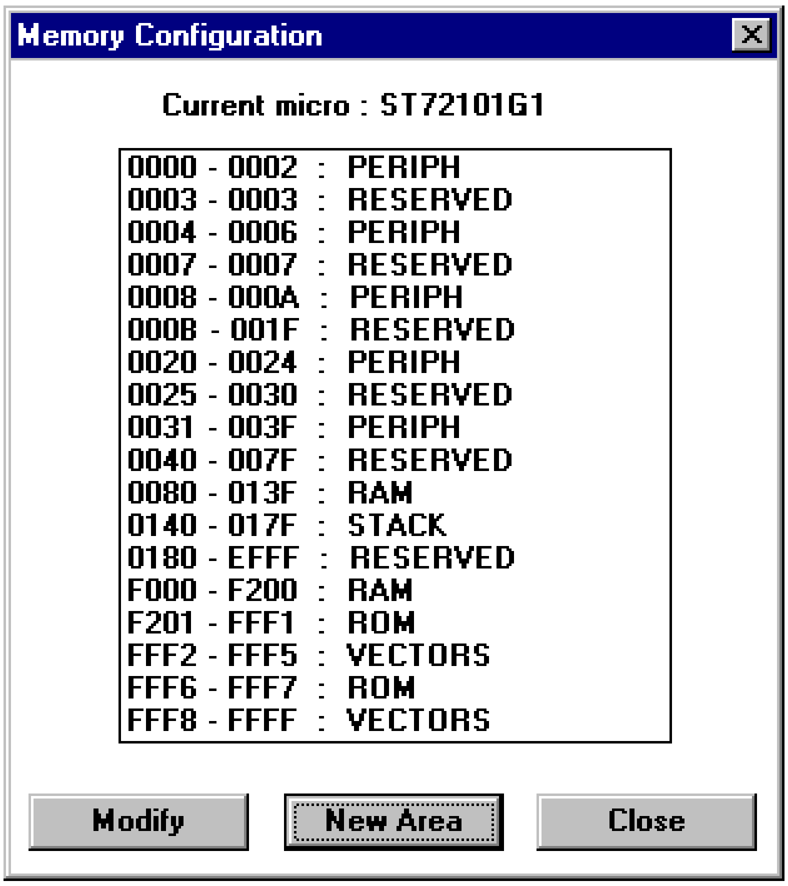
4 From the **Area type** combo box, select the type of memory for the above area: Peripherals, Stack, RAM, ROM, Vectors, or Reserved.

5 Click **Apply** to implement each creation/change.

6 Click the **OK** button to confirm the changes.

The new or modified memory area is now mapped.

WGDB7 with Emulator or Development Kit:



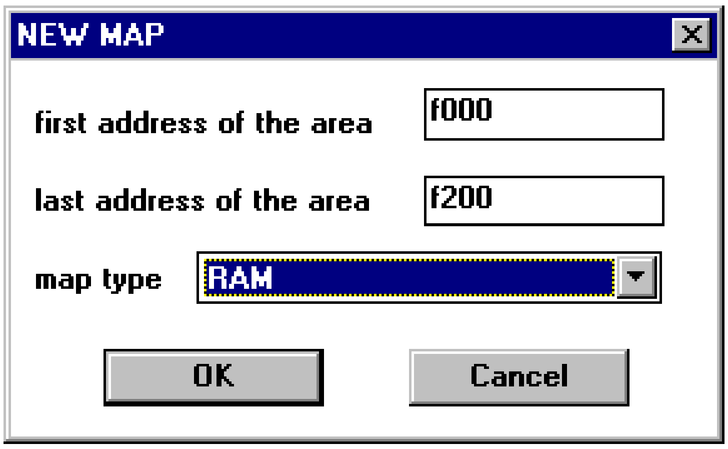
Current micro : ST72101G1

```
0000 - 0002  :  PERIPH
0003 - 0003  :  RESERVED
0004 - 0006  :  PERIPH
0007 - 0007  :  RESERVED
0008 - 000A  :  PERIPH
000B - 001F  :  RESERVED
0020 - 0024  :  PERIPH
0025 - 0030  :  RESERVED
0031 - 003F  :  PERIPH
0040 - 007F  :  RESERVED
0080 - 013F  :  RAM
0140 - 017F  :  STACK
0180 - EFFF  :  RESERVED
F000 - F200  :  RAM
F201 - FFF1  :  ROM
FFF2 - FFF5  :  VECTORS
FFF6 - FFF7  :  ROM
FFF8 - FFFF  :  VECTORS
```

7 To modify a mapped area, select the area and click the **Modify** button.

To create a new area, click the **New Area** button.

The New/Modify Map window opens:


NEW MAP

first address of the area f000

last address of the area f200

map type RAM

8 Next, enter in the appropriate fields the **First** and **Last address**s of the memory area you want to define/modify.

9 From the **Map type** combo box, select the type of memory for the above area: Peripherals, Stack, RAM, ROM, Vectors, or Reserved.

10 Click the **OK** button to confirm the changes.

The new or modified memory area is now mapped.

4.4.3 Viewing and Setting Additional Emulator Configuration Options

1 On the Commands menu, click **Micro Configuration**.

The configuration options available for the currently installed emulator or simulator are displayed.

The options that you can configure depends on the current debugging mode (with simulator, emulator or development kit).

This causes one of the following windows to open:

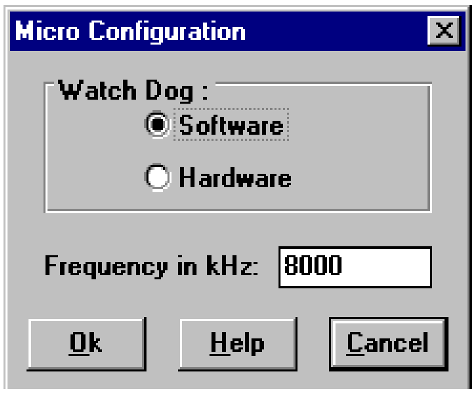
WGDB7 with Simulator:

2Select the feature you want to reconfigure

You may:

- Specify whether the watchdog function is hardware- or software-driven.
- Select a new clock frequency for the microcontroller.

Click **OK** to confirm.

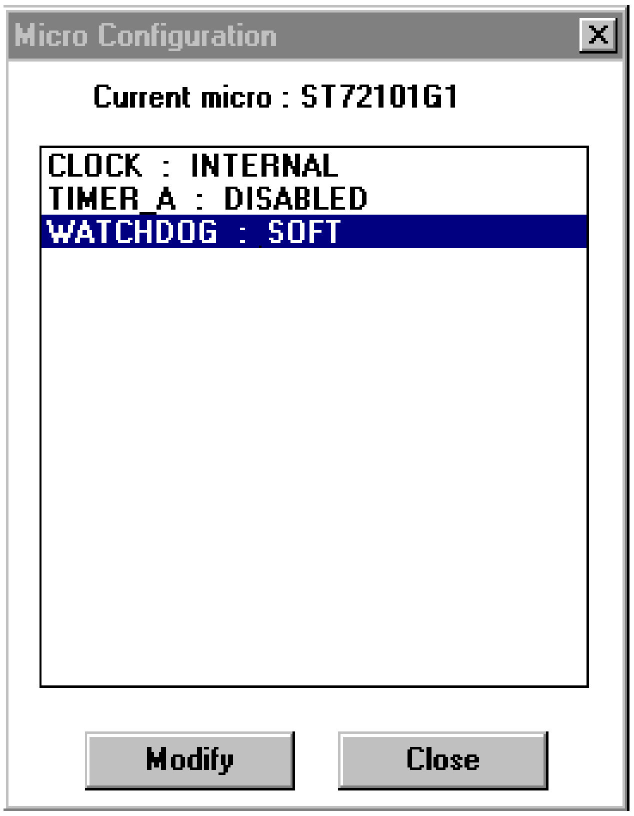

Micro Configuration

Watch Dog : [✓ Software] Hardware

Frequency in kHz: 8000

WGDB7 with Emulator or Development Kit:

2 Select the feature you want to reconfigure from the list, then click the **Modify** button.


Micro Configuration

Current micro : ST72101G1

```
CLOCK : INTERNAL
TIMER_A : DISABLED
WATCHDOG : SOFT
```

New dialog boxes appear for you to select a new setting for the selected feature.

The lists of the features that can be configured depends on the hardware in use with the debugger (emulator or development kit) and on the microcontroller currently selected (via the **Micro Name** dialog box).

For more information on available features, refer to the appropriate emulator or development kit user manual.

When you close an application, the micro definition settings are stored in the file **<application>.CF7**. The settings stored in this file are automatically reinstated when you load the application.

4.5 Viewing ST7 Resources

Once you have started the debugger, even if you have not yet loaded a program, you can view the contents of the ST7 memory and registers.

The information you can view includes:

- Memory contents
- Register contents
- Disassembled program code that is currently loaded in the ST7 emulator/simulator memory

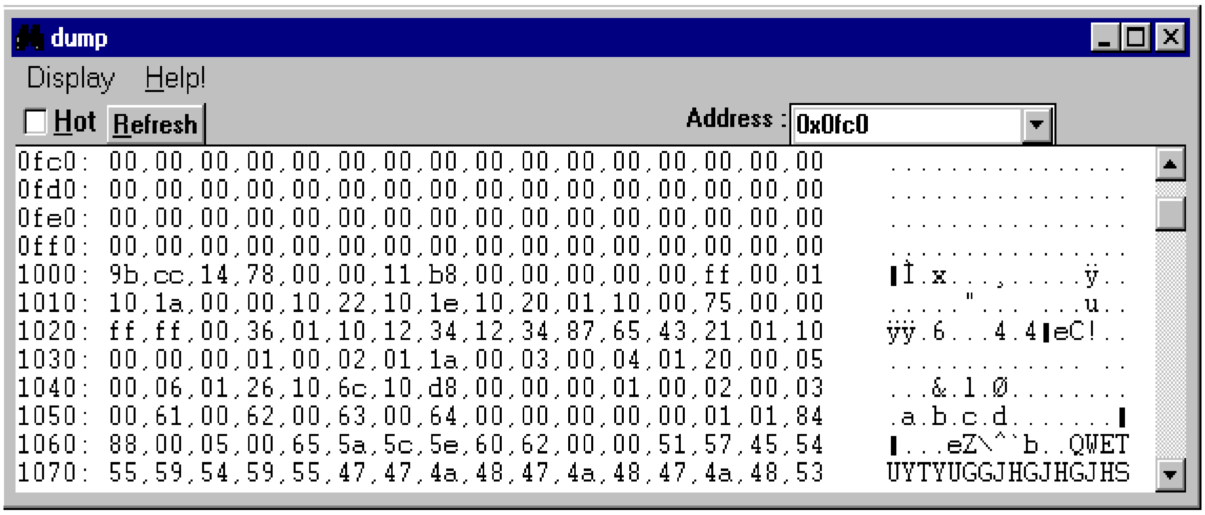
4.5.1 Viewing ST7 Memory Contents

To view the contents of the ST7 memory:

1 On the **Windows** menu, click **Dump**.

2 The Dump window now opens, displaying the contents of the ST7 memory:

Note that before you have loaded a program, the ST7 program memory will be set to 0.



Address : 0x0fc0

```
0fc0:  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00    ................
0fd0:  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00    ................
0fe0:  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00    ................
0ff0:  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00    ................
1000:  9b,cc,14,78,00,00,11,b8,00,00,00,00,00,ff,00,01    ▌Ì.x...¸.....ÿ..
1010:  10,1a,00,00,10,22,10,1e,10,20,01,10,00,75,00,00    ....."... ...u..
1020:  ff,ff,00,36,01,10,12,34,12,34,87,65,43,21,01,10    ÿÿ.6...4.4▌eC!..
1030:  00,00,00,01,00,02,01,1a,00,03,00,04,01,20,00,05    ............ ..
1040:  00,06,01,26,10,6c,10,d8,00,00,00,01,00,02,00,03    ...&.l.Ø........
1050:  00,61,00,62,00,63,00,64,00,00,00,00,00,01,01,84    .a.b.c.d.......▌
1060:  88,00,05,00,65,5a,5c,5e,60,62,00,00,51,57,45,54    ▌...eZ\^`b..QWET
1070:  55,59,54,59,55,47,47,4a,48,47,4a,48,47,4a,48,53    UYTYUGGJHGJHGJHS
```

From the **Dump** window, you can perform the following operations:

To do this:	Do this:
Display the memory contents in a different format	Choose another format from the **Display** menu.
Make the window Hot (so its contents are updated each time program execution is stopped)	Click the **Hot** checkbox (the window background becomes yellow indicating that the window is hot).
Display contents starting from a selected address	Enter the address you want to view in the **Address** field, then press the Enter key.
Display contents starting from a symbol name	Enter the symbol name or any valid C expression in the **Address** field.

NOTE: Setting this window as Hot can reduce the execution speed of WGDB7. You can save time by not setting this screen as hot, and updating the display when required using the **Refresh** button.

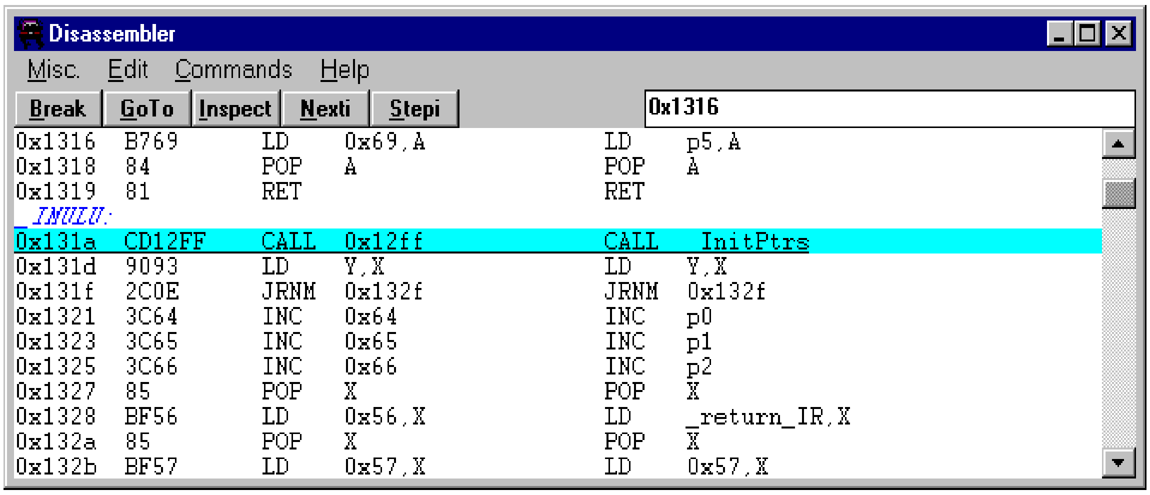
4.5.2 Viewing Disassembled Program Code

To view the disassembled program code that is loaded in memory:

1 On the **Windows** menu, click **Disassembler**.

2 The Disassembler window opens:

The Disassembler window shows each line of source code followed by its disassembled code.



```
0x1316
```

```
0x1316  B769      LD    0x69,A          LD    p5,A
0x1318  84        POP   A               POP   A
0x1319  81        RET                   RET
_INULU:
0x131a  CD12FF    CALL  0x12ff          CALL   InitPtrs
0x131d  9093      LD    Y,X             LD    Y,X
0x131f  2C0E      JRNM  0x132f          JRNM  0x132f
0x1321  3C64      INC   0x64            INC   p0
0x1323  3C65      INC   0x65            INC   p1
0x1325  3C66      INC   0x66            INC   p2
0x1327  85        POP   X               POP   X
0x1328  BF56      LD    0x56,X          LD    _return_IR,X
0x132a  85        POP   X               POP   X
0x132b  BF57      LD    0x57,X          LD    0x57,X
```

From the Disassembler window, you can perform the following operations:

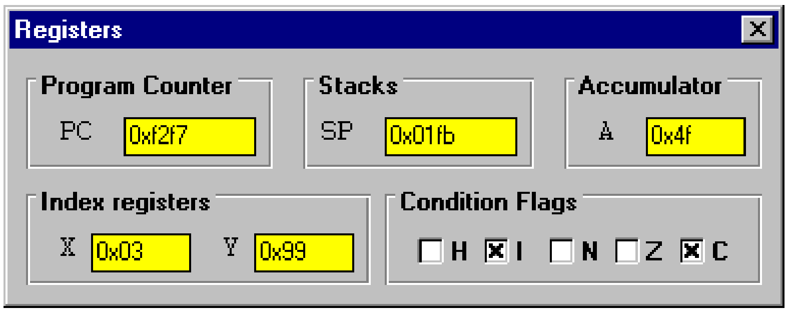
To do this:	Do this:
Set or cancel a breakpoint on a line	Select the line then click the **Break** button.
Display the memory contents in a different format	Choose another format from the **Options** command in the **Misc.** menu.
View more detailed information about a selected symbol	Select the symbol then click the **Inspect** button.
Display the contents starting from a selected address	Enter a symbol name or any valid C expression in the **Address** field, and press the Enter key on your keyboard.
Insert assembler instructions into emulated/ simulated ST7 memory	Select the **Online Assembler** option in the **Commands** menu.
Run the program with start/stop conditions	Select **Continue**, **Run**, **Reset**, **Stepi**, **Nexti**, **Go to Line**, **Finish Subroutine** options in the **Commands** menu.

4.5.3 Viewing Register Contents

You can view and modify register contents using WGDB7:

1 On the **Windows** menu, click **Registers**.

2 The Registers window now opens:



PC 0xf2f7
SP 0x01fb
A 0x4f
X 0x03
Y 0x99

3 To modify the value held in a displayed register, overtype the displayed value then press the **Enter** key.

The table below describes the fields in the Registers window:

This field:	Displays this:
Program Counter	The Program Counter (PC) register value. This is a 16-bit register that stores the address of the next instruction to be executed.
Stacks	The Stack Pointer (SP) value. This value points to the next free stack location.
Accumulator	The Accumulator (A). This is an 8-bit general purpose register used to hold operands and the results of arithmetic and logic calculations as well as data manipulations.
Index Registers	The Indirect Registers (X and Y) are used during register-indirect addressing mode as pointers to memory locations in the memory space.
Flags	H is the Half Carry Flag. The H bit is set when a carry occurs between the bit 3 and 4 of the ALU during an ADD or ADC instruction. The H bit is useful in BCD arithmetic subroutines. I is the Interrupt Mode Flag, which indicate whether or not the ST7 is in interrupt mode. N is the Negative Flag. When set, this bit indicates that the result of the last arithmetic, logical or data manipulation is negative (i.e. the most significant bit is set). The zero (Z) flag is set when the result of the last arithmetic or logic operation is zero, otherwise it is cleared. The carry (C) flag is set when a carry or borrow occurs during arithmetic operations, otherwise it is cleared.

4.6 Loading a Program

When you load a program, you must specify the **.s19** file (ST7 assembly chain executable) or **.abs** file (Hiware tool chain executable) of the program to be loaded. The symbols are automatically loaded from the appropriate symbol file. For details on the file types that you can debug, see “Preparing Programs for Debugging” on page 17. You may also load such files as:

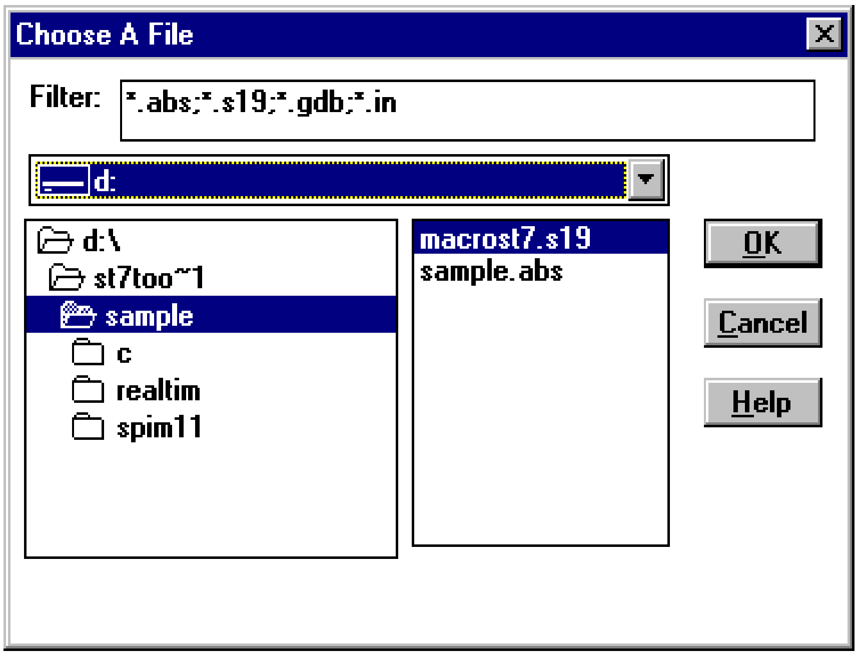
.in files, for pin input simulation files (See “Pin Input/Output Simulation” on page 53),

.gdb files, for GDB command files (See “Using GDB7 Commands” on page 66).

1 On the **File** menu, click **Open**.

The **Choose a File** dialog box opens:

2 In the drive list, click the drive that contains the program.



Filter: *.abs;*.s19;*.gdb;*.in

d:

d:\
st7too~1
sample
c
realtim
spim11

macrost7.s19
sample.abs

3 In the box beneath the drive list, double-click the name of the folder that contains the program. Continue double-clicking subfolders until you open the subfolder that contains the program.

4 In the list of files, click the program name.

5 Click **OK**.

When you load a program, a reset is implicitly performed. To open the Disassembler or module window at the current PC position, on the **Command** menu: click **Display PC** or click the **Reset** button.

TIP: To open a program you've used recently, click its name at the bottom of the **File** menu.

4.7 Viewing Program Information

4.7.1 Viewing a Source Module

To view a source module:

1 Open the **Sources** menu.

2 Either:

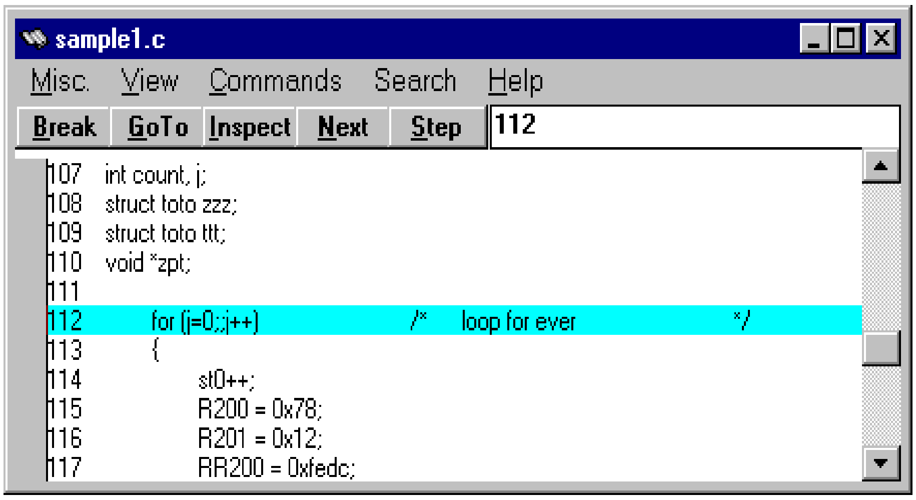
Click **Modules List...** then click the module you want to open.

or

Click the module you want to open from the list.

The module window opens displaying the module you just opened:

From the module window, you can perform the following operations:



```
107  int count, j;
108  struct toto zzz;
109  struct toto ttt;
110  void *zpt;
111
112      for (j=0;;j++)          /*     loop for ever              */
113      {
114          st0++;
115          R200 = 0x78;
116          R201 = 0x12;
117          RR200 = 0xfedc;
```

To do this:	Do this:
View further details about any item displayed in the window.	Click the item then click the **Inspect** button.
View a line of code disassembled.	Click the line number, then click the **Inspect** button.
Step or go to another part of the loaded program.	Click the appropriate button: **Next**, **Step** or **Goto**.
Find a character string within the module.	Click **Find** on the **Search** menu.
Insert a software breakpoint.	Click where you want to insert the breakpoint, then click the **Break** button, or click on the break bar to the left of the window next to the line on which you want to set the breakpoint.

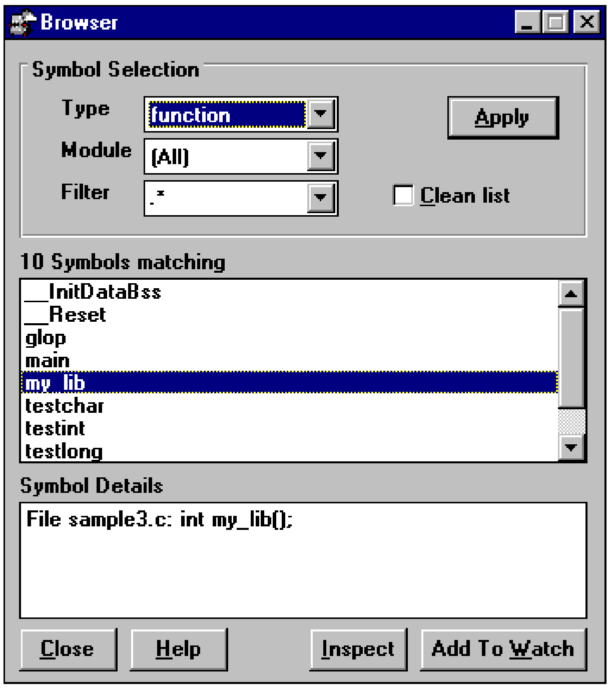
4.7.2 Finding and Viewing Symbols

To find symbols and view detailed information about them:

1 On the Windows menu, click **Browser**.

The Browser window opens:

2 In the Symbol Selection box:



File sample3.c: int my_lib();

i Select the symbol type you want to find from the Type drop-down list.

ii Select the module that contains the symbol you want to find from the Module drop-down list.

iii Check the Clean list checkbox if you want to prevent found symbols from appearing more than once if they are included in more than one module.

iv Enter the search operators in the Filter field:

To find:	**Use this operator:**	**Examples:**
All names	.*	
All the names containing a substring	The substring	“at” finds “data”, “date” and “bat”.

To find:	Use this operator:	Examples:
All the names starting with a substring	^ the substring	“^ba” finds “batch” and “back”
All the names ending with a substring	the substring $	“de$” finds “node” and “side”

v Click the **Apply** button.

6 The Symbols matching box lists the symbols found using the selection criteria entered in the Symbol Selection box.

7 You can now view further details about the found symbol by:

i Clicking the symbol and viewing the Symbol details box.

ii Clicking the symbol then clicking the Inspect button. If you selected a function, the function body is displayed in either the Disassembler window (see “Viewing Disassembled Program Code” on page 27) or the Module window (see “Viewing a Source Module” on page 30), depending on the available source file types. If you selected data or a constant, its value and type are displayed in the Inspect window.

iii Clicking the symbol then clicking the Add To Watch button to see its type and value in the Watch window (see below).

4.7.3 Watching Variable or Expression Values

WGDB7 enables you to watch the values of variables or expressions, which are updated each time program execution is suspended (for example, after a next or step instruction). To view a variable or expression value in the Watch window:

1 Either:

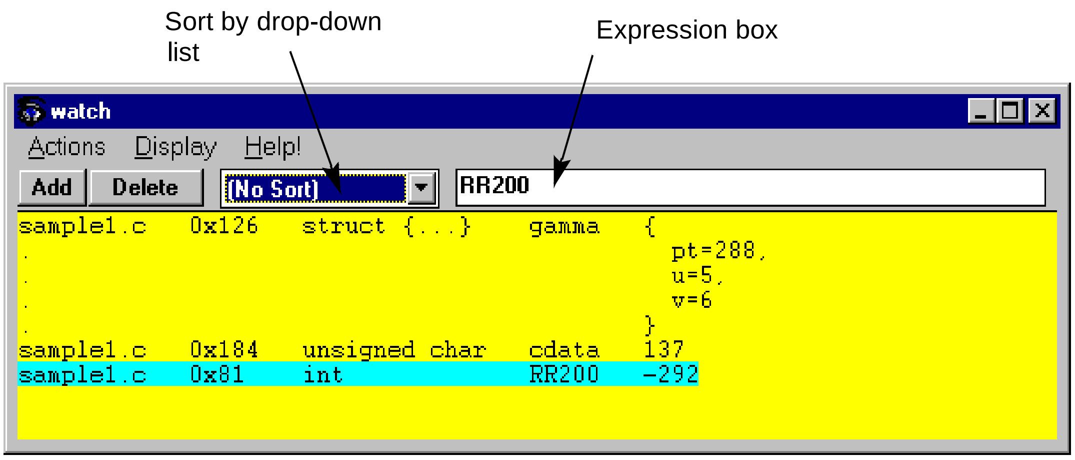
Select the variable whose value you want to watch in the Browser window, then click the **Add to Watch** button.

or:

On the Windows menu, click **Watch**, then type the name of the expression you want to watch in the expression box then press **Enter**.

The Watch window now opens, displaying the value of the selected variable:

From the Watch window, you can perform the following operations:



```
sample1.c   0x126   struct {...}    gamma   {
.                                             pt=288,
.                                             u=5,
.                                             v=6
.                                           }
sample1.c   0x184   unsigned char   cdata   137
sample1.c   0x81    int             RR200   -292
```

To do this:	Do this:
Change the format in which newly displayed symbols are displayed.	Choose the appropriate option from the **Preferences** submenu on the **Display** menu.
Choose the information you want to display (address, module, type).	Choose the appropriate option from the **Format** submenu on the **Display** menu.
Change the base in which the value of the selected symbol is displayed.	Choose the appropriate option from the **Base** submenu on the **Display** menu.
Choose the symbol type that data is sorted by.	Select the symbol type from the **Sort by** drop-down list.
View the value of an expression.	Enter an expression in the expression field.
View a range of *n* bytes starting from a variable or an address.	Enter the variable name or the address, preceded by a * and followed by @*n* in the expression box. For example, to view the 5 bytes starting at address 0x4003, enter: *0x4003@5.
View a subrange of *n* values of an array.	Enter * followed by the array or pointer name followed by @*n* in the expression field.
View a selected function disassembled.	Select the function, then click **Disassembler** on the **Actions** menu.
View and modify the data held in memory for a selected symbol.	Select the symbol, then click **Dump** on the **Actions** menu.
View a value of the selected symbol in the Inspect window.	Select the symbol, then click **Inspect** on the **Actions** menu.
View a selected function within the source code.	Select the symbol, then click **Sources** on the **Actions** menu.
Change the value of a symbol.	Click on the value and overtype it.

Add a symbol or expression to the Watch window, so it can be viewed along with other symbols and expressions as their values progress.	Click the **Add** button. A secondary window opens where you can select a symbol or an expression. Confirm by clicking OK in this new window.
Remove a line from the Watch list	Select the line and click the **Delete** button.

4.8 Executing Loaded Programs

WGDB7 includes the following program execution commands, which are available either as buttons in the main window or other windows (where appropriate) or on the **Commands** menu.

The following table explains what each command does:

This command:	Does this:
Run	Resets the emulator, and runs the program from the reset starting point until a breakpoint is reached.
Cont	Continues running the program from the current PC until a breakpoint is reached.
Next	Executes the source line at the current PC location, stepping over any function calls.
Step	Executes the source line at the current PC location, stepping into any function calls.
Finish	Runs the program from the current PC line until the end of the current function is reached, then returns to the calling procedure.
Stop	Stops the program running.
Reset	Resets the ST7 emulator. The PC is set to the reset vector value and A, X, and Y are set to 0 (see “Viewing Register Contents” on page 28 for further details about these values).
Goto	Runs the program from the current PC position to the specified marker position.
Stepi	Steps one instruction immediately following the PC line, stepping inside any call instructions.
Nexti	Steps one instruction immediately following the PC line, executing any call instructions.

TIP: To execute a program from a specific address, set the PC value to where you want to start running the program in the Register window (see “Viewing Register Contents” on page 28), then click the **Cont** button.

4.9 Using Software Breakpoints

4.9.1 Setting Software Breakpoints

Software breakpoints are breakpoints that are triggered when a source line or a disassembled instruction is executed.

You set software breakpoints in either the module window or the Disassembler window.

To open the module window, click the **Source** menu, then click the name of the source module in which you want to set a breakpoint. To open the Disassembler window, on the **Windows** menu, click **Disassembler**.

In the module or disassembler window:

either:

1 Click the line of code on which you want to set a breakpoint.

2 Click the break button.

or:

Click on the break bar on the left side of the window next to the line on which you want to set the breakpoint.

A breakpoint is now set on the selected line. This line is highlighted in bold (by default), indicating that it includes a breakpoint.

NOTE: If you try to place a breakpoint on an inappropriate line, such as on a While condition or a comment, WGDB7 will set the breakpoint on the next available line.

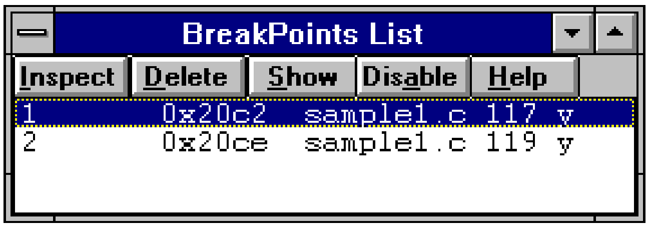
4.9.2 Managing Software Breakpoints

You can perform all software breakpoint management tasks in the BreakPoints List window. Note that this is only available when at least one software breakpoint is set.

To open the BreakPoints List window, on the **Windows** menu click **Soft Breakpoints.**

The BreakPoints List window is as follows:

. From the BreakPoints List window, you can perform the following operations:



```
1         0x20c2   sample1.c 117 y
2         0x20ce   sample1.c 119 y
```

To do this:	Do this:
Set specific software conditions under which a breakpoint is triggered.	Select the breakpoint, then click the **Inspect** button.
Delete a breakpoint	Select the breakpoint, then click the **Delete** button.
Display the selected breakpoint location within the source code.	Select the breakpoint, then click the **Show** button.
Enable/disable the selected breakpoint.	Select the breakpoint, then click the **Enable/ Disable** button.

4.10 Using Hardware Breakpoints

4.10.1 Hardware and Advanced Breakpoints

As a general rule, hardware breakpoints stop program execution when a specific hardware event occurs. WGDB7 provides you with various features that enable you to set hardware breakpoints that cover a large number of situations.

These breakpoints fall into two main categories referred to in the following as:

Hardware breakpoints and ***Advanced breakpoints***.

Hardware breakpoints stop program execution when any of the following events occur:

- A variable or constant is accessed,
- A data memory address or range of addresses are accessed,
- A program memory address or a range of addresses are accessed,
- An opcode fetch occurs.

Advanced breakpoints stop program execution when any of the following events occur:

- A stack overflow occurred,
- A stack underflow occurred,
- An external signal has been triggered in (Development Kit only).

4.10.2 Setting Hardware Breakpoints

To set a hardware breakpoint:

1 In the Windows menu, click **Hardware Events**

A cascading menu opens where you can select two other options:

- **Hardware Breakpoints**
- **Output Triggers** (option available only for the Emulator and the Development Kit, see "Working with Output Triggers" on page 43).

2 Click **Hardware Breakpoints**

A **Hardware Breakpoints** window opens showing options that depends on the type of device WGDB7 is associated with: simulator, emulator, or development kit.

This dialog box appears:

The upper box "**Current Settings**" reminds you which Advanced Breakpoint options are set.

To change them, click the **Configure...** button.

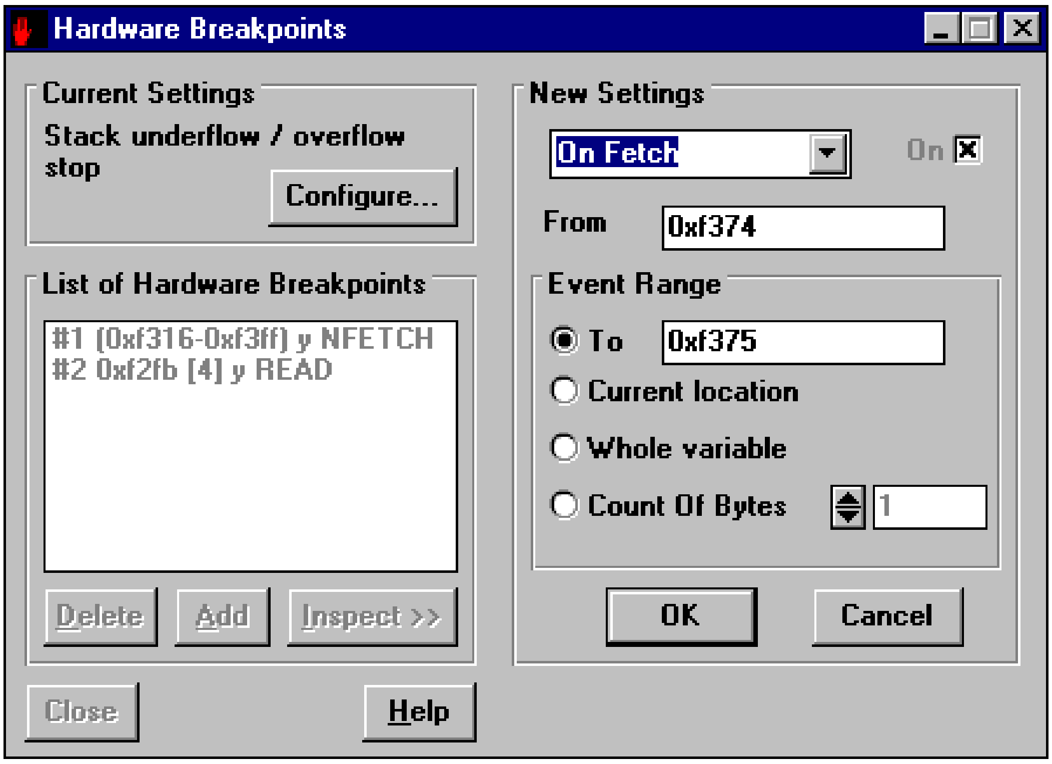
For more information refer to "Setting Advanced Breakpoints" on page 40.

NOTE: The Advanced Breakpoints options are not available with the emulator.

The central box shows a list of hardware breakpoints already set.

3 Click the **Add** button.

The **Hardware Breakpoint** window extends, letting you access new setting buttons and boxes:



Stack underflow / overflow stop

```
#1 (0xf316-0xf3ff) y NFETCH
#2 0xf2fb [4] y READ
```

On Fetch   On [✓]

From 0xf374

[✓ To] 0xf375

Current location

Whole variable

Count Of Bytes 1

4 In the New Settings box, select an option according to the table below:

To stop program execution if:	Select this:
The address being accessed is within the hardware breakpoint range. The address must be a data address.	Read/Write (Not On Fetch)
The address of the instruction to be executed is within the hardware breakpoint range.	On Fetch
The specified area is accessed via a READ operation	Read (Not available with the emulator)
The specified area is accessed via a WRITE operation	Write (Not available with the emulator)

5 In the **From** field, enter the start address of the range for which the breakpoint is activated. This can be expressed either as a variable (symbolic name) or as an address.

6 In the **Event range** box, choose the range on which the breakpoint is triggered:

Select **To** to enter the end address of the memory range on which the breakpoint is set. This can be expressed either as a variable (symbolic name) or as an address.

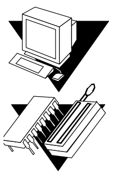
Select **Current location** to activate the breakpoint only if the address defined in the from field is accessed (range = 1 byte).

Select **Whole variable** to activate the breakpoint on the whole variable range which is directly linked to the symbol type specified in the **From** field (for example on whole fields of data symbols for a C struct data type).

Select **Count Of Bytes** to define the number of bytes from the location specified in the From field for which the breakpoint is activated.

7 Click the **OK** button to set the breakpoint.

4.10.3 Setting Advanced Breakpoints



WGDB7 with Simulator or Development Kit Only

To set an advanced breakpoint:

1 In the Windows menu, click **Hardware Events**.

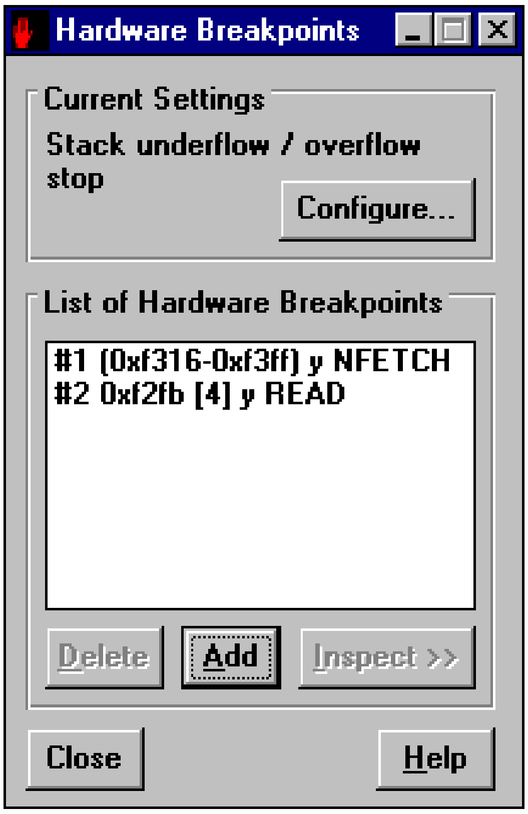
A cascading menu opens where you can select two other options:

- **Hardware Breakpoints**
- **Output Triggers** (option available only for emulator and development kit, see “Working with Output Triggers” on page 43).

2 Click **Hardware Breakpoints**.

A **Hardware Breakpoints** window opens showing options that depends on the type of device WGDB7 is associated with: simulator, emulator, or development kit.

This dialog box appears:



Stack underflow / overflow stop

#1 [0xf316-0xf3ff] y NFETCH
#2 0xf2fb [4] y READ

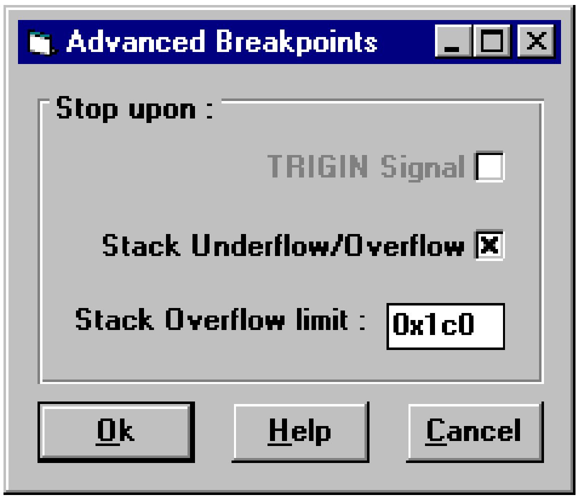
The upper box **Current Settings** reminds you which Advanced Breakpoint options are set.

To change them, click the **Configure** button.

A new dialog box is displayed for you to select new breakpoint specifications.

You may specify any of these conditions:

- Stop upon occurrence of an input signal (TRIGIN Signal option),
- Stop upon stack underflow,



TRIGIN Signal ☐

Stack Underflow/Overflow ☒

Stack Overflow limit : 0x1c0

- Stop upon stack overflow,
- Stop upon stack overflow down to a specified value of the stack pointer.

By default, the limit is the lower address of the stack area for the current microcontroller.

Note that the **TRIGIN Signal** option is available only if you run WGDB7 with the Development Kit

(the option is greyed with the simulator).

Select an option according to the following table:

To stop program execution if:	Do this:
An external input signal is triggered on the evaluation board TRIGIN pin.	Check TRIGIN Signal box.
A stack underflow occurs.	Check Stack Underflow/Overflow box.
A stack overflow occurs.	Check Stack Underflow/Overflow box.
The stack pointer (SP) reaches a specified limit toward lower values.	Check Stack Underflow/Overflow box, and specify an SP value in the Stack Overflow Limit edit box.

3 Click the **OK** button to set the breakpoint.

The information in the Current Settings box is updated according to the new Advanced Breakpoint settings you specified.

4.10.4 Managing Hardware Breakpoints

You can perform all Hardware Breakpoint management tasks from the Hardware Breakpoints window (refer to “Setting Hardware Breakpoints” on page 38). The table below lists the Hardware Breakpoint management tasks you can perform and explains how to perform them:

To do this:	Do this:
Enable/disable a breakpoint.	Select the breakpoint from the **List of Breakpoints**, click the **Inspect** button, then click the **On** check box in the **New Settings** box. When the **On** box is checked, the breakpoint is enabled.
Delete a breakpoint.	Select the breakpoint from the **List of Breakpoints**, then click the **Delete** button.

4.11 Working with Output Triggers

4.11.1 Overview

Triggers are output signals that can be connected to an external resource from the Emulator or Development Kit. These signals can be used to synchronize an external measurement instrument, such as an oscilloscope. When the defined set of conditions is met, an impulse (TTL level) is emitted or the level of a signal is changed at the specified outlet:

- On TRIGGER OUT 1 or 2 outlets on the emulator front panel, where a one-clock cycle impulse is emitted.
- On a special outlet (TRIGOUT pin) on the evaluation board for the Development Kit. A special dialog box lets you choose the waveform mode for the external signal (see “Setting the TRIGOUT Mode” on page 46).

Associated events are referred to below as **Trigger events**.

You can set the trigger events so that the signals are sent under the following circumstances:

- When a variable or constant is accessed.
- When a data memory address or range of addresses are accessed.
- When a program memory address or a range of addresses are accessed or its contents are executed.
- When an opcode fetch occurs.

4.11.2 Setting Trigger Events

To set a trigger event**:**

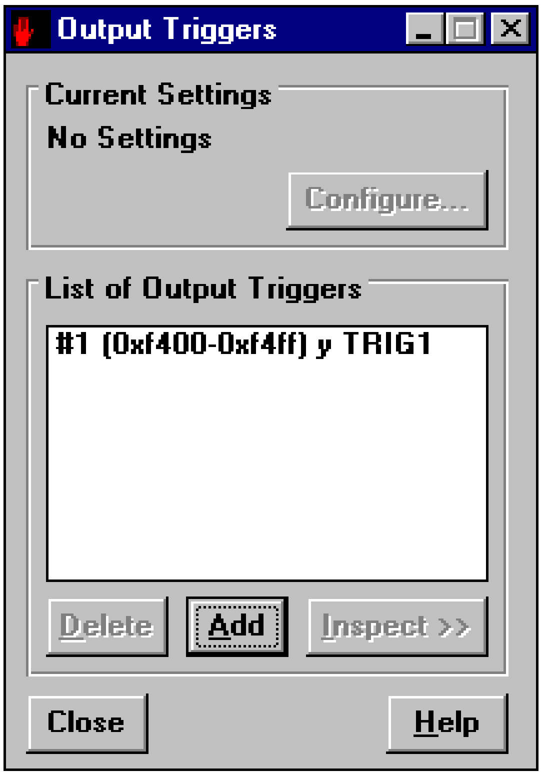
1 In the Windows menu, click **Hardware Events**.

A cascading menu opens where you can select two other options:

- **Hardware Breakpoints**
- **Output Triggers**

2 Click **Output Triggers**.

This dialog box appears:



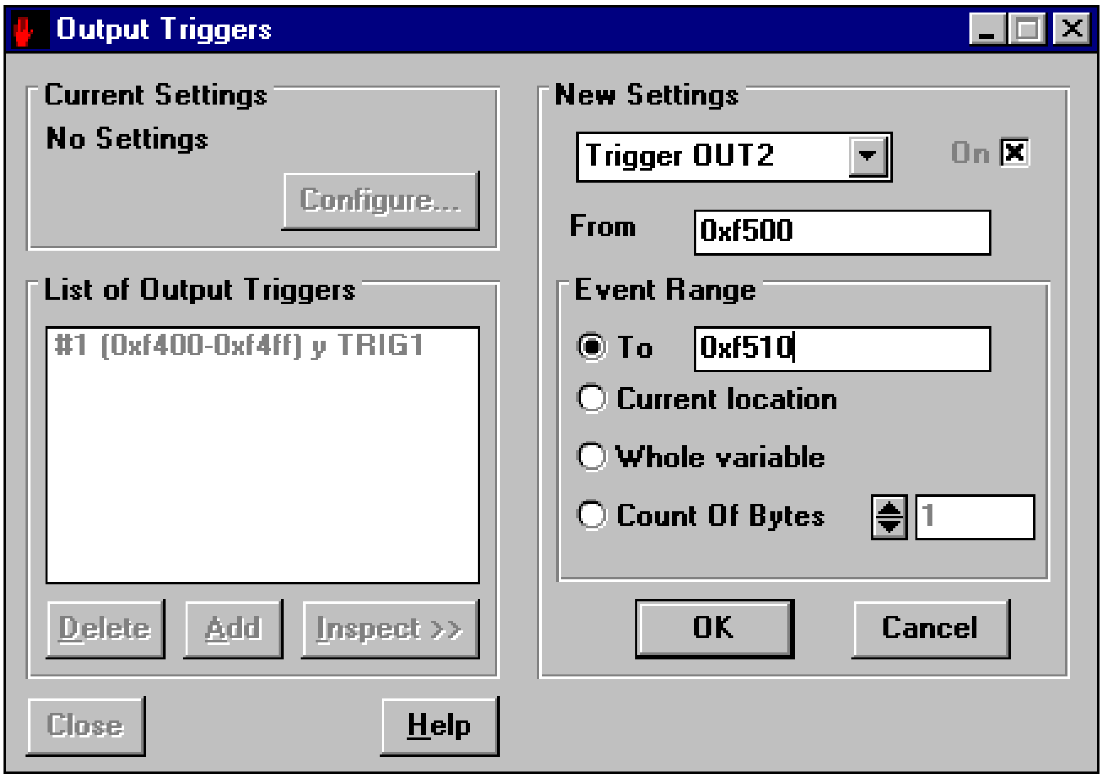
#1 (0xf400-0xf4ff) y TRIG1

The upper box “Current Settings” shows no advanced settings for output triggers. This feature is not available for WGDB7 running with an emulator.

The central box shows a list of Output Triggers already set.

3 Click the **Add** button.

The **Output Triggers** window extends, letting you access new setting buttons and boxes:



4 In the **New Settings** box, select a trigger event type:

For the ***Emulator***, select **Trigger OUT1** to output the signal on Trigger 1, or **Trigger OUT2** to output the signal on Trigger 2.

For the ***Development Kit***, select **Force High** to generate a high-level signal on the TRIG-OUT pin, or **Force Low** to generate a low-level signal on the TRIGOUT pin.

5 In the **From** field, enter the start address of the memory range which, when accessed, will cause a signal to be output. This can be expressed either as a variable (symbolic name) or as an address.

6 In the **Event range** box, choose the range which, when accessed, will cause a signal to be output:

Select **To** to enter the end address of the memory range which, when accessed, will cause a signal to be output. This can be expressed either as a variable (symbolic name) or as an address.

Select **Current location** to send a signal only if the address defined in the from field is accessed (range = 1 byte).

Select **Whole variable** to send a signal on if the whole variable range which is directly linked to the symbol type specified in the **From** field (for example on whole fields of data symbols for a C struct data type) is accessed.

Select **Count Of Bytes** to define the number of bytes from the location specified in the From field which will cause a signal to be output when accessed.

7 Click the **OK** button.

The trigger event is now set.

4.11.3 Setting the TRIGOUT Mode

WGDB7 with Development Kit Only

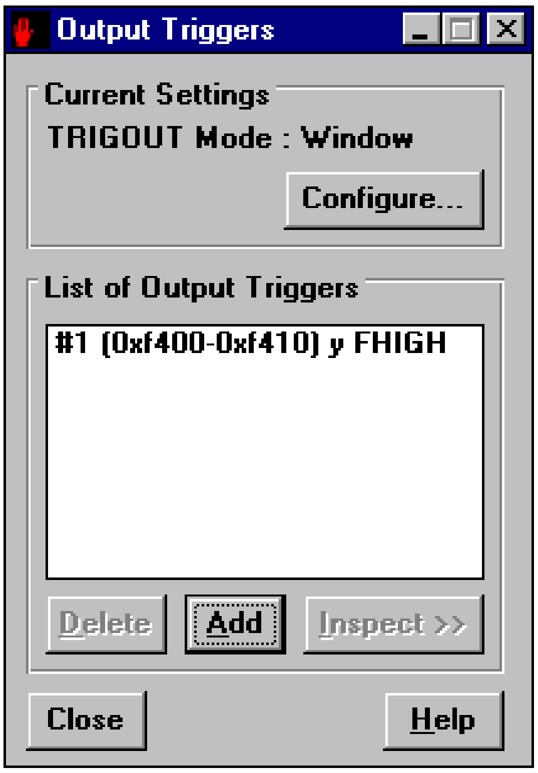
To choose the waveform mode for the external TRIGOUT signal:

1 In the Windows menu, click **Hardware Events**.

A cascading menu opens where you can select two other options:

- **Hardware Breakpoints** Output Triggers

Click **Output Triggers**



TRIGOUT Mode : Window

#1 [0xf400-0xf410] y FHIGH

This dialog box appears:

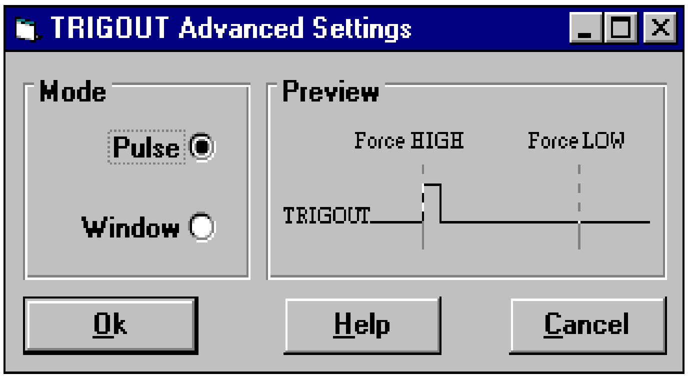
The upper box **Current Settings** reminds you which TRIGOUT mode is set.

To change it, click the **Configure** button.

A new dialog box is displayed for you to select new TRIGOUT mode specifications.

You may specify any of these modes:

- **Pulse Mode**
- **Windows Mode**



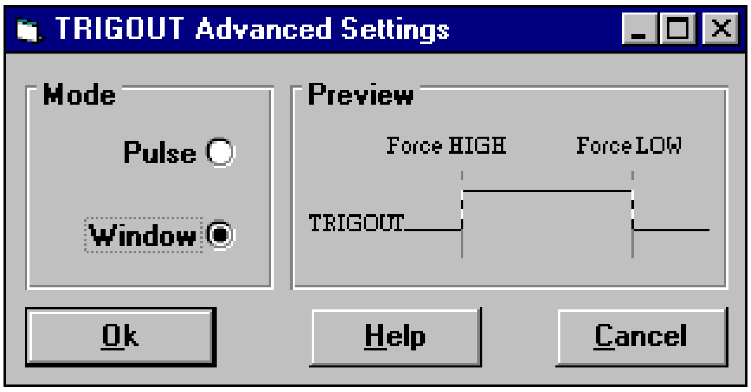
Mode: [✓ Pulse] Window

In ***Pulse Mode***, only Force High events cause a signal to be generated on the TRIGOUT pin. In this case, the signal is a one clock cycle impulse as shown in the picture opposite.

.

In ***Windows Mode***, the signal generated by a Force High event is a level transition from LOW to HIGH.

Similarly, the signal generated by a Force Low event is a level transition from HIGH to LOW.



Mode: Pulse [✓ Window]

For example, setting the entry in a function as a Force High event, and the exit from the same function as a Force Low event would enable you to control the full execution of the function on the TRIGOUT pin.

4.11.4 Managing Trigger Event Definitions

You can perform all trigger event definition management tasks from the Hardware Breakpoints window (refer to “Setting Hardware Breakpoints” on page 38). The table below lists the trigger definition management tasks you can perform and explains how to perform them:

To do this:	Do this:
Enable/disable a trigger event definition.	Select the trigger event definition from the **List of Output Triggers**, click the **Inspect** button, then click the **On** check box in the **New Settings** box. When the **On** box is checked, the trigger definition is enabled.
Delete a trigger event definition.	Select the trigger event definition from the **List of Output Triggers**, then click the **Delete** button.

4.12 Working with the Trace Buffer

WGDB7 with Emulator Only

The emulator trace buffer records hardware events that occur when a program is executed. WGDB7 enables you to view either all the trace buffer contents or filter those that you view. You can also use the logic analyser to define up to three areas of memory, which when accessed either:

- Stop trace buffer recording after a specified number of cycles.

  or

- Cause each access made to the specified area to be recorded a specified number of times.

Each defined area of memory is called an **event**.

WGDB7 enables you to define up to three events, Event 1, Event 2 and Event 3.

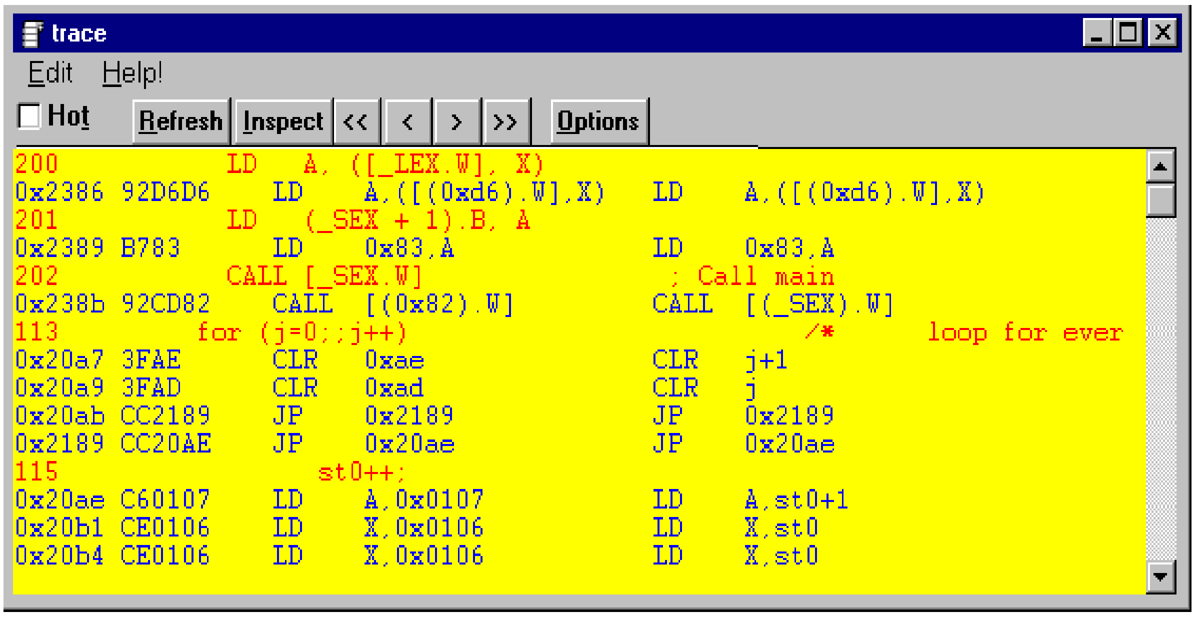
Event 1 and **Event 3** stop trace buffer recording after a specified number of cycles.

Event 2 causes each access made to the specified area to be recorded a specified number of times.

4.12.1 Viewing Trace Buffer Contents

1 On the Windows menu, click **Trace**.

2 The Trace window opens, displaying the events recorded in the emulator hardware trace buffer. Source code is displayed in red, assembler code in blue, and all other lines are displayed in black:



```
200              LD   A, ([_LEX.W], X)
0x2386 92D6D6    LD     A,([(0xd6).W],X)    LD     A,([(0xd6).W],X)
201              LD    (_SEX + 1).B, A
0x2389 B783      LD     0x83,A              LD     0x83,A
202              CALL [_SEX.W]               ; Call main
0x238b 92CD82    CALL   [(0x82).W]          CALL   [(_SEX).W]
113            for (j=0;;j++)                       /*      loop for ever
0x20a7 3FAE      CLR    0xae                CLR    j+1
0x20a9 3FAD      CLR    0xad                CLR    j
0x20ab CC2189    JP     0x2189              JP     0x2189
0x2189 CC20AE    JP     0x20ae              JP     0x20ae
115                     st0++;
0x20ae C60107    LD     A,0x0107            LD     A,st0+1
0x20b1 CE0106    LD     X,0x0106            LD     X,st0
0x20b4 CE0106    LD     X,0x0106            LD     X,st0
```

From the Trace window, you can perform the following operations:

To do this:	Do this:
Set the window as Hot, so that its contents are updated every time program execution is suspended.	Click the **Hot** check box. When the **Hot** box is checked, window is Hot.
Update the Trace window contents.	Click the **Refresh** button.
View the value of a constant or data.	Select the constant, variable, or register, then click the **Inspect** button.
Display the disassembled code of a selected address.	Select the address then click the **Inspect** button.
View the source code of a selected function.	Select the function, its address or line number, then click the **Inspect** button.

Browse through the flow of the traced code.	Click the '**<<**', '**>>**' buttons to browse through source code. Click the '**<**', '**>**' buttons to browse through assembler code.
Filter the displayed event types.	Click the **Options** button, then select the appropriate options from the **Disassembly options** in the Trace Options window.
View source code, opcodes and operand binary dump, hexadecimal operands of disassembled instructions with variable names each instruction cycle in the Trace window.	Click the **Options** button, then select the appropriate options from the **Disassembly options** in the Trace Options window.

NOTE: Setting this window as *Hot* reduces the WGDB7 performance for application load or save. To update the display and save time, don't use the Hot check box; use the **Refresh** button instead.

4.13 Managing Trace Buffer Recording Using the Logic Analyser

WGDB7 with Emulator Only

The logic analyser enables you to define up to three events, named ***Event 1***, ***Event 2***, and ***Event 3***.

Event 1 and Event 3 cause trace buffer recording to be stopped after a specified number of cycles. Event 2 causes each access made to a specified area associated with the event to be recorded a specified number of times. This enables you, for example, to count the occurrence of a complex event, or trace all accesses to an address or address range. If several event conditions are set together (for example, Event 1 and Event 2), recording for an event condition is only triggered when all previous event condition recordings are completed (for example, Event 2 condition recording only starts when Event 1 condition recording is completed). Trace buffer recording is always done in the following order: Event 1, Event 2 then Event 3.

You can use any combination of Event 1, Event 2 and Event 3 with the following two exceptions:

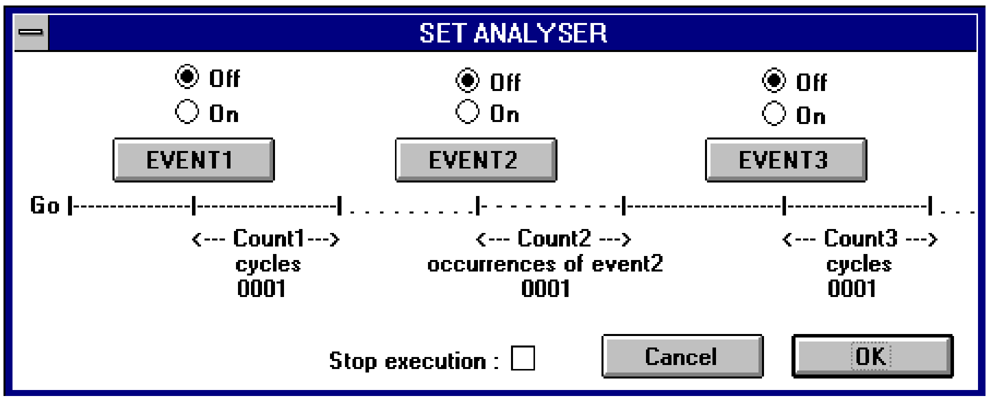
- Event 3 cannot be used alone, Event 1 should be used instead.

- Event 1 and Event 3 cannot be used together without Event 2, Event 1 should be used instead.

To define an event:

1 On the **Commands** menu, click **Set Analyser**.

The **Set Analyser** dialog box opens:


SET ANALYSER

EVENT1	EVENT2	EVENT3
[✓ Off] On	[✓ Off] On	[✓ Off] On
<--- Count1 ---> cycles 0001	<--- Count2 ---> occurrences of event2 0001	<--- Count3 ---> cycles 0001

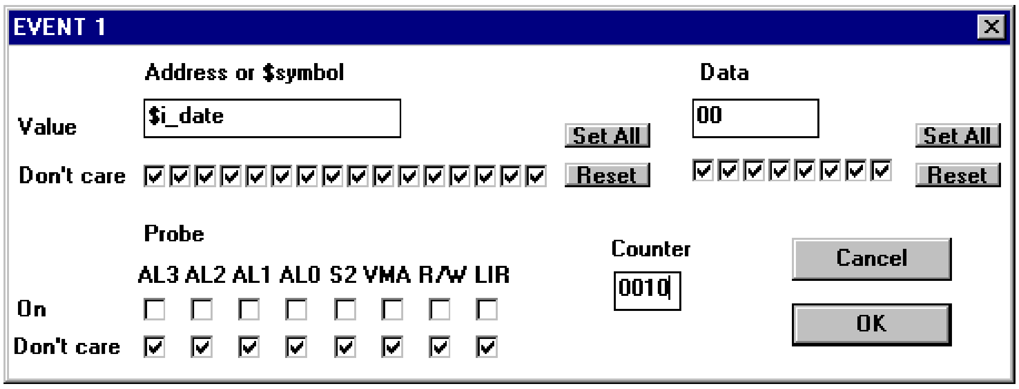
Go

Stop execution : ☐

2 Above the button associated with the event you want to define, select **On**. If you want to stop program execution when trace buffer recording is completed, check the **Stop execution** box.

3 Click the button of the event you want to define.

The **Event definition** dialog box opens:


EVENT 1

	Address or $symbol	Data
Value	$i_date	00
Don't care	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

Probe

	AL3	AL2	AL1	AL0	S2	VMA	R/W	LIR
On	☐	☐	☐	☐	☐	☐	☐	☐
Don't care	✓	✓	✓	✓	✓	✓	✓	✓

Counter: 0010

To set the event on an address or a symbol in the memory space, enter the address or the symbol name preceded by the **$** symbol in the **Address or $symbol Value** field.

Don’t care check boxes indicate individual bits to be ignored in the address. The leftmost bit is bit 15. To consider all bits, click the **Reset** button. By default, all **Don’t care** check boxes are checked, which means that the entire address will be ignored.

To set the event on a data value in the memory space, enter a data value in the **Data Value** field.

Again, **Don’t care** check boxes indicate individual data bits to be ignored. To consider all bits, click the **Reset** button. By default, all Don’t care check boxes are checked, which means that the value will be ignored.

To set the event on a signal value, check the appropriate check box for the signal whose value you want to consider, according to the following table:

Check this box:	**To consider this value:**
AL3 to AL0	The appropriate input probe signal (binary)
S2	Probe S2
VMA	Valid memory address
R/W	Read/write memory access
LIR	Load instruction register (fetch) memory access

If you want to set the event on any value, leave the Don't care check box checked for that probe. If you want to set the event on the input of a signal to the probe, uncheck the Don't care check box for that probe.

4 For **Event 1** and **Event 3**, by default, the trace records are recorded until the defined condition is met, then for an additional one cycle after that. To continue recording for more than one cycle, enter the number of cycles to be recorded after the event is met in the **Counter** field.

5 For **Event 2**, by default, only one occurrence of the event is recorded. To record more than one occurrence of Event 2, enter the number of cycles you want to record after the event is first met in the **Counter** field.

6 Click the **OK** button to implement the changes you made.

To begin trace recording use the **Run**, **Continue** or any other program execution commands. Recording continues until the program is stopped by a breakpoint, the logic analyser, or because the **Stop** button has been clicked. When the execution stops, the data in the Trace window is updated if it is ‘hot’, otherwise you must press the **Refresh** button to update it.

If you execute the program in step by step mode, no trace records are recorded.

4.14 Pin Input/Output Simulation

WGDB7 with Simulator Only

4.14.1 The Input-Process-Output Scheme

Simulating the execution of a program on an ST7 microcontroller would be quite frustrating if you were not able to input signals and see the resulting output. The behaviour of a program cannot be completely understood unless the occurrence of external events is considered. Input signals enter a black box (the simulated microcontroller) which in turn produces output signals after the program has been executed.

External information (pin stimulus) is conveyed through the simulated microcontroller pins, configured as I/O ports (or parts of I/O ports). Some pins can be configured as input or output and some are dedicated to only input or only output (see the ST7xxxx datasheets). The following is an example of the ST72251 microcontroller pin configuration:

Table 2. Example of Pin Configuration (ST72251 MCU)

Pin No. (SDIP32)	Pin Name	Type	Description
...	...	...	...
4	PB7/SS	I/O	Port B7 or SPI Slave Select
5	PB6/SCK	I/O	Port B6 or SPI Serial Clock
...	...	...	...
10	PB1/OCMP1_A	I/O	Port B1 or Timer A Output Compare 1
11	PB0/ICAP1_A	I/O	Port B1 or Timer A Input Capture 1

Table 2. Example of Pin Configuration (ST72251 MCU)

Pin No. (SDIP32)	Pin Name	Type	Description
12	PC5/EXTCLK_A/AIN5	I/O	Port C5 or Timer A Input Clock or ADC Analog Input 5
...	...	...	...

For more information on a specific ST7 MCU pin assignment, refer to the appropriate ***ST7xxxx Datasheet***.

4.14.2 How to Setup Pin Input Simulation

Pin Input Simulation File

To simulate pin input signals, the user must create a text file that includes pin input simulation information. This text file must be saved as:

```
<filename>.in
```

where **`<filename>`** is any name complying with the usual naming conventions for Windows 3.1, 95, or NT.

It is advisable to choose for this name the name of the application to be simulated and to place the file in the directory of the application.

Pin Input Simulation File Format

Three types of input signals can be simulated:

- Binary
- Analog
- Square wave cyclic

Each line in the file should follow the corresponding syntax (fields separated by one or more spaces):

Binary Signals

```
PIN <pin_name> -i <value> <start_time>
```

Where

`<pin_name>`

is the name of the pin. You can use any name (maximum 8 alphameric characters). It is recommended to use port names as specified in the corresponding datasheet.

Examples: **`PB7, PC0, PA1`**

`-i`

is the signal type for binary signals.

`<value>`

is a code for the signal at the time indicated next field. Specify 1 to simulate a rising edge.

Specify 0 to simulate a falling edge.

NOTE: At zero CPU ticks the signal is supposed to be low.

`<start_time`

is the time when the state transition should occur. Specify an absolute time in machine cycles (CPU ticks).

Analog Signals

```
PIN <pin_name> -a <value> <start_time>
```

Where

`<pin_name>`

is the name of the pin. You can use any name (maximum 8 alphameric characters). It is recommended to use port names as specified in the corresponding datasheet.

Examples: **`PB7, PC0, PA1`**

-a

is the signal type for analog signals.

<value>

is the value of the simulated analog input signal. Specify a value between 0 and 255.

<start_time>

Specify the absolute time in machine cycles (CPU ticks) when the simulated event (analog signal) should occur.

Square Wave Cyclic Signals

```
PIN <pin_name> -c <init_value> <start_time> <period>
```

Where

<pin_name>

is the name of the pin. You can use any name (maximum 8 alphameric characters). It is recommended to use port names as specified in the corresponding datasheet.

Examples: **PB7, PC0, PA1**

-c

is the signal type for square wave cyclic signals.

<init_value>

is the starting level of the cyclic signal. Specify 0= low, 1= high.

<start_time>

is the time the state transition should occur. Specify an absolute time in machine cycles (CPU ticks).

<period>

is the period of the signal in machine cycles (CPU ticks).

Example of Pin Input Simulation File

File Name: **`example.in`**

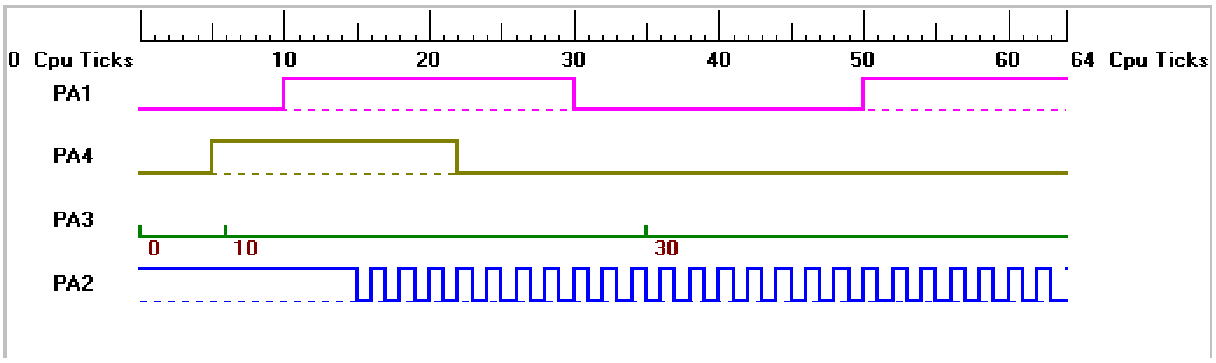
This text file would contain, for example, these lines:

```
PIN PA1 -i 0 0
PIN PA1 -i 1 10
PIN PA1 -i 0 30
PIN PA1 -i 1 50
PIN PA4 -i 0 0
PIN PA4 -i 1 5
PIN PA4 -i 0 22
PIN PA3 -a 10 6
PIN PA3 -a 30 35
PIN PA2 -c 1 15 1
```

The **`example.in`** file is just a line by line description of events occurring on the pins specified.

You may add or remove any lines in the file, thus modifying the script for the I/O simulation. The pin names do not have to be in sequential order within the Pin Input Simulation File.

Loading the Pin Input Simulation File into a Waveform Editor would produce the following diagram:

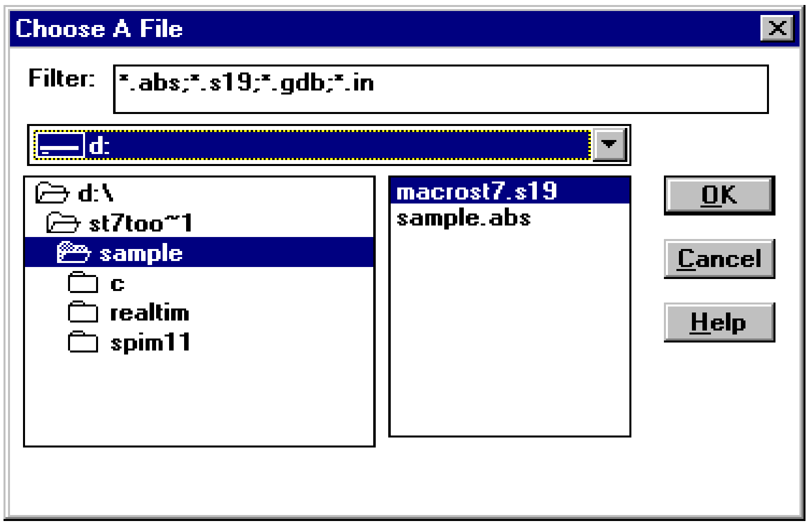


NOTE: The Pin Input Simulation File can also be created directly by the Waveform Editor. For more information refer to the on-line help messages of the Waveform Editor you are running.

4.14.3 Starting Pin Input Simulation

To start pin input simulation proceed as follows, assuming that you have already started the simulation session:

1 Start your application via the **Choose a File** dialog box:


Choose A File

Filter: *.abs;*.s19;*.gdb;*.in

d:

d:\
st7too~1
sample
c
realtim
spim11

macrost7.s19
sample.abs

2 Follow the instructions as indicated above in “Loading a Program” on page 29.

3 Using the same **Choose a File** dialog box load the Pin Input Simulation File (**`<filename>.in`** file) that contains the script you want to execute.

Specified values are automatically simulated as input on the specified pins when the application is started.

NOTE: The order in which you load the application and the pin input simulation file is irrelevant as long as you are in the WGDB7 simulator environment. You may load the **`<filename>.in`** file first as well.

4 Stop your application.

4.14.4 Pin Output Signals Generated by your Program

Naturally, as a result of the simulation process pin output signals are generated on output pins.

Their characteristics are systematically recorded in a new text file, the pin output file automatically saved and referred to as:

`<filename>.out`

where **`<filename>`** is the name of the corresponding pin input simulation file. This file is named **port.out** by default. It can be found in the same directory as the pin input simulation file.

The **`<filename>.out`** file has the same format as the **`<filename>.in`** file (with **`-o`** as the common type for all pin output signals), and may be edited by any text editor.

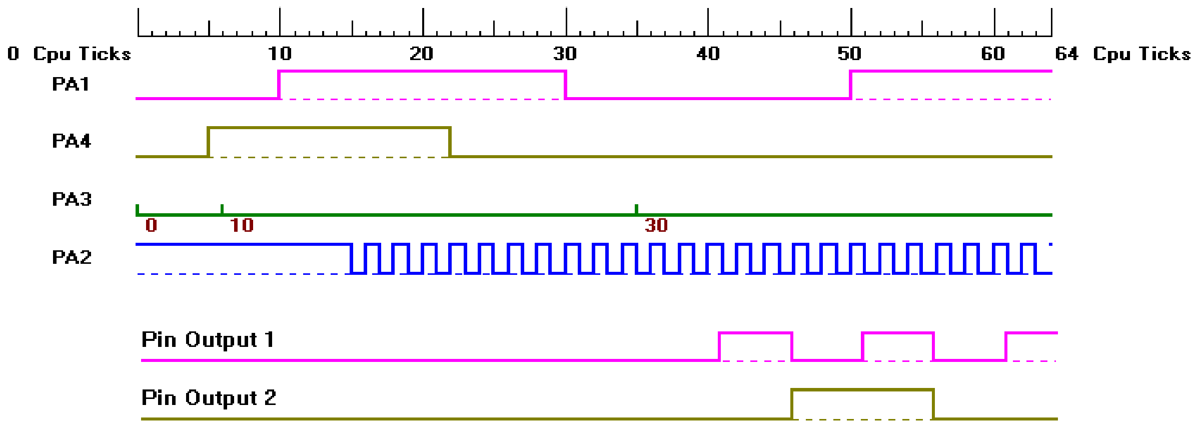
4.14.5 Viewing Pin Output Generated by your Program

To display the output signals resulting from the simulation process use the Waveform Editor that has been supplied with your simulator.

For this, click the Waveform Editor icon or select the corresponding menu in the cascading sequence on your Windows desktop.

NOTE: The name of the Waveform Editor file supplied with your simulator is ***grapher.exe***, NOT *waveform.exe*.

Within the Waveform Editor environment, click on the **File** menu, then click **Load**. Loading **`<filename.out>`** into the Waveform Editor will produce a diagram as the example shown below:



This diagram is only a snapshot of the output pin behaviour as a result of the simulation process. This information is not real-time and cannot be considered real-time.

NOTE: Input information (from the **`<filename>.in`** file) is systematically copied into the **`<filename>.out`** file. So simulated input information and output values can be examined at the same time.

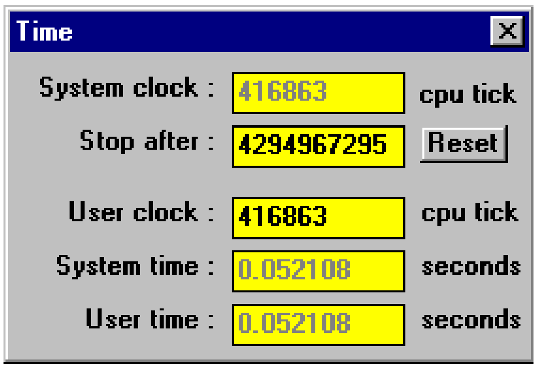
For more information on how to use the ***Waveform Editor***, open the on-line help file provided with the Editor.

4.15 Time Management

When using WGDB7 with a simulator, you can access features closely related to time management.

These features enables you to:

- View the cpu time spent since the program to be debugged was started (***system time***), or the time calculated from an arbitrary origin you specified (***user time***),
- Set (or reset) program breaks based on elapsed cpu time.



System clock :	416863	cpu tick
Stop after :	4294967295	Reset
User clock :	416863	cpu tick
System time :	0.052108	seconds
User time :	0.052108	seconds

To set a new time origin:

1 On the **Windows** menu, click **Time**. The Time window opens.

2 In the **User clock** field, enter the time origin in clock cycles (“ticks”) for the user clock. The value in the User time field changes accordingly.

To set a time break:

1 On the **Windows** menu, click **Time**. The Time window opens.

2 In the **Stop after** field, enter the number of clock cycles (“ticks”) to be generated until a program break is triggered.

To reset a time break:

1 On the **Windows** menu, click **Time**. The Time window opens.

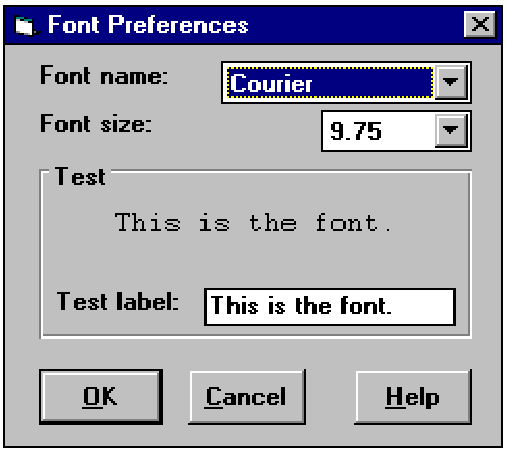
2 Click the **Reset** button. The value in the Stop after field changes to a quasi-infinite value, which means that no time break would occur.

5 CUSTOMISING WGDB7

You can customise the following features in WGDB7:

- The screen fonts.
- The breakpoint line, Program Counter line and currently selected line colours.
- The action taken when events occur.

1 **Changing the Screen Fonts**On the **File** menu, point to Preferences, then click **Screen Fonts**. The **Font Preferences** dialog box opens:



Font Preferences

Font name: Courier

Font size: 9.75

Test

This is the font.

Test label: This is the font.

2 Select a new font type from the **Name** drop-down list.

3 Select a new font size from the **Size** drop-down list.

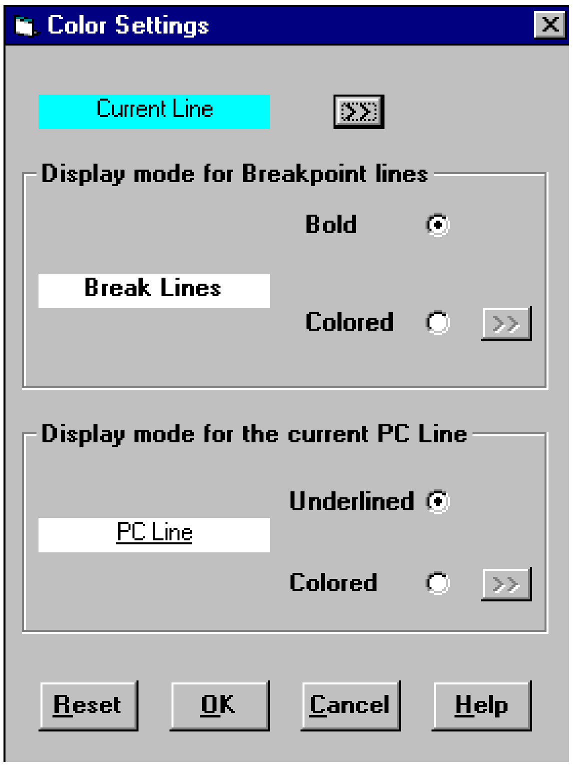
4 Click **OK** to implement the changes you made.

5.1 Changing the Colour Settings

To change the breakpoint line, Program Counter line and currently selected line colours and appearance:

1 On the **File** menu, point to **Preferences**, then click **Color Settings**.

The **Color Settings** dialog box opens:



Color Settings

Current Line

Display mode for Breakpoint lines

Break Lines — Bold [✓] / Colored

Display mode for the current PC Line

PC Line — Underlined [✓] / Colored

2 To change the color of the current line marker, click ⊳⊳ next to the Current Line caption to list the available marker colours and select one.

3 To change the display mode for breakpoint lines, in the **Display mode for breakpoint lines box**:

iSelect the **Bold** button to display the current breakpoint line in bold.

- **ii** Select the **Colored** button then click >> to list the available marker colours and select one for the current breakpoint line.

3 To change the display mode for the current PC line, in the **Display mode for the current PC line** box:

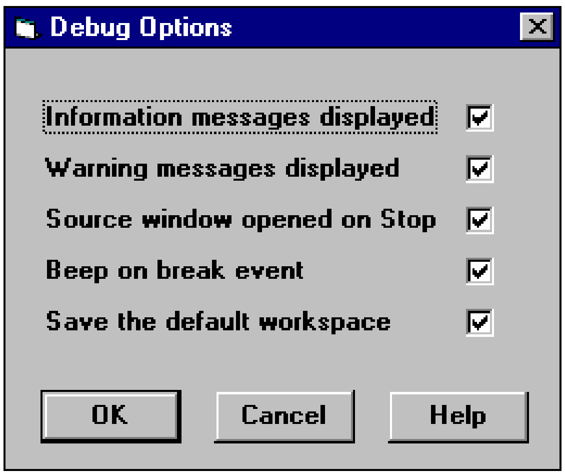
- **i** Select the **Bold** button to display the current PC line in bold.
- **ii** Select the **Colored** button then click >> to list the available marker colours and select one for the current PC line.

3 Click **OK** to implement the changes you made.

5.2 Selecting Which Events are Indicated

1 On the **File** menu point to **Preferences**, then **Debug Options**.

The **Debug Options** dialog box opens:



Information messages displayed ✓
Warning messages displayed ✓
Source window opened on Stop ✓
Beep on break event ✓
Save the default workspace ✓

Click the appropriate check box:

When the **Information messages displayed** box is checked, information messages are displayed. Information messages inform you, for example, when a Hardware Breakpoint is triggered or a stack overflow occurs.

When the **Warning messages displayed** box is checked, warning messages are displayed. Warning messages indicate, for example, when required symbol information is not available.

When the **Source window opened on Stop** box is checked, the module window is displayed when program execution is stopped.

When the **Beep on break Event** box is checked, a beep sound is made when a breakpoint is reached.

When the **Save the default workspace** box is checked, program workspace configurations are saved when programs are closed (see next section).

6 WORKING WITH WORKSPACES

Workspaces are made up of the following definitions relating to each program you load to WGDB7:

- Open windows.
- Window positions.
- Software breakpoint definitions.
- Current cursor position in each module window.
- Window states (Snapshot, Hot or Real-time).
- Disassembly options.
- Trace options.

When you load a program, the default workspace configuration that you were using when you last closed it is restored, thus you can continue working from where you left off the previous time.

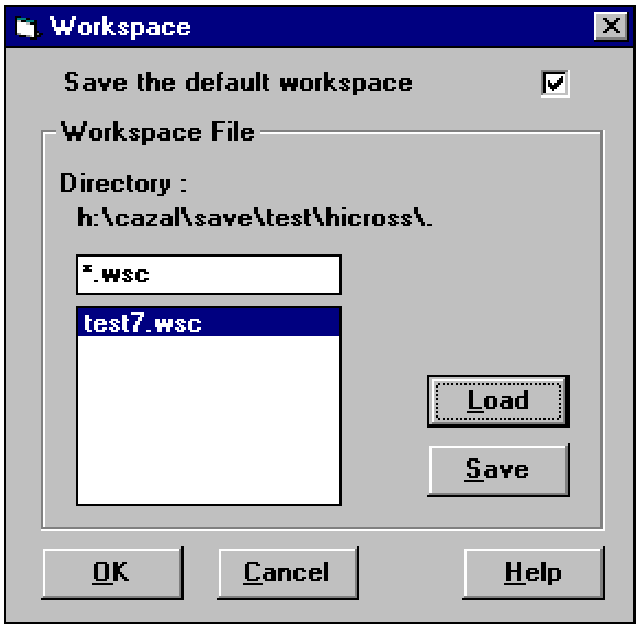
WGDB7 enables you to save a workspace definition, so that you can restore it at a later date. It also enables you to enable/disable automatic default workspace saving, so each time you close a program its workspace is saved.

NOTE: Since workspaces are directly linked to programs, you can only load or save workspace configurations after the program to which they pertain have been loaded.

6.1 Saving and Loading Workspace Definitions

1 On the **File** menu point to **Preferences**, then click **Workspace**.

The **Workspace** dialog box opens:



Save the default workspace ✓

Directory : h:\cazal\save\test\hicross\.

*.wsc

test7.wsc

2 To load a saved workspace settings file:

i Select the file you want to load from the list.

ii Click the **Load** button

To save the current workspace settings:

i Type the file name in the file name box.

ii Click the **Save** button.

3 Click **OK** to implement the changes you made.

6.2 Enabling/Disabling Automatic Default Workspace Saving

To enable/disable automatic workspace saving when you close a program:

1 On the **File** menu point to **Preferences**, then click **Workspace**.

The **Workspace** dialog box opens.

2 In the **Workspace** dialog box, click the **Save the Default Workspace** check box.

When this box is checked, any workspace modifications you make are saved when you close programs.

7 USING GDB7 COMMANDS

WGDB7 is a Windows interface to GDB7 commands (that is GNU and specific ST7 commands). When you choose a WGDB7 menu option, button, or enter a field, WGDB7 executes an appropriate GDB7 command. Using WGDB7, you can:

- Execute GDB7 commands when a program is loaded.
- Execute GDB7 command batch files.
- Enter GDB7 commands using your keyboard.
- View the GDB7 dialog executed by WGDB7.
- Record GDB7 command dialog in a log file.

The following paragraphs explain how to perform these tasks.

7.1 Executing GDB7 Commands When a Program is Loaded

When you load a program, WGDB7 executes either of the following files if they exist in the program directory:

```
hardware.gdb
<program>.gdb
```

where **`<program>`** is the name of the loaded program. This enables you to create program-specific commands each time you load a program.

You can create GDB7 files using any ASCII text editor.

Each command must be on a separate line.

For details about the available commands and their syntax, either type

```
help <command_name>
```

in the Console window or click GDB Commands on Contents page of the WGDB7 online help.

7.2 Executing GDB7 Command Batch Files

Executing batch files can be useful for automatic test-driving programs and creating regression testing suites. You can create batch files of GDB7 commands that are executed without the WGDB7 windows interface. An example of such a file is included with WGDB7, it is named **batch.gdb** and is stored in the WGDB7 installation directory (**`c:\st7tools`**).

To execute this batch file, execute **GDB7.exe** in the WGDB7 installation directory, with the argument:

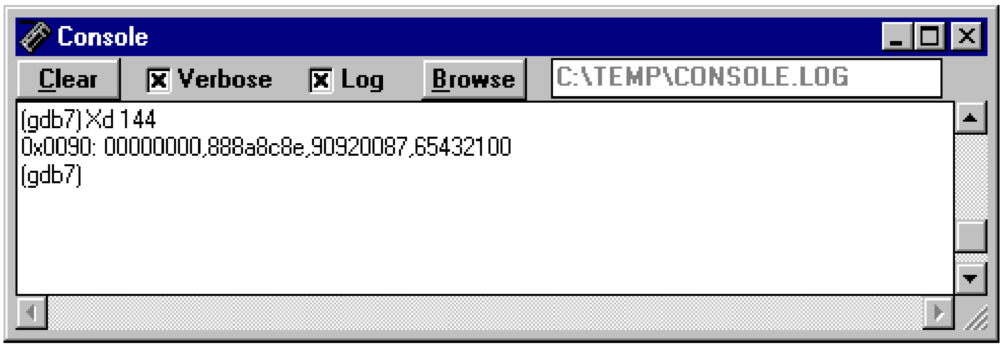
```
-command=c:/st7tools/batch.gdb -batch
```

7.3 Entering GDB7 Commands Using Your Keyboard

WGDB7 lets you enter GDB7 commands using your keyboard. To enter GDB7 commands using your keyboard:

1 On the **Windows** menu, click **Console**.

The Console window now opens, enabling you to enter commands using your keyboard:



C:\TEMP\CONSOLE.LOG

```
(gdb7) Xd 144
0x0090: 00000000,888a8c8e,90920087,65432100
(gdb7)
```

2 Click the first available line in the Console window.

3 Type the command, then press the Enter key. GDB7 interprets the line on which the cursor was when you pressed Enter. Therefore, to reissue a command, place the cursor on the command, and press Enter. GDB7 repeats the last command if Enter is pressed on an empty line.

Note that you can access help on GDB commands either by:

- Typing **`help`** followed by the command name about which you want help in the Console window, then pressing **Enter**

  or

- In the WGDB7 main window, clicking **Help|Contents** then choosing GDB Commands for STMicroelectronics-specific commands

  or

- In the WGDB7 main window, clicking **Help|GDB Commands** for general GDB commands.

7.4 Viewing GDB7 Dialog Executed by WGDB7

To view the GDB7 commands that WGDB7 executes:

1 On the **Windows** menu, click **Console**.

The Console window now opens.

2 In the Console window, check the **Verbose** check box.

You can now view all the GDB7 commands that are executed by WGDB7 in the Console window and their results.

7.5 Recording GDB7 Commands in a Log FIle

WGDB7 lets you record all the GDB7 commands that it executes and their results in a session in a log file:

1 On the **Windows** menu, click **Console**.

The Console window now opens.

2 In the Console window, check the **Log** check box.

3 To create a new log file, click the **Browse** button and select the drive and folder you want to create the new file in, then enter the file name in the **File Name** box.

4 To use an existing log file, click the **Browse** button and select the drive and folder containing the file, then select the file name in the **File Name** list.

You can also store input commands (the results of commands generated by WGDB7 and sent to GDB7) and command outputs (those commands generated by GDB7 and sent to WGDB7) separately in files.

To record input commands, use the GDB7 command: **`set remotelogfile <filename>`**.

To record command outputs, use the commands:

```
wfopen <filename>
```

where **`<filename>`** is the file to record command outputs in, and

```
wfclose
```

when you have finished.

8 QUESTIONS AND ANSWERS

8.1 What does the hour glass cursor mean?

When the cursor appears as an hour glass (⌛), this means that WGDB7 is executing a command, or the program is running.

Note that you can click the WGDB7 Stop button to stop program execution and return the cursor to its normal state.

8.2 Why is the Locals window empty?

This is either because you are debugging a macro-assembler application—macro assembler language does not have local variables, or you are debugging a C-language application, and the current function does not have any variables or parameters.

8.3 How do I specify the location of source files?

To find source files, WGDB7 uses a 'source path' that works like the MS-DOS path feature: the directories specified in the source path are read one after another until the file name is found.

By default, source files must be located in the same directory as the program file. If your source files are in another directory or in several directories, you can use the **`directory <pathname>`** command to the front of the source path.

You can specify several path names using the "**`;`**" separator. When used without parameters, the directory command resets the source path to the current directory.

Example:

```
directory c:\tester\sources;c:\tester\library
```

You can put this command into the program startup file named **`<program>.GDB`** to automatically set the path to the application source files (see “Executing GDB7 Commands When a Program is Loaded” on page 66).

8.4 How can I modify a Hardware Breakpoint?

You cannot modify the address or address range of a Hardware Breakpoint. You must delete the breakpoint then and create a new one.

8.5 Why are some software breakpoints never triggered?

If a software breakpoint is not triggered when you think it should be, check that it is not disabled in the BreakPoints List window (see “Managing Software Breakpoints” on page 36).

8.6 Why are some Hardware Breakpoints never triggered?

If a Hardware Breakpoint is not triggered when you think it should be, check that it is not disabled in the Hardware Breakpoints window (see “Setting Hardware Breakpoints” on page 38).

8.7 What does “Stop at user request” mean?

This message is displayed when you stop the execution of your program. If you do not want this kind of message to be displayed:

1 On the File menu, point to Preferences, then click Debug Options.

2 Uncheck the Information Messages displayed check box.

Information messages will no longer be displayed.

8.8 How Do I execute WGDB7 from Winedit?

To execute WGDB7 from Winedit, enter the full wgdb7.exe path, and specify the same argument string as that specified by the WGDB7 emulator/simulator shortcut or icon.

Index

Index

Index

Notes: