

WGDB6 Windows Debugger For the ST6 Family User Manual

Release 2.1

November 2000



Ref: DOC-ST6-WGDB6

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED.

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

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

Table of Contents

Chapter 1:	WGDB6	5
1.1	Your system requirements	5
1.2	Installing WGDB6 and other ST6 software tools	5
1.3	Launching WGDB6	7
Chapter 2:	Introduction	9
2.1	What is the difference between Simulator ST6 and Emulator ST6?	9
2.2	What Can I Use WGDB6 For?	9
2.3	What Can I Use Wave Form Editor For?	10
2.4	What are Pretty Format Addresses?	10
Chapter 3:	Preparing Programs for Debugging	13
Chapter 4:	Using WGDB6	15
4.1	Using the Control Bar	15
4.2	Getting Help	16
4.3	Selecting a micro	17
4.4	Loading a Program	19
4.5	Viewing Program Information	20
4.6	Viewing/Editing Disassembled Program Code	25
4.7	Viewing/Editing ST6 Memory Contents	26
4.8	Viewing/Editing Register Contents	27
4.9	Viewing/Editing Time Information	28
4.10	Executing Loaded Programs	29
4.11	Using Software Breakpoints	30
4.12	Using Memory Breakpoints	31
4.13	Working with the Trace Buffer	35
Chapter 5:	Customizing WGDB6	39
5.1	Changing the Screen Fonts	39
5.2	Changing the Color Settings	39
5.3	Selecting Which Events are Indicated	41
Chapter 6:	Working with Workspaces	43
6.1	Saving and Loading Workspace Definitions	44
6.2	Enabling/Disabling Automatic Default Workspace Saving	44

Table of Contents

Chapter 7:	Using GDB6 Commands	45
7.1	Executing GDB6 Commands on Startup	45
7.2	Entering GDB6 Commands Using Your Keyboard	45
7.3	Viewing GDB6 Commands Executed by WGDB6	46
7.4	Recording GDB6 Commands in a Log File	47
Chapter 8:	WGDB6 Questions and Answers	49
8.1	What does the hour glass cursor mean?	49
8.2	What does the information message “Program stopped at Stack Overflow breakpoint” mean?	49
8.3	Why is the Locals window empty?	49
8.4	How do I specify the location of source files?	49
8.5	How can I modify a memory breakpoint?	50
8.6	Why are some software breakpoints never triggered?	50
Appendix A:	Bug List	51
Appendix B:	Glossary	53
Product Support		57
Index		59

1 WGDB6

1.1 Your system requirements

To run/install the software provided on the *MCU on CD* CD-ROM, you must have a PC running either Microsoft® Windows® 95, Windows® 98 or Windows® NT®.

1.2 Installing WGDB6 and other ST6 software tools

Your emulator comes with the *MCU on CD* CD-ROM which contains a number of ST6 software tools. To install them, follow these steps:

- 1 Place the *MCU on CD* CD-ROM in your CD-ROM drive. The CD-ROM's autorun feature opens up a welcome screen on your PC.

If the autorun feature does not work, use Windows® Explorer to browse to the CD-ROM's root folder, and double-click on `welcome.exe`.

- 2 Select ***Install Your Development Tools*** from the list of options. A new screen appears listing the different families of STMicroelectronics MCUs.
- 3 Use your mouse to place the cursor over the ***ST6 TOOLS*** option. Choose ***ST TOOLS*** and ***ST6 TOOLCHAIN*** from the lists that appear.
- 4 The install wizard is launched. Follow the instructions that appear on the screen.

You can choose the package you wish to install. To install the complete ST6 Toolchain for your emulator, select ***Complete Toolchain for Emulator***. This option installs the WGDB6 debugger version for your emulator, as well as a Windows Epromer and ST6 Assembler-Linker software.

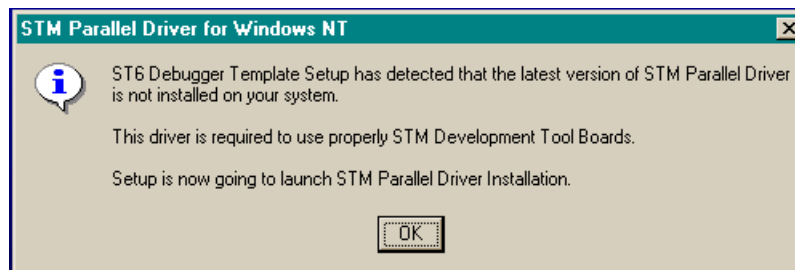
Alternatively, you can choose to perform a custom installation where you choose which of the available software applications you wish to install.

Note: In order to configure your emulator, you must, as a minimum, install the ***ST6 WGDB6 for Emulator***

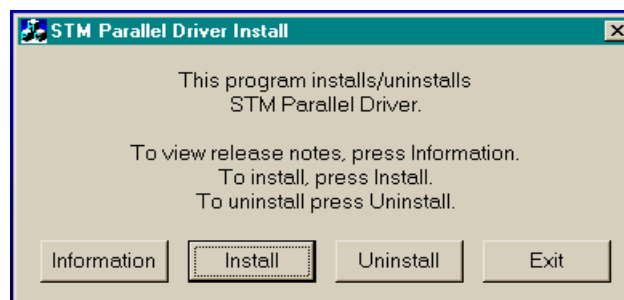
If you do not choose any options, but click ***Next>***, the ST6 Assembler-Linker will be installed by default.

- 5 Follow the instructions that appear on your screen. You will be prompted to select the parallel port you wish to connect the emulator to, as well as the program folder that the software will be installed to.
- 6 If you are installing WGDB6 on a Windows® NT® platform, you must install the Windows® NT® parallel port driver supplied on the CD-ROM.

A window pops up if you have not already installed this driver (*parstm.sys*).



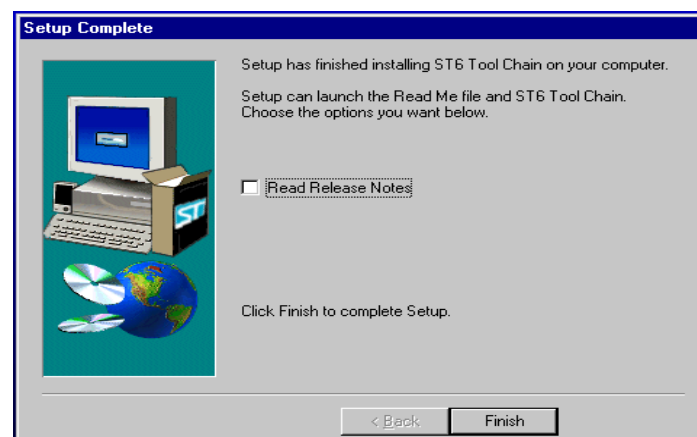
- 7 Click **OK**. The following window appears:



- 8 Click **Install**. The following window appears:



- 9 Click **OK**, the installation is now complete. The following window appears.



- 10 Choose to read the **Release Notes** or start WGDB6, then click the **Finish** button.

1.3 Launching WGDB6

From Windows[®] 95, 98 or Windows[®] NT, click the **Start** button, point to **Programs** -> **ST6 Tool Chain** -> **Development Tools**, then choose **Wgdb6 Emulator**.

Refer to the *WGDB6 Windows Debugger for the ST6 Family User Manual* for full instructions on how to use WGDB6. This manual is available in PDF format on the *MCU on CD* CD-ROM.

2 INTRODUCTION

2.1 What is the difference between Simulator ST6 and Emulator ST6?

Simulator ST6 and Emulator ST6 are two distinct software tools for debugging ST6 programs.

- **Simulator ST6** controls the execution of ST6 programs running in the memory of the development PC.
- **Emulator ST6** controls the execution of ST6 programs running in the memory of an ST6 emulator.

However, both tools have much in common since they both:

- Are supplied in the same installation package.
- Have the same user interface.
- Are described in this manual and jointly called **WGDB6**.
- Can be used for debugging programs developed using the STMicroelectronics AST6 Macro-Assembler and LST6 linker.
- Run under Windows[®] NT[®], Windows[®] 98, Windows[®] 95, or Windows[®] 3.x.

When you run Emulator ST6, if the emulator probe is connected to your application board, the I/O signals will interact with your hardware in the same way as they would if your program was programmed in an ST6 device on your application board.

Simulator ST6 does not require the presence of the ST6 emulator—it can work as a standalone software or with an ST6 Starter Kit (for performing simple emulating functions).

2.2 What Can I Use WGDB6 For?

WGDB6 enables you to execute ST6 programs, and view the contents of the ST6 data and program memory as the program progresses. You can examine source assembler code. Program execution history can be viewed at source or instruction level using the Trace and Stack displays. WGDB6 lets you read and write all ST6 registers and memory locations.

WGDB6 can display data in either *normal*, *hot* or *real-time*. Normal display data is displayed as it was when you chose to view it; it is not automatically updated. Normal display data is displayed on a white background. Hot display data is updated every time the execution of the program you are debugging is suspended, and is displayed on a yellow background. Real-time display data is updated when the program is running, and is displayed on a red background.

WGDB6 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 WGDB6 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 later. See “Working with Workspaces” on page 43 for further details.

WGDB6 includes the following debugging features:

- Lets you set software breakpoints on source or disassembled code, that stop the program running when a chosen instruction is reached.
- Lets you set memory breakpoints, which stop the program running when a pre-defined area of memory is accessed and optionally when a hardware-related condition is met.
- 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.
- Keeps a trace of memory access during program execution.
- Enables you to view the stack contents.
- Enables you to view and modify the simulated/emulated ST6 memory and register contents.
- Enables you to view and modify data symbol values in real time, that is when your program is running. (A data symbol can be a RAM variable, a register, a label, etc.)
- Executes automatically GDB6 command batch files, with or without the WGDB6 graphical interface, at start-up.

2.3 What Can I Use Wave Form Editor For?

The wave form editor is a standalone program for simulating input signals by software. It is intended for use with **Simulator ST6**. You do not need to use it with **Emulator ST6**, since the emulator works with input signals from the application hardware.

2.4 What are Pretty Format Addresses?

To access an object in areas of paged memory within the program or data space you can use an address format known as *pretty address*.

Pretty addresses have the following format:

P:pp:nnn to address the program space.

or

D:pp:nn to address the data space.

where pp is the page number in hexadecimal and nnn or nn is the offset from the beginning of the page in hexadecimal.

Each page in the program space is 2048 bytes (nnn in the range 0x000 to 0x7FF). An offset greater than or equal to 0x800 always means page number 1 (the static page): in this case pp is insignificant.

Each page in the data space is 64 bytes (nn in the range 0x00 to 0x3F). An offset greater than or equal to 0x40 means a data address outside the dynamic page area: in this case pp is insignificant.

Examples:

P:00:0F0 addresses byte 240 (F0h) in page 0 of the program space.

D:08:1A addresses byte 26 (1Ah) in page 8 of the data space.

P:01:080 is equivalent to P:00:880 or P:01:880 or P:02:880.

D:00:80 is equivalent to D:01:80.

3 PREPARING PROGRAMS FOR DEBUGGING

You can debug programs that were generated using the STMicroelectronics ST6 Macro-Assembler.

Some example macro-assembler programs that are fully prepared for debugging are provided in the `simex` subdirectory of the WGDB6 installation directory.

To be able to load a program, WGDB6 must have access to the *.hex file in the Intel hexadecimal format. With the *.hex file, WGDB6 recognizes the symbol files that were generated by the toolchain. For the STMicroelectronics ST6 Macro-Assembler compiler, the symbol files may include .SYM, .MAP, .DSD and .LIS files.

To generate files with the correct debug information for WGDB6, when you assemble programs using the STMicroelectronics ST6 Macro-Assembler, you must use the `PP_ON` directive (refer to the AST6/LST6 user guide).

```
ast6 -l -o filename.asm for each source module, and lst6 -i -m -s -  
o filename filename.obj for each object module
```

If no symbol file is provided with the .hex file, you can still perform some debugging tasks, such as viewing ST6 memory contents, viewing disassembled code, displaying and modifying register values, executing programs and viewing trace buffer contents.

Note: *If the program is in a single source file and the generated code is less than 4K bytes, it is possible to generate code without using the linker, by entering the following command line:*

```
ast6 -l -m -s filename.asm
```

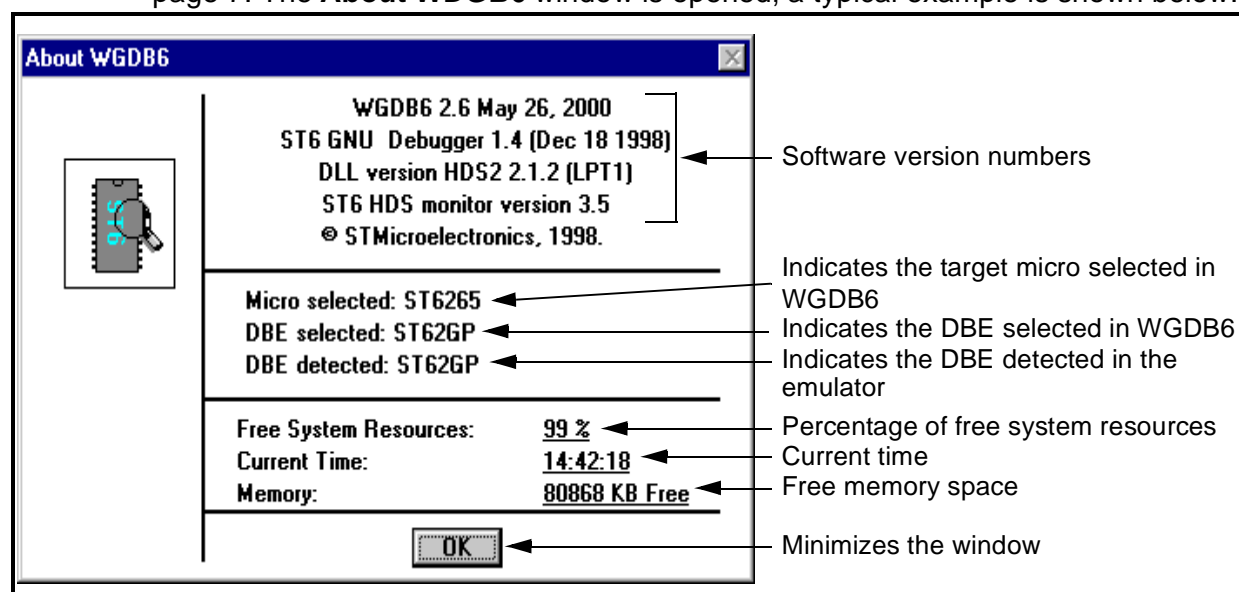
However, this option should be used with caution, because the debug information is not sufficient for WGDB6 and correct debugging functionality is not guaranteed.

4 USING WGDB6

This section describes how to perform the basic tasks that you will carry out when you debug a program using WGDB6. It aims to get you quickly started with WGDB6. For detailed information about windows, menus and dialog boxes, and on how to perform all tasks, use the WGDB6 online help (see “Getting Help” on page 16).

4.1 Using the Control Bar

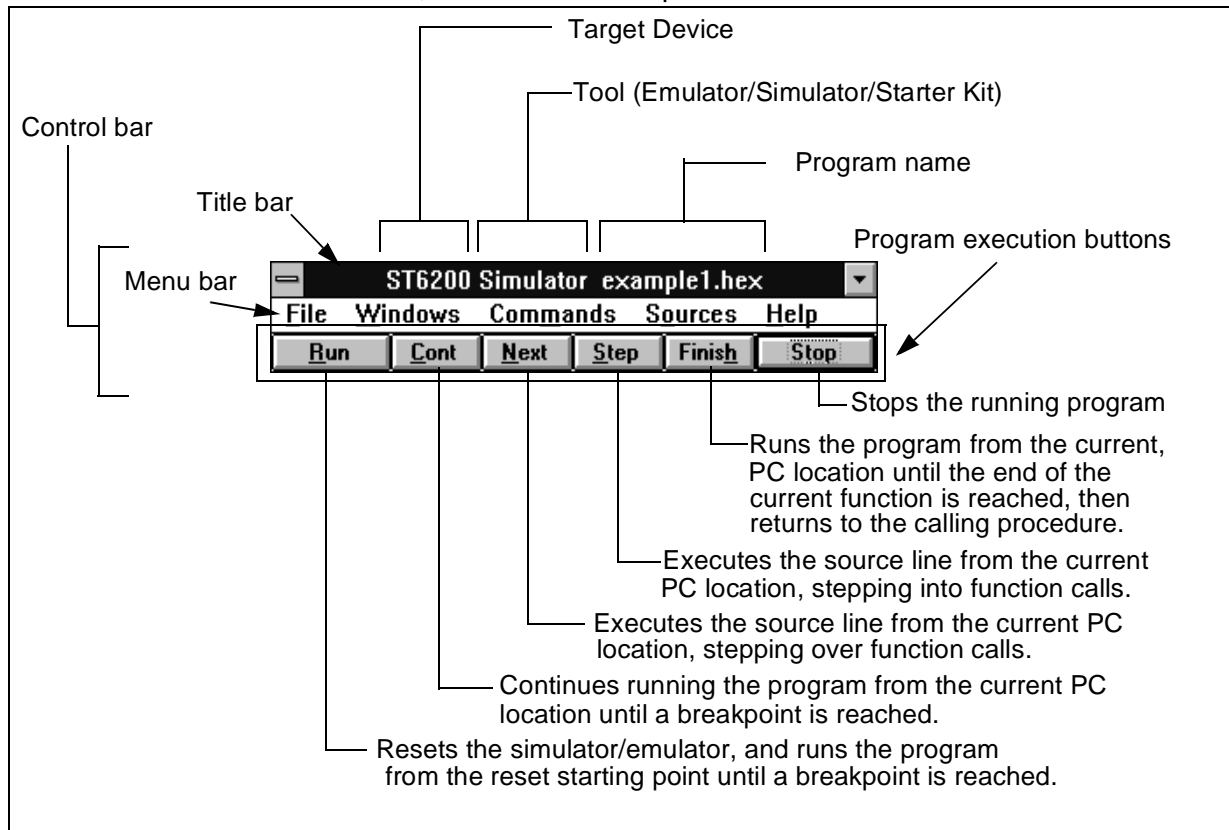
To start WGDB6 proceed as described in *Section 1.3: Launching WGDB6* on page 7. The **About WGDB6** window is opened, a typical example is shown below:



Note: The appearance of the About WGDB6 window varies depending on whether Emulator ST6, Simulator ST6 or Starter Kit ST6 is being used.

When an emulator is being used, do not click on the **OK** button but wait for the communication link between the emulator/simulator and the PC to be synchronized.

Once WGDB6 starts, the Control Bar opens:



The Title bar displays the debugging tool (Emulator, Simulator or Starter Kit, the ST6 family name currently being simulated/emulated and the name of the program that is currently loaded (see “Loading a Program” on page 19).

The Menu bar displays the available menus. Clicking a menu name opens it.

The program execution buttons provide you with quick access to the program execution functions.

Note: All the task procedures described in this book start from the control bar.

4.2 Getting Help

WGDB6 enables you to access two types of help:

- 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 Control Bar click **Contents**.

- 2 Click a topic from the list to display the information.

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

- 1 In 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 the item, such as a menu name or field, about which you want further information.

4.3 Selecting a micro

Having started WGDB6, you can now select the micro to be emulated/simulated. If you are using an emulator you must also select a DBE family. Proceed as follows:

- 1 **Only perform this step if you are using an emulator.** Select a DBE family by clicking **Commands -> DBE Family -> ST6xxx** on the main menu.

Note: Only the ST62GP-DBE is automatically detected by WGDB6.

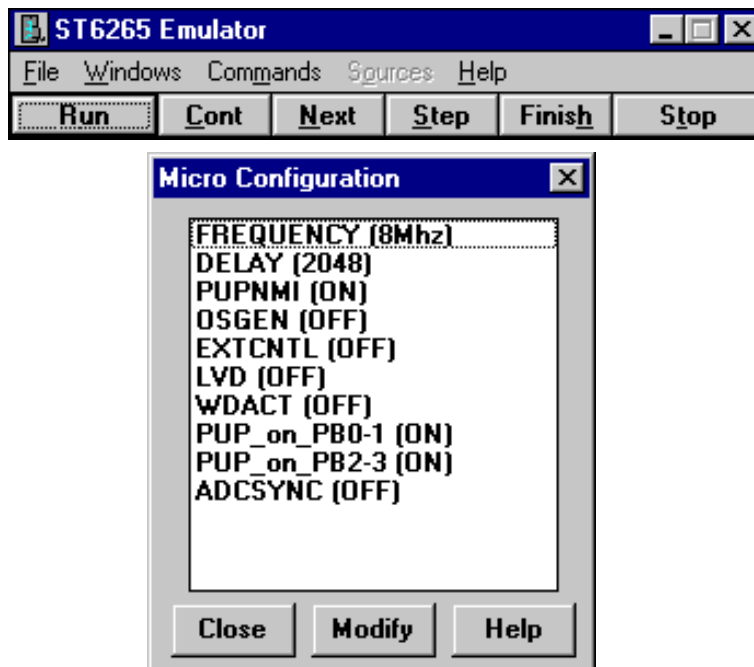
- 2 Select **Commands -> Micro_Name...**, the **Micro Name** window is displayed (if you are using an emulator, micro name selection is only available when a ST62GP-DBE is detected):



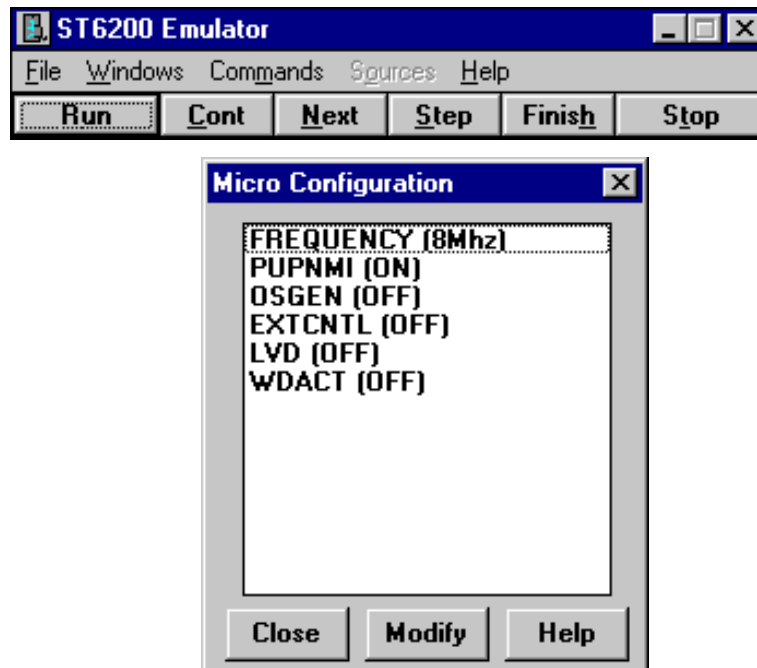
- 3 In the **Micro Name** window choose a family from the pull down list and then choose a micro from the micro list.

- 4 After you have selected the micro name, from the main menu, select **Commands -> Micro_Configuration....** Depending on the micro name selected, different configuration options are available.

For example, the ST6265 emulator has the following Micro Configuration options:



However, the ST6200 emulator has the following Micro Configuration options:



The first option, **frequency**, is the same for all emulators. The options listed below this depend on the specific emulator.

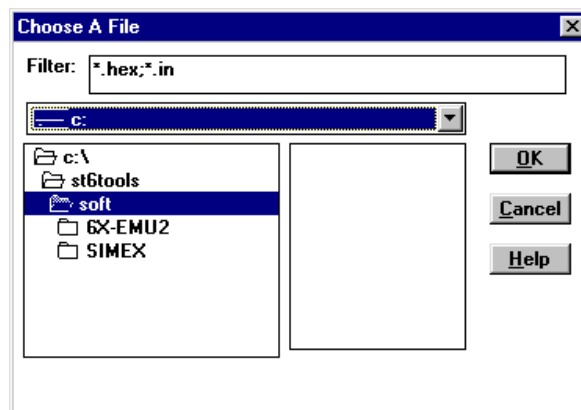
These options can be configured using the **Modify** button.

4.4 Loading a Program

When you load a program, you must specify the .HEX file or the .IN file of the program to load. 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 13.

- 1 In the **File** menu, click **Open**.

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



- 2 In the drive list, click the drive that contains the program.
- 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, in the **Command** menu: click **Display PC** or click the **Reset** button.

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

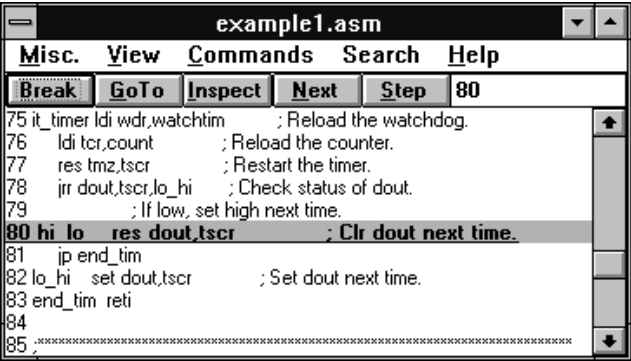
4.5 Viewing Program Information

4.5.1 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:



The source module will open automatically when a file is loaded.

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

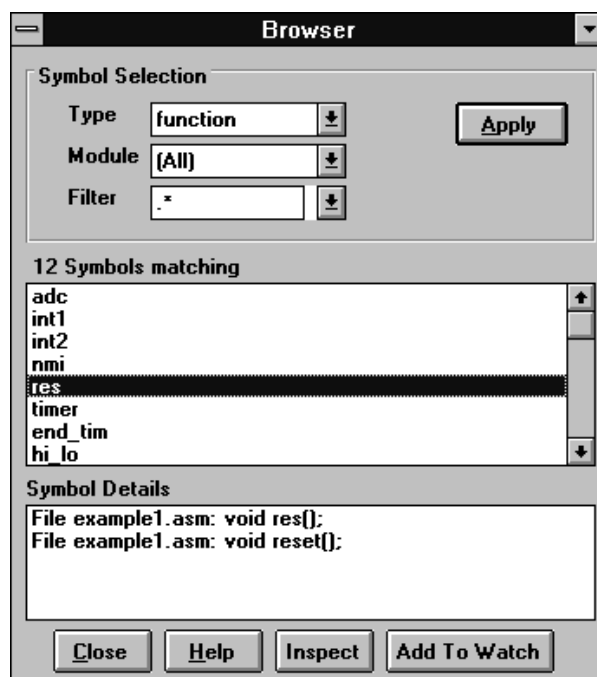
To do this:	Do this:
View further details about any item displayed in the window.	Click the item then click the Inspect button.
View a line of code disassembled.	Click the line number, then click the Inspect button.
Step or go to another part of the loaded program.	Click the appropriate button: Next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

4.5.2 Finding and Viewing Symbols

To find symbols and view detailed information about them:

- 1 In the **Windows** menu, click **Browser**.

The **Browser** window opens:



- 2 In the **Symbol Selection** box:
 - a Select the symbol type you want to find from the **Type** drop-down list. (A data symbol can be a RAM variable, a register, a label, etc.)
 - b Select the module that contains the symbol you want to find from the **Module** drop-down list.
 - c 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”.

- d Click the **Apply** button.
- 3 The **Symbols matching** box lists the symbols found using the selection criteria entered in the **Symbol Selection** box.

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

- Clicking the symbol and viewing the **Symbol Details** box.
- 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/Editing Disassembled Program Code” on page 25) or the **Module** window (see “Viewing a Source Module” on page 20), depending on the available source file types. If you selected data or a constant, its value and type are displayed in the **Inspect** window.

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

4.5.3 Watching Variable or Expression Values

WGDB6 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).

Note: Variables located in data RAM banks cannot be watched.

To view a variable or expression value in the **Watch** window:

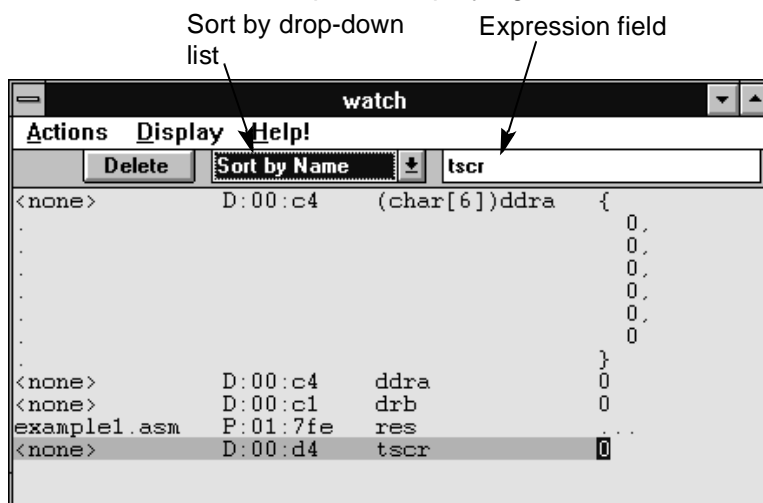
1 Either:

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

or

In 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:

To do this:	Do this:
Change the value of a symbol.	Click on the value and type over it.
Change the format in which newly displayed symbols are displayed.	Choose the appropriate option from the Prefs submenu In the Display menu.
Change the type of data displayed.	Choose the appropriate option from the Format submenu In the Display menu.
Change the radix in which the value of the selected symbol is displayed.	Choose the appropriate option from the Base submenu In the Display menu.
Choose the symbol type that the 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 box.
View a range of <i>n</i> bytes starting from a variable or an address.	Enter the variable name or * followed by the address, then followed by @ <i>n</i> in the Expression box. For example, to view the 5 bytes starting at address D:00:80, enter: *D:00:80@5.
View a subrange of <i>n</i> values of an array.	Enter * followed by the array or pointer name followed by @ <i>n</i> in the Expression box.
View a selected function disassembled.	Select the function, then click Disassembler In the Actions menu.
View and modify the data held in memory for a selected symbol.	Select the symbol, then click Dump In the Actions menu.
View the value of a selected symbol in the Inspect window.	Select the symbol, then click Inspect In the Actions menu.
View a selected function within the source code.	Select the symbol, then click Sources In the Actions menu.

4.5.4 Viewing/Editing Data Symbols in Real Time

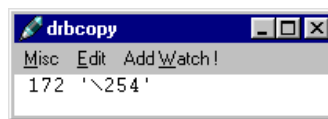
1 Either:

Select a data variable using the **Browser** window (see “Finding and Viewing Symbols” on page 21), then click the **Inspect** button,

or

In the **Module** window, type the data symbol name in the field in the top-right corner of the window, then press the Enter key (“Viewing a Source Module” on page 20).

The **Inspect** window opens:



2 In the **Misc.** menu, click **Real Time**.

The window background color becomes red, indicating that it is running in real-time mode. Executing the program will now result in the displayed values being updated.

3 To update the displayed data value while the program is running:

In the **Edit** menu, click **Modify**.

Type a new value, then click the **Set** button.

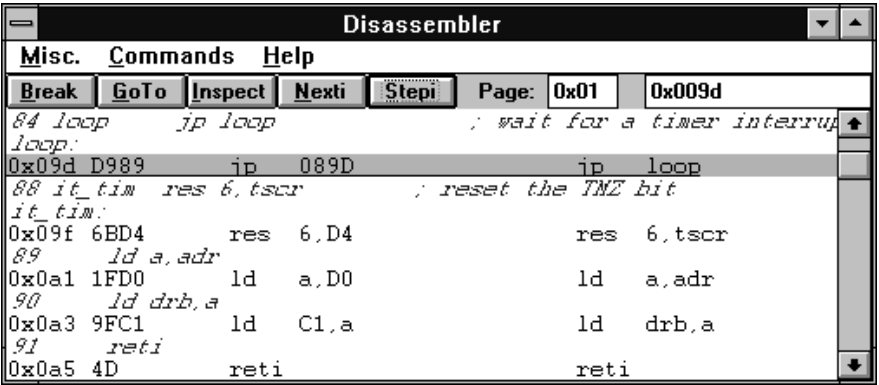
The data value is now updated accordingly.

Note: This function cannot be used in programs including *Stop* or *Wait* instructions.

4.6 Viewing/Editing Disassembled Program Code

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

- 1 In the **Windows** menu, click **Disassembler**.
- 2 The **Disassembler** window opens:



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

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

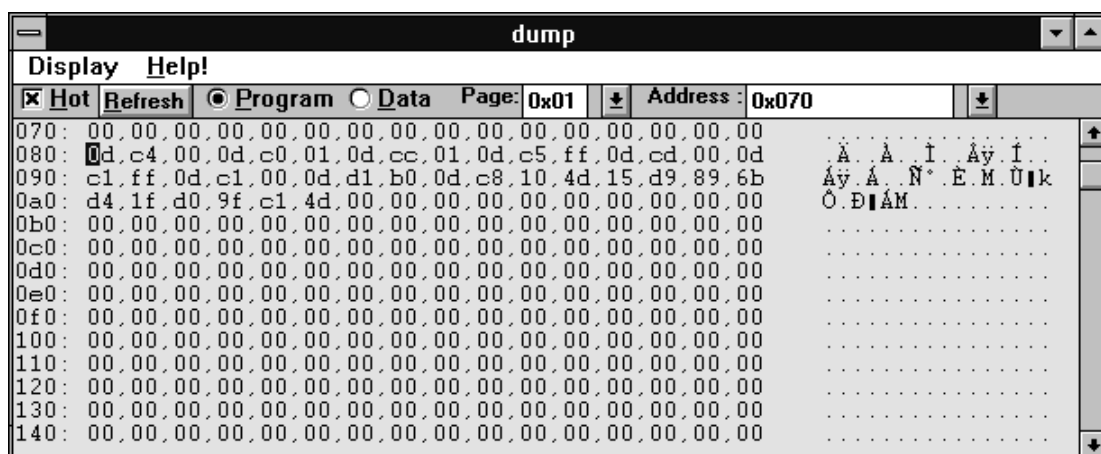
To do this:	Do this:
Display the memory contents in a different format	Choose another format from the Options command in the Misc. menu.
Display contents starting from a selected page	Click the Page field and enter a page number.
View more detailed information about a selected symbol	Select the symbol 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.
Enter and assemble instructions online in the ST6 memory.	Click on the disassembled line you want to modify, then in the Commands menu, click Online Assemble . For more information about, this refer to the online help

Note: The destination address of conditional jumps is not displayed correctly in the **Disassembler** window. The jumps are executed correctly.

4.7 Viewing/Editing ST6 Memory Contents

To view the contents of the ST6 memory:

- 1 In the **Windows** menu, click **Dump**.
- 2 The **Dump** window now opens, displaying the contents of the ST6 memory:



Note that before you have loaded a program, the ST6 program memory will be set to 0. 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 check box (the window background becomes yellow indicating that the window is hot).
Display contents starting from a selected page (for ST6 devices with dynamic pages)	Click the Program or Data radio button and select or enter the Page value from the DRPR register for data or the PRPR register for programs.
Display contents starting from a selected address	Choose the memory space (Program or Data) whose contents you want to view, by clicking the appropriate button, then enter the page whose contents you want to view in the Page field, and the offset to the beginning of the area you want to view in the Address field. Press the Enter key.
Display contents starting from a symbol name	Enter the symbol name in the Address field.
Edit contents of a selected address	Click the value that you want to modify and enter a new value on the keyboard.

4.8 Viewing/Editing Register Contents

You can view and modify register contents using WGDB6:

- 1 In the **Windows** menu, click **Registers**.
- 2 The **Registers** window now opens:



- 3 To modify the contents of a register, select it, type in the new value and press the [Enter] key.

The table below describes the fields in the Registers window:

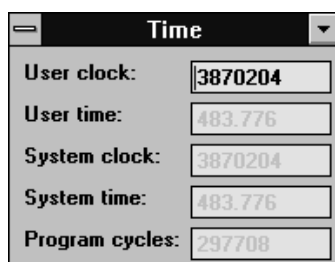
This field:	Displays this:
Program Counter	The Program Counter (PC) register value. This is a 12-bit register that stores the address of the next instruction to be executed.
Stack	The Stack Pointer (SP) value. This value is in the range 0 to 5 indicating the current nested call or interrupt level.
Accumulator	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) and the Short Direct registers (V and W). The Indirect registers are used during register-indirect addressing mode as pointers to memory locations in the data space. The Short Direct registers are used to save one byte in short direct addressing mode.

This field:	Displays this:
Paged Registers	<p>PRPR is the Program ROM Page Register. This defines the 2-Kbyte program space page to be addressed. The area 0 to 0x7FF in the program space is paged.</p> <p>DRBR is the Data RAM/EEPROM Bank Register. This selects the 64-byte RAM/EEPROM data space page to be addressed. The area 0 to 0x3F in the data space is paged.</p> <p>DRWR is the Data ROM Window Register. This defines the data ROM window location. This area is addressed as part of the data space between 0x40 to 0x7F.</p>
Flags	<p>NMI is the Non-Maskable Interrupt Flag. This indicates that the ST6 memory is in non-maskable interrupt mode, which is the default mode after reset.</p> <p>INT is the Interrupt Mode Flag. This is used during interrupt mode operation. When the Int bit is set, all interrupts are disabled except for NMI. Clearing this bit enables them.</p> <p>NORM is the Normal Flag. This indicates that the ST6 is in normal mode, and not currently executing an interrupt.</p> <p>The carry (C) flag is set when a carry or borrow occurs during arithmetic operations, otherwise it is cleared. The zero (Z) flag is set when the result of the last arithmetic or logical operation was zero, otherwise it is cleared. See the appropriate Databook for the ST6 family type you are using for further details.</p>

4.9 Viewing/Editing Time Information

You can only perform this task if you are using the ST6 simulator.

- 1 In the **Windows** menu, click **Time**.
- 2 The **Time** window now opens:



The table below describes the fields in the **Time** window:

This field:	Displays this value:
User clock	Displays the current user clock time. This is the time (expressed as a number of external clock oscillations) that elapsed between the loaded program being executed and that execution being stopped. By default, this value is the same as the system clock value, however this value can be modified. You can modify this value by over typing the displayed value and pressing the [Enter] key.
User time	Displays the user time value (in milliseconds). This is the user clock value divided by simulated clock frequency (to view or update the frequency, click Frequency in the Commands menu).
System clock	Displays the system clock value. This value cannot be modified.
System time	Displays the system time value. This is the system clock value divided by simulated clock frequency (to view or update the frequency, click Frequency in the Commands menu).
Program cycles	Displays the number of cycles that the program has executed. In ST6 family microcontrollers, 1 cycle = 13 oscillator pulses (1.625 μ s), if an 8 MHz external oscillator is used.

4.10 Executing Loaded Programs

WGDB6 includes the following program execution commands, which are available either as buttons in the **Control Bar** or other windows (where appropriate) or in the **Commands** menu. The following table explains what each command does:

This command:	Does this:
Run	Resets the simulator/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 running program.

This command:	Does this:
Reset	Resets the ST6 simulator/emulator. The PC is set to the starting point and A, X, Y, PRPR, DRBR, DRWR are set to 0 and mode is set to NMI (see “Viewing/Editing Register Contents” on page 27 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 and the PRPR values to where you want to start running the program in the **Register** window (see “Viewing/Editing Register Contents” on page 27), then click the **Cont** button.

4.11 Using Software Breakpoints

4.11.1 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, In the **Windows** menu, click **Disassembler**.

In the **Module** or **Disassembler** window:

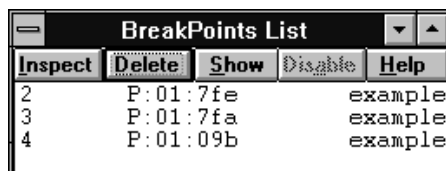
- 1 Click the line of code on which you want to set a breakpoint.
- 2 Click the **Break** button.

A breakpoint is now set on the selected line. This line is highlighted in bold, 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, WGDB6 will set the breakpoint on the next available line.

4.11.2 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, In the **Windows** menu click **Soft Breakpoints**. The **BreakPoints List** window is as follows:



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

To do this:	Do this:
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.

4.12 Using Memory Breakpoints

4.12.1 Setting Memory Breakpoints

You can set memory breakpoints on the following events:

- 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 specified hardware conditions are met.

Note: Memory breakpoints are not saved in workspace so they will not be available when you reload your application. (However, software breakpoints are saved in the workspace.)

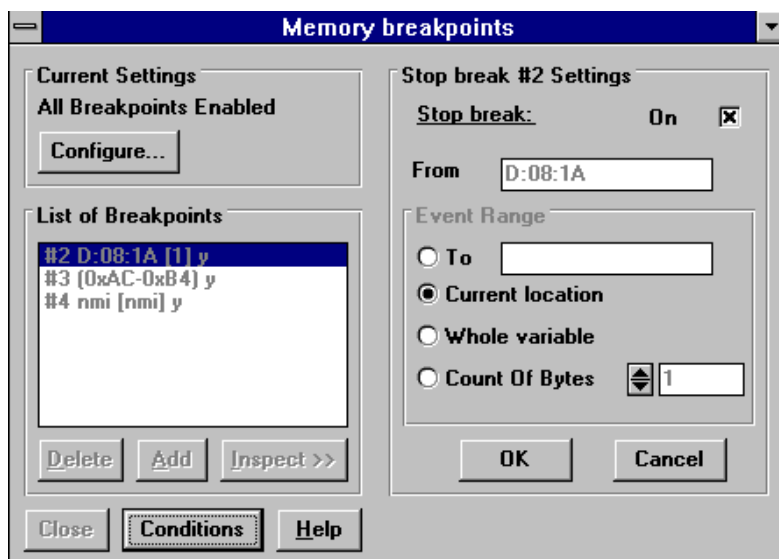
To set a memory breakpoint:

- 1 In the **Windows** menu, point to **Hardware Events** then click **Memory Breakpoints**.

The **Memory breakpoints** window opens.

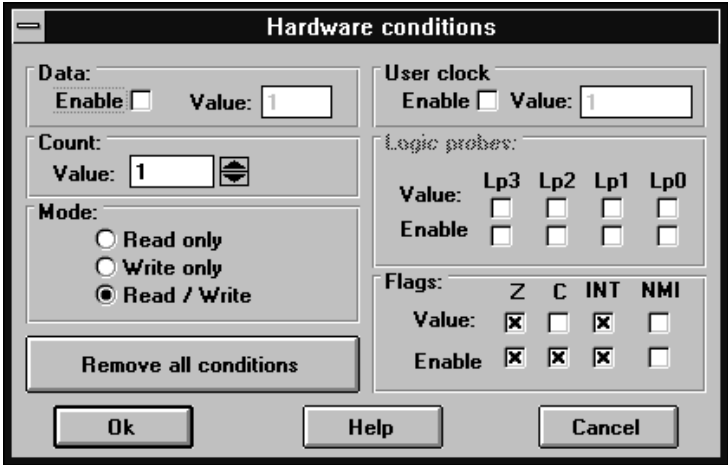
- 2 In the **Memory breakpoints** window, click the **Add** button.

The **New Settings** box opens:



- 3 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 a Pretty address (see “What are Pretty Format Addresses?” on page 10).
- 4 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 must be expressed as a Pretty 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 entire 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.
- 5 To attach specific hardware conditions to memory breakpoints, such as activation when a specified value is written to or read from memory, click the **Conditions** button.

The **Hardware conditions** dialog box opens:



Follow the instructions in the table below to set conditions on memory breakpoints:

To do this:	Do this:
Trigger the breakpoint when a specified data value is read or written. This condition will apply to all breakpoints set on data.	Check the Enable check box in the Data: box, and enter the value you want to trigger the breakpoint on in the Value: field.
Trigger the breakpoint after it has been reached a specified number of times. This condition will apply to all breakpoints.	Enter the number of times you want the breakpoint to be reached before it is triggered in the Value: field in the Count box. For example, if you enter 2 here, the breakpoint is triggered the second time it is reached.
Choose the access type. This should be the condition on which a breakpoint that is set on data is triggered. This condition will apply to all breakpoints set on data.	In the Mode box: Select Read only to trigger the breakpoint on read access. Select Write only to trigger the breakpoint on write access. Select Read/Write only to trigger the breakpoint on read or write access.
Trigger a breakpoint on the simulated clock counter value (simulator only).	In the User clock box, click Enable , then enter the simulated clock value at which you want the breakpoint to be triggered in the Value field. This value may range between 0 and $2^{32} - 1$. Its default value is 0.

To do this:	Do this:
Trigger a breakpoint on a specified logic probe value pattern (emulator only).	In the Logic probes: box, check the Enable boxes associated with the logic probes whose values you want to match, then click the Value: boxes to specify the values you want to match.
Trigger a breakpoint set on data on a specified flags value pattern. This condition will apply to all breakpoints set on data.	In the Flags: box, check the Enable boxes associated with the flags whose values you want to match, then click the Value: boxes to specify the values you want to match.

4.12.2 Managing Memory Breakpoints

You can perform all memory breakpoint management tasks from the **Memory Breakpoints** window (refer to “Setting Memory Breakpoints” on page 31). The table below lists the memory breakpoint management tasks you can perform and explains how to perform them:

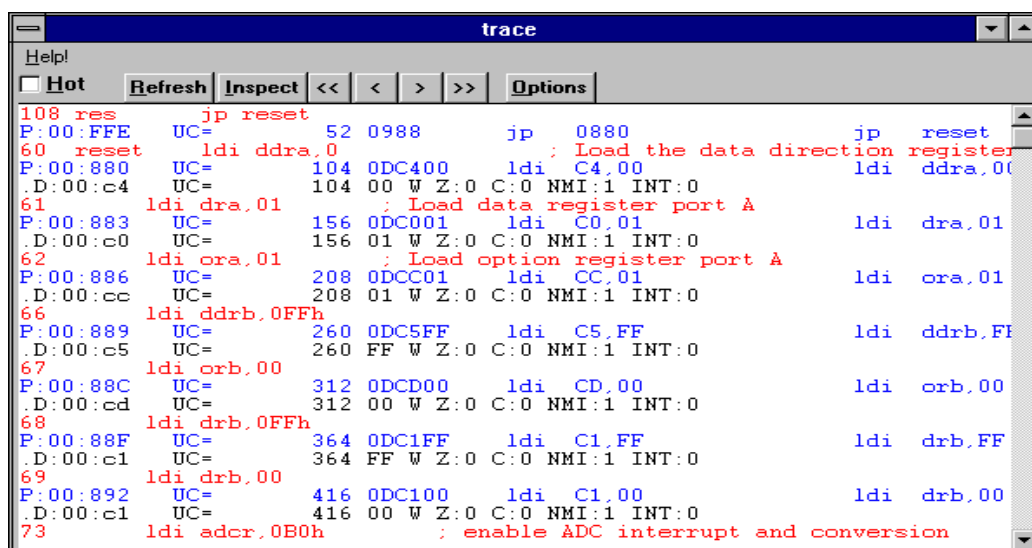
To do this:	Do this:
Enable/disable a breakpoint.	Select the breakpoint from the List of Breakpoints , click the Inspect button, then click the On check box in the New Settings box. When the On box is checked, the breakpoint is enabled.
Delete a breakpoint.	Select the breakpoint from the List of Breakpoints , then click the Delete button.
Enable/disable all breakpoints (memory and software).	In the Windows menu, point to Hardware Events, then click Memory Breakpoints . The Memory Breakpoints window opens. Click the Configure button. The Hardware Event Advanced Settings window opens. Click the Enable/Disable All check box in the Breakpoints box to enable/disable all breakpoints. When it is checked, all breakpoints are enabled.

4.13 Working with the Trace Buffer

The emulator trace buffer records some hardware events that occur when a program is executed. Since its size is limited, you can specify what type of events you want to record. WGDB6 enables you to view either all the trace buffer contents or selected event types. The following paragraphs explain how to view trace buffer contents, how to select the type of events to view and how to specify what type of events are recorded in it.

4.13.1 Viewing Trace Buffer Contents

- 1 In 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:



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, the window is Hot. The window changes from white to yellow, if Hot.
Update the Trace window contents.	Click the Refresh button. You only need to do this if the window is not Hot ; if it is Hot , it updates automatically every time program execution is suspended.
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.
View selected event types.	Click the Options button, then select the event types you want to display from 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 can reduce the execution speed of WGDB6. You can save time by not setting this screen as hot, and updating the display when required using the **Refresh** button.*

4.13.2 Selecting the Type of Events to be Recorded in the Trace Buffer

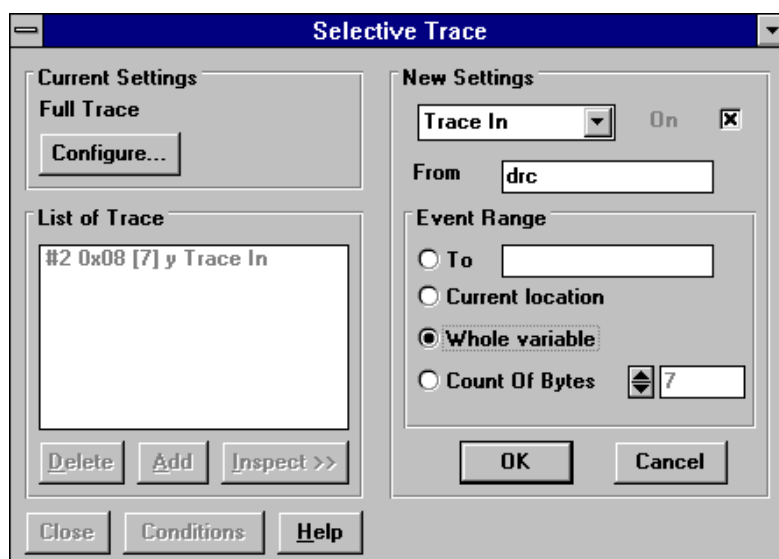
By default, event selection is disabled. This corresponds to the 'full-trace' selection mode. When event selection is disabled, the following information is recorded in the trace buffer:

- The instruction fetch cycle, that provides information on the instruction that was executed.
- The last cycle of the instruction, that provides information on the data that was accessed: its address, value, access mode and other related information, such as the current ST6 mode and flag values.

To record selected event types only:

- 1 In the **Windows** menu in the Control Bar, click **Hardware Events**, then click **Selective Trace**.

The **Selective Trace** dialog box opens.



- 2 Click the **Configure** button.
 - a The **Hardware Event Advanced Settings** dialog box opens.
 - b To only record the cycles defined as TRACEIN events (see below), select **Sampling Filter**.
 - c To only record the cycles that occur between a TRACEON event and a TRACEOFF event (see below), select **Trace On / OFF**.
 - d Click **OK**.
- 3 In the **Selective Trace** dialog box, click the **Add** button.
The **New Settings** box opens.
- 4 Select the event type you want to define from the event type drop down list in the **New Settings** box:

Select this:	To define:
TRACEIN	An area of memory whose related events are recorded in the trace buffer (Sampling Filter mode only).
TRACEON	An area of memory, which when accessed, starts event recording in the trace buffer (Trace On / Off mode only).
TRACEOFF	An area of memory, which when accessed, ends event recording in the trace buffer (Trace On / Off mode only).

5 In the **From** field, enter the start of the address range whose related events you want to define. This can be expressed as a variable (symbolic name) or as a pretty address.

6 In the **Event Range** box, choose the address range whose related events you want to define:

Select **To** to enter the end address of the memory range whose related events you want to define. This can be expressed either as a variable (symbolic name) or as a Pretty address.

Select **Current location** to set the related events to only the address defined in the **From** field is accessed (range = 1 byte).

Select **Whole variable** to set the related event to the whole variable range that is directly linked to the symbol type specified in the **From** field (for example, on whole fields of data symbols for a C-language struct data type).

Select **Count of bytes** to define the number of bytes, starting from the address pointed to by the **From** field, whose related events you want to define.

7 Click **OK**.

Repeat steps 3 to 7 if you want to define more event types.

You can also delete event types by selecting the definition and clicking the **Delete** button, or enable/disable definitions by clicking the event definition in the **List of Trace** field then clicking the **Inspect** button, then checking/unchecking the **On** check box and clicking **OK**.

To reinstate the default (full trace mode) so that all cycles are recorded in the trace buffer:

1 In the **Selective Trace** dialog box, click the **Configure** button.

2 In the **Hardware Event Advanced Settings** dialog box, select **Full Trace**.

3 Click **OK** in both the **Hardware Event Advanced Settings** dialog box and the **Selective Trace** dialog box.

5 CUSTOMIZING WGDB6

You can customize the following features in WGDB6:

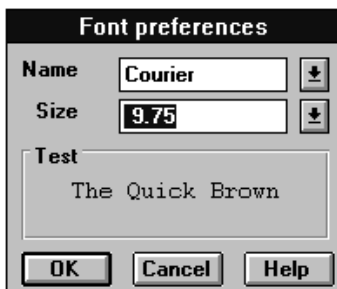
- The screen fonts.
- The breakpoint line, Program Counter line and currently selected line colors.
- The action taken when events occur.

The following paragraphs explain how to do this.

5.1 Changing the Screen Fonts

- 1 Close the file if it is open.
- 2 In the **File** menu, point to **Preferences**, then click **Screen Fonts**.

The **Font preferences** dialog box opens:



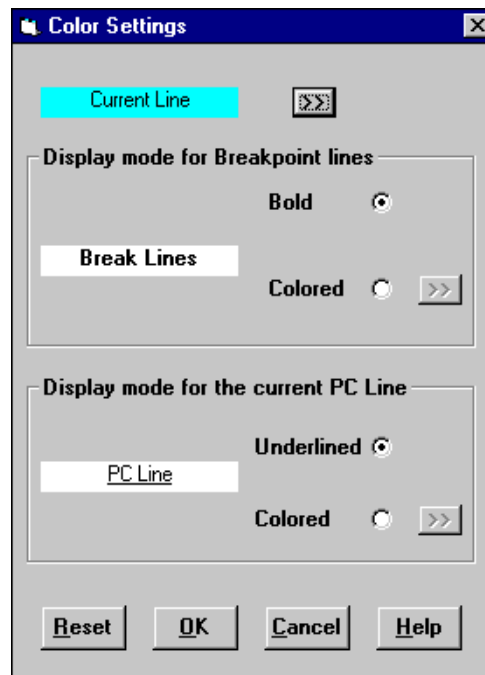
- 3 Select a new font type from the **Name** drop-down list.
 - 4 Select a new font size from the **Size** drop-down list.
 - 5 Click **OK** to implement the changes you made.
- When you re-open the file the changes will be reflected.




5.2 Changing the Color Settings

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

- 1 In the **File** menu, point to **Preferences**, then click **Color Setting**.

The **Colors Settings** dialog box opens:

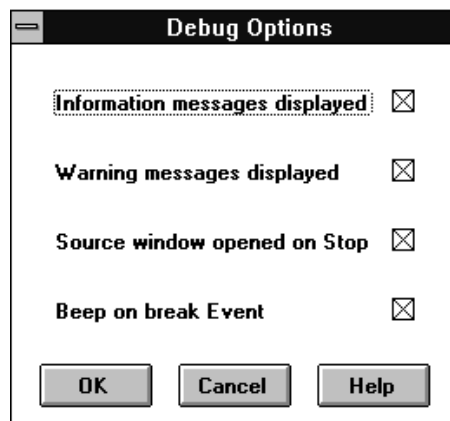


- 2 To change the color of the current line marker, click  next to the **Current Line** caption to list the available marker colors and select one.
- 3 To change the display mode for breakpoint lines, in the **Display mode for Breakpoint lines** box:
 - a Select the **Bold** button to display the current breakpoint line in bold.
 - b Select the **Colored** button then click  to list the available marker colors and select one for the current breakpoint line.
- 4 To change the display mode for the current PC line, in the **Display mode for the current PC Line** box:
 - a Select the **Underlined** button to display the current PC line as underlined.
 - b Select the **Colored** button then click  to list the available marker colors and select one for the current PC line.
- 5 Click **OK** to implement the changes you made.

5.3 Selecting Which Events are Indicated

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

The **Debug Options** dialog box opens.



- 2 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 memory 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.

6 WORKING WITH WORKSPACES

Workspaces are made up of the following definitions relating to each program you load to WGDB6:

- 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.

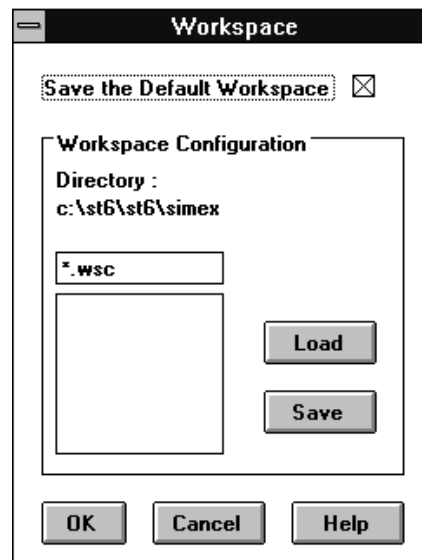
WGDB6 enables you to save a workspace definition, so that you can restore it later. It also enables you to enable/disable automatic default workspace saving, so each time you close a program its workspace is saved.

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 In the **File** menu point to **Preferences**, then click **Work Space**.

The **Workspace** dialog box opens:



- 2 To load a saved workspace settings file:
 - Select the file you want to load from the list.
 - Click the **Load** button.

To save the current workspace settings:

- Type the file name in the **File Name** box.
- 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 In the **File** menu point to **Preferences**, then click **Work Space**.

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 GDB6 COMMANDS

WGDB6 is a Windows[®] interface to GDB6 commands (that is GNU and specific ST6 commands). When you choose a WGDB6 menu option, button, or enter a field, WGDB6 executes an appropriate GDB6 command. Using WGDB6, you can:

- Execute GDB6 commands when WGDB6 is started.
- Execute GDB6 command batch files.
- Enter GDB6 commands using your keyboard.
- View the GDB6 commands executed by WGDB6.
- Record GDB6 commands in a log file.

The following paragraphs explain how to perform these tasks.

7.1 Executing GDB6 Commands on Startup

When you load a program, WGDB6 executes either of the following files if they exist:

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.

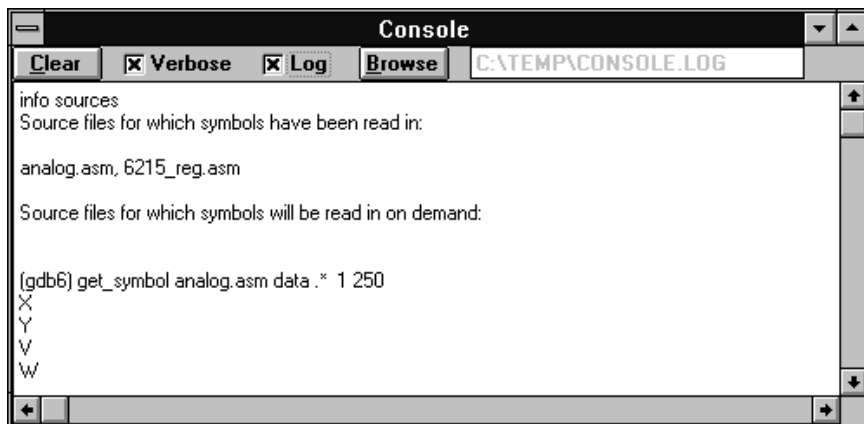
You can create GDB6 files using any ASCII text editor. Each command must be on a separate line. For details about the available commands and their syntax, click GDB Commands on Contents page of the WGDB6 online help.

7.2 Entering GDB6 Commands Using Your Keyboard

WGDB6 lets you enter GDB6 commands using your keyboard. To enter GDB6 commands using your keyboard:

- 1 In the **Windows** menu, click **Console**.

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



- 2 Click the first available line in the **Console** window.
- 3 Type the command, then press the Enter key. GDB6 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. GDB6 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 WGDB6 Control Bar, clicking **Help|Contents** then choosing GDB Commands for STMicroelectronics-specific commands,
- or
- in the WGDB6 Control Bar, clicking **Help|GDB Commands** for general GDB commands.

7.3 Viewing GDB6 Commands Executed by WGDB6

To view the GDB6 commands that WGDB6 executes:

- 1 In 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 GDB6 commands that are executed by WGDB6 in the **Console** window.

7.4 Recording GDB6 Commands in a Log File

WGDB6 lets you record all the GDB6 commands that it executes and their results in a session in a log file:

- 1 In the **Windows** menu, click **Console**.

The **Console** window now opens.

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

The **Browse** window now opens.

- 3 In the **Browse** window 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, 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 WGDB6 and sent to GDB6) and command outputs (those commands generated by GDB6 and sent to WGDB6) separately in files. To record input commands, use the GDB6 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 WGDB6 QUESTIONS AND ANSWERS

8.1 What does the hour glass cursor mean?

When the cursor appears as an hour glass () , this means that WGDB6 is executing a command.

Note that if you move the cursor to outside a **WGDB6** window, or a non-active **WGDB6** window, although the cursor returns to its normal shape, you cannot use other programs.

8.2 What does the information message “Program stopped at Stack Overflow breakpoint” mean?

This means that the program has exceeded the maximum of six nested functions or has been interrupted in the ST6 stack during execution.

To disable break on stack overflow, In the **Windows** menu, select **Hardware Events** then **Memory breakpoints**, click the **Configure** button, then uncheck the **Stack Overflow Stop** check box.

8.3 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.

8.4 How do I specify the location of source files?

To find source files, WGDB6 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 GDB6 Commands on Startup” on page 45).

8.5 How can I modify a memory breakpoint?

You cannot modify the address or address range of a memory breakpoint. You must delete the breakpoint then and create a new one. The conditions that you set on the deleted breakpoint are not affected since they apply to all defined memory breakpoints.

8.6 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 31). Next, check that the **Enable/Disable All** check box is checked in the Advanced Settings dialog box (**Windows|Hardware events|Memory Breakpoints|Configure** button).

Note that in some older versions of the HDS emulator (monitor version 0.0), the program is stopped at the instruction immediately following the breakpoint.

APPENDIX A: BUG LIST

	Bug	Work-around
1	The save_memory command does not work with pretty addresses.	Use the save_memory command with real addresses; i.e., save_memory 0x80020000 64 file.hex
2	Windows do not resize properly.	When you wish to re-dimension a window, take care to place the mouse on the outermost edge of the border of the window. When the cursor is correctly positioned, the cursor will become a resizing arrow and when you click the left mouse button the entire window border will be highlighted. If the resizing arrow is incorrectly positioned, you will highlight a portion of the text inside the window you wish to resize. Once the resizing arrow is correctly positioned, you may resize the window.
3	The trace_trace gdb command does not work.	None.
4	<p>The disassembler window displays wrong addresses, but the jump is correctly executed</p> <p>Example:</p> <pre> 0x88a 04 jrz 088b jrz088b 8 JRR flag_nb,flag_reg, \$+5 0x88b 23C102 jrr4,C1,0892 jrr4,drb,0892 </pre> <p>The instruction JRR flag_nb, glaf_reg, \$=5 is translated into jrr 4,C1,0892. 0892 is a wrong address for the jump; it must be 0890. However, the jump is correctly executed.</p>	
5	It is not possible to set a breakpoint at the location where a macro is called in the source window.	None.

APPENDIX B: GLOSSARY

ADC

Analog to digital converter

Dedication board

A dedication board is one of the two boards installed in the enclosure of the ST626x-EMU2. It is the part of the emulator that is specific to a set of ST6 devices. It emulates the RAM, EEPROM, oscillator, timers, watchdog, ADC, SPI and I/Os. It is connected to the probe board by two flat cables. The ST6 HDS2 emulator is available in specific versions that support a group of ST6 devices. Each version of the ST6 HDS2 has a specific dedication board and set of probe boards (See table in Appendix A).

DIL

Dual In Line. Designates a type of device package with two rows of pins for thru-hole mounting. Sometimes also called DIP (Dual In-line Package).

ECP

Extended capabilities port

EPP

Enhanced parallel port

Footprint

Designates the dimensions of the location of a component on a printed circuit board or in a socket. It depends on the number of pins, their size, type and positioning. The footprint of each ST6 device is specified in the datasheet in the section titled *Package Mechanical Data*. (see ST6 MCU FAMILY DATABOOK or CD-ROM).

gdb

GNU debugger

GNU

Acronym for Not UNIX. Name of a project developed by the Free Software Foundation.

Mainframe

Emulator enclosure with only the main board installed.

Main board

The main board is the board installed in the bottom slot of the ST626x-EMU2. It emulates the ST6 program memory, and contains the breakpoint logic, trace memory and all the logic needed for real-time emulation.

LP

Logic probe

Probe board

A probe board is a small printed circuit board with a connector to allow you to insert it in the MCU socket of the target board. It is connected to the dedication board by two flat cables.

RC Network

Resistor-capacitor network

SMD

Surface Mounted Device. Designates a device with pins that are designed to be glued to the surface of a printed circuit board. Contrast with thru-hole devices which have pins that are designed to be inserted in holes on a printed circuit board and soldered.

SO

Small outline. Designates a type of device package with two rows of pins for SMD or socket mounting.

SPI

Serial Peripheral Interface

SSOP

Shrink Small Outline Package. Designates a type of device package like SO but smaller. Emulating these devices requires a special adapter that can be ordered from STMicroelectronics.

Starter Kit

A printed circuit board for microcontroller evaluation and programming. It has the ability to program one DIP EPROM or OTP part at a time. Different starter kits are available for programming different ST6 devices.

ST6 HDS2 Series

The ST6 HDS2 Series is a generic name covering several emulator versions each of which have a specific dedication board to support a set of ST6 devices (See table in Appendix A).

ST626x-EMU2

The ST626x EMU2 is the order code of this specific ST6 HDS2 emulator version with a dedication board for the ST620x, ST621x, ST622x, ST625x and ST626x device families.

Target Board

Designates your application board. It should include a socket for inserting the ST6 device or the emulator probe.

WGDB6

Windows GNU Debugger for ST6. This is a software tool running under Windows. It is the main user interface when operating the ST626x emulator, and is part of the ST6 software development tool chain (editor, assembler, linker and debugger). WGDB6 is supplied on the CD-ROM with the ST626x emulator and must be installed on your development PC.

PRODUCT SUPPORT

Software Updates

You can get software updates from the ST Internet web site:

<http://mcu.st.com>

For information on firmware and hardware revisions, call your distributor or ST using the contact list given above.

If you experience any problems with a software tool, contact the distributor or ST sales office nearest you.

Contact List

*Note: For **American and Canadian customers** seeking technical support the US/Canada is split in 3 territories. According to your area, contact the following sales office and ask to be transferred to an 8-bit microcontroller Field Applications Engineer (FAE).*

Canada and East Coast

STMicroelectronics
Lexington Corporate Center
10 Maguire Road, Building 1, 3rd floor
Lexington, MA 02421
Phone: 781-402-2650

Mid West

STMicroelectronics
1300 East Woodfield Road, Suite 410
Schaumburg, IL 60173
Phone: 847-517-1890

West coast

STMicroelectronics, Inc.
30101 Agoura Court
Suite 118
Agoura Hills, CA 91301
Phone: 818-865-6850

Europe

France (33-1) 47407575
Germany (49-89) 460060
U.K. (44-1628) 890800

Asia/Pacific Region

Japan (81-3) 3280-4120

Hong-Kong (852) 2861 5700

Sydney (61-2) 9580 3811

Taipei (886-2) 2378-8088

Index

Symbols

.DSD files	13
.HEX files	13, 19
.LIS files	13
.MAP files	13
.SYM files	13

A

access type	33
Accumulator register	27
address	
memory breakpoint	32
pretty	10
software breakpoint	32
Address button	26
Address field	25, 26

B

Browser command	21
Browser window	21

C

Choose a File dialog box	19
color settings	40
commands	
Browser	21
Console	46
Disassembler	25
Dump	26
GDB6	45 to 47
Open	19
set remotelogfile	47
Watch	22
wfclose	47
wfopen	47
compatible programs	13
Conditions button	32
configuring	
workspaces	43
Console command	46
Console window	46
Cont button	29
Count of bytes	32

D

Debug Options dialog box	41
Debugging capabilities	10
Default	
Default application directory	49
Default source file directory	49
Directory	
Default application directory	49
Default source file directory	49
Disassembler command	25
Disassembler window	25
display modes	9
DRBR register	28
driver	
NT	5
DRWR register	28
Dump command	26
Dump window	26

E

Event range box	32
events	
choosing the types to record	36
recording in trace buffer	35
selecting indication	41
viewing selected types	36
executing programs	29
expression values	23

F

finding symbols	20
Finish button	29
flags	
INT	28
NMI	28
NORM	28
From field	32

G

GDB6 commands	9, 45 to 47
entering	45
executing on startup	45
recording in log files	47

Index

viewing	46
Goto button	30

H

hardware breakpoints, see memory breakpoints	
hardware conditions	32
Hardware conditions dialog box	33
hardware.gdb file	45
help	16
hot	9, 35
hour glass cursor	49

I

Indirect register	27
Inspect button	20, 22, 35, 38
Inspect window	
real-time	24
INT flag	28

L

List of Trace field	38
local variables	49
Locals window	49
location of source files	49
log file	47

M

Macro-Assembler	13
main window	16
memory breakpoints	
address range	32
deleting	34
disabling	34
enabling	34
hardware conditions	32
setting	31
Memory Breakpoints dialog box	32
Micro_Configuration	18
Micro_Name	17
module window	20

N

New Settings box	32
------------------------	----

Next button	29
Nexti button	30
NMI flag	28
NORM flag	28
NT driver	5

O

online help	16
Open command	19
options	
button	36
command	25
debug	41
Disassembly	36
Macro-Assembler	13

P

Page field	25, 26
Paged Registers	28
pretty address	10
Program Counter register	27
Program cycles	29
PRPR register	28

Q

Questions and answers	49
-----------------------------	----

R

real-time	9, 24
registers	
Accumulator	27
DRBR	28
DRWR	28
Indirect	27
Program Counter	27
PRPR	28
Short Direct	27
Stack Pointer Flag	27
viewing	27
Registers window	27
Reset button	30
Run button	29

Index

S

search operators	21
Selective Trace	36
set remotelogfile command	47
Short Direct register	27
software breakpoints	30
setting	30
Sources	
Default source file directory	49
menu	20
Source files	49
Source path	49
ST6	
C compiler	13
Stack Pointer Flag register	27
starting WGDB6	7
Step button	29
Stepi button	30
Stop button	29
support	
contact numbers for	57
for development kit	57
symbol	
data	23, 24
files	13
finding	21
finding and viewing	20
loading	19
sort by	23
System clock	29
System time	29

T

trace buffer	35 to 38
selecting events to record in	36
viewing contents of	35
Trace window	35

U

updating data symbols in real-time	24
--	----

User clock	29
User time	29

V

viewing	
GDB6 commands	46
program information	20 to 29
registers	27
symbols	20
trace buffer contents	35
viewing data symbols in real-time	24

W

Watch command	22
Watch window	22
WGDB6	
customizing	39 to 41
debugging capacities	10
display modes	9
functionalities	9
icons	16
main window	16
starting	7
windows	
Browser	21
Console	46
Disassembler	25
Dump	26
Inspect	24
Locals	49
main	16
module	20
Registers	27
Trace	35
Watch	22
Windows 3.1	9
Windows 3.x	5
Windows 95	5, 9
Windows 98	5
Windows NT	5
workspaces	43 to 44

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without the express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics.

Intel® is a U.S. registered trademark of Intel Corporation.

Microsoft®, Windows® and Windows NT® are U.S. registered trademarks of Microsoft Corporation.

©2000 STMicroelectronics - All Rights Reserved.

Purchase of I²C Components by STMicroelectronics conveys a license under the Philips I²C Patent. Rights to use these components in an I²C system is granted provided that the system conforms to the I²C Standard Specification as defined by Philips.

STMicroelectronics Group of Companies

Australia - Brazil - China - Finland - France - Germany - Hong Kong - India - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain
Sweden - Switzerland - United Kingdom - U.S.A.

<http://www.st.com>