



# 4

## 186-Specific Debugger Functions Tutorials

### In This Chapter

We present tutorial exercises that introduce you to the features of VisualProbe and how they work with the capabilities of 186 processors.

Target Configuration . . . . .	4-2	Real Time Watch . . . . .	4-25
The Trace Function . . . . .	4-8	Interrupt Registers . . . . .	4-32
Hardware Events . . . . .	4-13	The Interval Analyzer . . . . .	4-35
Multiple Breakpoints and Multiple Defined Hardware Event Terms . . . . .	4-20		

### Exercise 1: Target Configuration

*Exercise 2 is not a formal step-by-step tutorial exercise; rather, it is intended to help you become acquainted with several important target configuration tools offered by VisualProbe Q.E.D.*

*This exercise does not require that a sample file be loaded.*

**Important!**

*This exercise does NOT apply to Q.E.D. Jtag Debugger. For the 386 version, see the 386-specific tutorial examples.*

Target Configuration controls affect fundamental characteristics of the emulation environment in which you debug your code. Although the *specific* Target Configuration controls for 186 and 386 processors differ, we have included Target Configuration as a topic in this “general” section because you must configure these environmental characteristics - and these choices affect how other settings in other windows are implemented.

*If you are working with a 386EX processor, please refer to the exercise **Target Configuration in Chapter 5** (386-specific tutorials) for important information about configuring this protected-mode processor. Then return to this section and continue with Exercise 3.*

We urge you to carefully consider how these settings might best be implemented as you create an appropriate debugging environment for your code.



## Target Configuration Dialog Box

The Target Configuration dialog box and its tabbed pages provide information about the presence or absence of a target in your current emulator/target configuration; and when a target is present, it allows you to configure emulator/target interactions.



Figure 4-1. Target Configuration Status Tab

### 186 Processor Target Configuration: Status Tab

The Status tab provides information about your current target. There are no settings on this page that users can alter - it only provides information that enables you to ascertain whether your expectations about your current target setup are confirmed by VisualProbe.

#### Target Present:

- If a target is present, a “yes” status statement confirms its presence.
- If a target is not present, a “no” statement informs you that VisualProbe Q.E.D. is running in null-target mode.



### Target Power:

Target power can be either 3 or 5 volts - depending on the target processor in use.

### 186 Processor Target Configuration: Pause Mode Tab

The Pause Mode tab lets you specify how target processor timers and interrupts will behave when the processor enters Pause (stopped) mode.

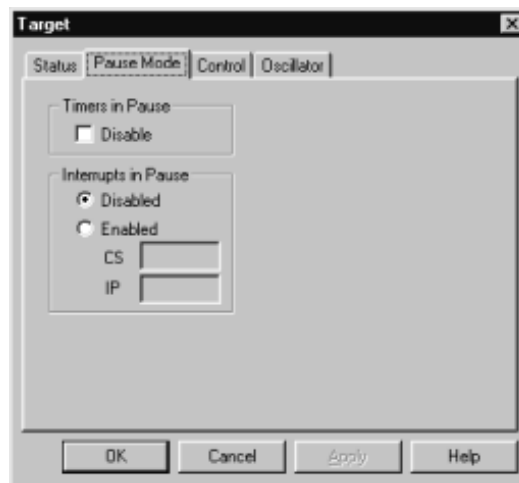


Figure 4-2. Target Configuration Pause Mode Settings

### Timers in Pause

By default, your code will receive timer interrupts when VisualProbe Q.E.D. enters Pause (stopped) mode. If any timers are running, they will continue running when the emulator enters Pause mode.

- If your code depends upon receiving timer interrupts/restores - leave this Disable check box *empty* to allow timer interrupts to be serviced, even when you're stopped.
- If you place a check in the Disable check box, timers will stop when the target processor enters Pause mode, and the state of the timers is restored upon reentering Run mode



**Note:** This applies to ALL of the 386 peripheral units on the 386 Pause mode page.

### Interrupts in Pause

If your code requires that interrupts to be serviced when VisualProbe Q.E.D. enters Pause mode, you can enable interrupts in Pause mode by clicking the Enable check box. You must also assign an *unused* memory address for CS:IP where VisualProbe Q.E.D. can run an algorithm that allows interrupts to be serviced in Pause mode.

## 186 Processor Target Configuration: Control Tab Settings

Control tab settings let you choose how VisualProbe Q.E.D. responds to certain signals related to the target.



**Figure 4-3.** Target Configuration Control Tab Settings

**Note:** If you are running VisualProbe Q.E.D. in null-target mode, certain selections will be grayed-out.

### Ignore Ready:

Check this box if you wish VisualProbe Q.E.D. to ignore the Target Ready signal.



### **Ignore NMI:**

Check this box if you wish VisualProbe Q.E.D. to ignore the target processor's NMI signal.

### **Isolate Target:**

Check this box if you wish to isolate the target's address and data buffers. This setting can be used to help isolate target address problems. For example, if you're having problems starting your system in target mode, you might wish to disconnect the target processor, restart VisualProbe Q.E.D. up in null-target mode, set the Isolate Target check box, then restart VisualProbe Q.E.D. with the target processor attached.

### **Hold Enable:**

Hold is an externally-generated signal that allows a processor to give up the bus to an external device for an arbitrary period of time. Three Hold Enable settings are possible - and only when a target processor is present.

- Never: Ignores any Hold signal from the target. The processor does not relinquish control of the bus whether it is in Run or Pause mode.
- In Run: This setting allows the processor to relinquish control of the bus if a Hold signal is generated by the target during run mode. However, the processor ignores this Hold signal if it is in Pause mode.
- Always: This allows the processor to relinquish control of the bus when a Hold signal is generated by the target in both run and Pause modes.

### **CPU Reset:**

If CPU Reset is disabled, hard resets received from the target will be ignored by the probe tip. When CPU Reset is enabled, the probe tip responds to hard resets received from the target.



- Address Pins Usage (AMD Processors only) allows the specified pins to be toggled between address/PIO settings.

**Ready Break Time:**

This control allows you to set a time - in seconds - after which if a Ready signal is not received from the processor, the probe tip will enter Pause mode and generate a Ready Hang error message. Enter a zero (0) in this field - or leave the line blank to disable Ready Break.

**Drive Target Reset:**

Enabling this control forces a reset of peripherals on the target that is congruent with emulator processor resets. If you wish to reset target circuitry whenever you reset the emulator CPU, check this box.

**186 Processor Target Configuration: Oscillator Tab**

**Figure 4-4.** Target Configuration Oscillator Tab Settings

You can select either the Target or Emulator Generated clock. When the emulator-generated clock is selected, you can enter a frequency in kHz at which you wish the probe tip processor to operate.



## Exercise 2: The Trace Function

*In this exercise, you will open the Tracing window, start Tracing functions, execute the sample code, and view the results of the Tracing function.*

*To run this tutorial in VP Q.E.D., use the 186 sample code file*

*[Timerc.omf](#) included on your BeaconSuite installation CD*

The ability to trace the actual flow of data - as it relates to your source code, physical bus cycles, or disassembly code is an important debugger function that can help you resolve problems that result in unpredicted program behaviors. VisualProbe Q.E.D.'s Trace function captures and displays bus-level code execution history in real time while the target runs.

Tracing is often initiated with manual execution commands selected from the Main menu or Toolbar. Be aware, however, that you can create Defined Hardware Events that cause Tracing to capture bus cycles automatically when a set of logical conditions are met.

This tutorial illustrates the Trace function associated with manual execution of code. Information about creating Defined Hardware Events that trigger Trace functions can be found in the next exercise.

### Enable the Trace function:

1. Open the Tracing window:
  - In the Main menu, select View→Trace. *Or,*
  - Click the Tracing button on the Main window toolbar.



VisualProbe Q.E.D.'s Trace window displays information in several modes, including Bus Cycles only, Disassembly only, Interleaved Source and Disassembly, Source only, and Bus Cycles and Source. By default, all Trace display modes are enabled the first time you run VisualProbe Q.E.D.

2. If the toggling Start/Stop button in the lower left corner of the Tracing window says Stop, then the Trace buffer is already enabled and results of previous Trace executions may be displayed. Click Stop, then click the Clear button and go on to Step 3.

However, if the button says Start, then Trace is not enabled. Click the Start button in the lower left corner of the Tracing window to enable the Trace buffer.



3. For this exercise, you should reset the CPU. From the Main menu, select Debug→Reset CPU.

### Run Source Code and View Trace

Now that the Trace function is enabled and the CPU has been reset, you will instruct VisualProbe Q.E.D. to run the sample code to `main`, then stop source code execution.

1. In the Main menu, select Debug→Go Until...
2. The Enter Address dialog box opens, and requires that you enter a symbol or address to mark the stopping point for code execution. For this exercise, you will enter the value `main`.



**Figure 4-5.** The Enter Address Dialog Box

Click in the Address data entry field and enter the variable name `main` then click the OK button.



As a result of this step, VisualProbe Q.E.D. runs this brief portion of the example code immediately. Execution stops at `main`, and the Tracing window displays the trace buffer contents.

```

Tracing [16554]
[16543] 000F8146 DONE <- <FETCH>
[16544] MOV SP, OFFSET DGROUP:stack_top
[16544] 000F8148 EC2050 mov sp, 5020E
[16544] 000F8148 20BC <- <FETCH>
[16545] STI ; Enable interrupt now
[16545] 000F814B FB sti
[16545] 000F814A FD50 <- <FETCH>
[16546] CALL _main ; Pass control to C main() function
[16546] 000F814C 9ADE0024F8 call _main
[16546] 000F814C DE9A <- <FETCH>
[16547] 000F814E 2400 <- <FETCH>
[16548] HLT
[16548] 000F8151 F4 hlt
[16548] 000F8150 F4F8 <- <FETCH>
[16549] 000F8152 FDEB <- <FETCH>
[16550] 0000541E F800 -> <MEM_WR>
[16551] 0000541C 0151 -> <MEM_WR>
[16552] 000F831E 8BCC <- <FETCH>

```

Figure 4-6. The Tracing Window

### The Tracing Window and Trace Controls

The first column of the Tracing window displays the physical bus cycles (blue background) that are related to the last Trace operation. Tracing window information appears in ascending numerical order, beginning at cycle 0000. This is the Top of the Tracing window. The last cycle written is at the Bottom of the Tracing window.

In addition to physical bus cycles, the Tracing window can display source code (white background) and instructions (gray background). You can specify the contents of the Tracing window display and make other choices for Trace functions by using the toolbar at the bottom of the Tracing window.



Figure 4-7. The Tracing Window Toolbar

**Note:** Notice that *while the Trace window is open*, an access to Trace function controls is also made available in the Main menu.



- **Stop/Start** enables or disables the trace buffer to capture execution information
- **Clear** clears the Trace buffer and Tracing window
- **Collect...** opens the Tracing Controls dialog box:



**Figure 4-8.** The Tracing Controls Dialog Box

The Tracing Controls dialog box lets you choose how Trace data is collected, and how the system should manage the Trace buffer for your current project. You can disable the timestamp or set the timestamp resolution, set limits on Trace functions according to Trace buffer status, and clear the Trace buffer upon each execution.



- **Show...** opens the Change Trace Display dialog box.



**Figure 4-9.** Change Trace Display Dialog Box

This dialog lets you specify what kind of information (Show Mode) is displayed in the Trace window, set the Timestamp Display resolution, and choose between off/binary/hexadecimal display of the Logic State Input (LSI) information.

- **Top** scrolls the view to the oldest trace information - displayed at the top of the Tracing window
- **Bottom** scrolls the view to the most recent trace information - displayed at the bottom of the Tracing window
- **Index-/Index+** toggles the relative positions of the Top and Bottom views in the Tracing window, and toggles the numbering of physical bus cycles between an ascending (n+1) and descending (n-1) numbering scheme (the first cycle is always n=0).



### Exercise 3: Hardware Events

*In this exercise, you will define a Hardware Event that when detected, causes VisualProbe Q.E.D. to take a specific action.*

*To run this tutorial in VP Q.E.D., use the 186 sample code file [Timerc.omf](#) included on your BeaconSuite installation CD*

In many instances it may be helpful during execution of your source code to have VisualProbe Q.E.D. detect specific logical conditions related to system hardware that cause the debugger to perform specific actions that would be impossible (or frighteningly inconvenient!) to perform manually. For example, it may be useful in your debugging procedure to have the Tracing function initiated if any address information flows through the bus that is greater than FFFFFFFH.

Several of VisualProbe Q.E.D.'s features - including Tracing and the Interval Timer - can be activated by establishing a set of logically-related terms that operate on data flowing through the bus called **Defined Hardware Events**. Via the Breakpoints dialog box, you can construct a set of **Defined Hardware Event Terms** that cause VisualProbe Q.E.D. to perform subsequent actions before the next bus cycle is permitted.

Defined Hardware Event Terms consist of two elements:

1. **Conditions** that include your specifications for the logical evaluation of processor cycle types, address and data types, processing levels and any of 16 channels of Logic State Input.
2. **Actions** that result when the logical evaluation of **Conditions** is satisfied. Actions can include breaks in execution, event counter increments, Interval Timer on/off, event level toggling, and Tracing functions. VisualProbe Q.E.D.'s hardware event system can also initiate multilevel decision tree functions.



## Create a Breakpoint for a Hardware Event

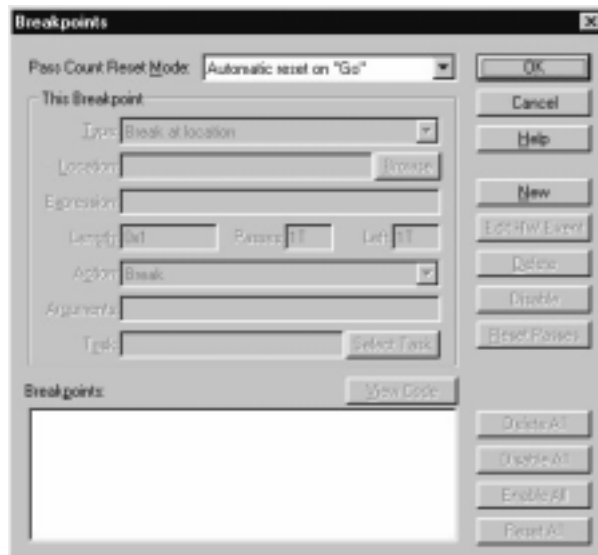
To define a hardware event, start by opening the Breakpoints dialog box. To be active, hardware events **MUST** always be associated with a breakpoint, whether or not your hardware event is intended to break code execution.

In this case, our intent is to detect a “Write to Address” event for a specific variable. The action that results is a break in execution and a trace event that captures the cycle to which the symbol address was written.

1. Open the Breakpoints dialog box:
  - Select View→Breakpoints from the Main menu. Or,
  - Click the Breakpoints button in the toolbar.



The Breakpoints dialog box appears:



**Figure 4-10.** The Breakpoints Dialog Box

2. In the column of buttons that appear on the right side of the Breakpoints dialog box, click New to create a new breakpoint.



- The default breakpoint Type is Break at location. However, we want to create a Defined Hardware Event. Note that the current Breakpoint: list is empty. Click the down arrow on the right side of the Type drop-down listbox. From the Type list, select Hardware Event. If no breakpoint of this type already exists in the Breakpoint: list, VisualProbe Q.E.D. automatically creates one.
- To further define the hardware event, click the Edit HW Event button on the right.

The Hardware Event dialog box opens:

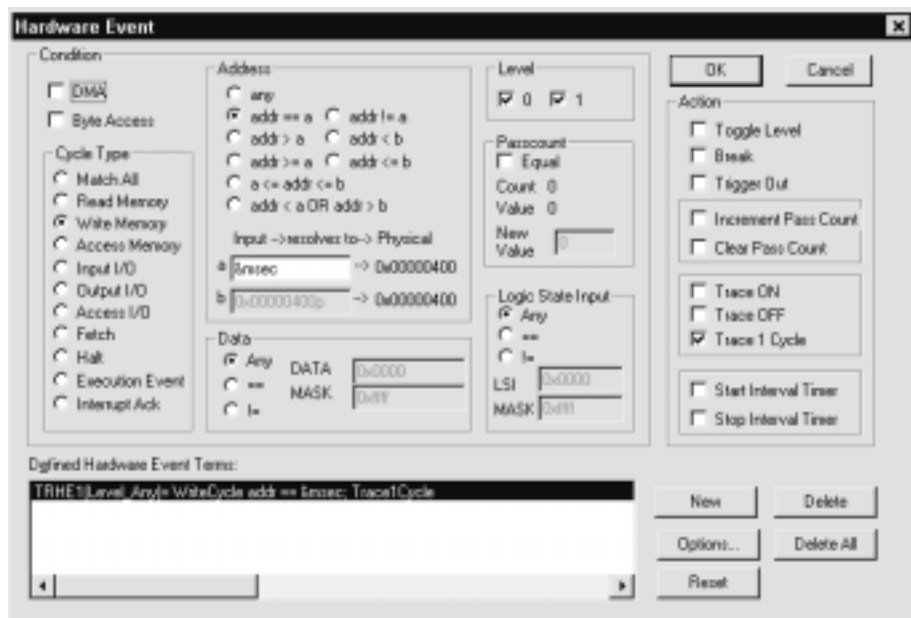


Figure 4-11. Hardware Event Dialog Box

- The Hardware Event dialog box is divided into two general sections: Conditions on the left that result in Actions on the right. In this example, we wish to have VisualProbe recognize a write-to-memory operation. Therefore, in the Condition section, under Cycle Type, click the Write Memory radio button.



6. In the Address section, click the `addr == a` radio button.

**Note:** For these logical operations, we have used the logical notation “==” to mean “equal to”, as the case would be in C++ programming language conventions, rather than a single “=”.

The Address section of your Hardware Event will look like this example when it is completed in the next step:



**Figure 4-12.** The Address Section of the Hardware Event Dialog Box

7. Click within the `a` data entry box, or press the Tab key once to move the focus to the `a` data entry box. You can either:

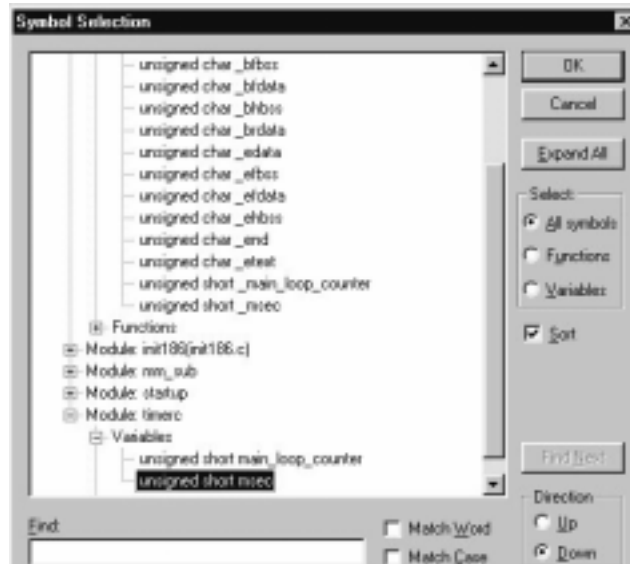
Enter the value `&msec&msec` using your keyboard,

OR

You may click the Browse... button to browse for this symbol from your source code.



When you click the Browse... button, the Symbol Selection dialog box opens.



**Figure 4-13.** The Symbol Selection Dialog Box

8. From the hierarchy of symbols and functions represented in the Symbol Selection dialog box, you must choose the symbol *unsigned short msec* from the module **timer.c**. Symbols preceded by a box can expand [+] or collapse [-] to reveal more or less of the structure, a function that operates in a way that is similar to Windows Explorer's file management application.
  - Click to expand Program > Module Timerc[timerc.c] > Variables
  - Then highlight the item unsigned short msec; then
  - Click OK to select *msec* as the value of variable *a*.

The symbol name (*msec*) now appears in the data entry box as the value of variable *a*.
9. In the Data section, click the Any radio button. We wish to look at ALL Data types.
10. In the Level section, click the check boxes to add a check to both the 0 and 1 conditions.



11. In the Logic State Input section, click the Any radio button to keep this setting nonspecific.
12. In the Action section, check the Break and Trace 1 Cycle boxes.
13. Note that VisualProbe automatically constructs a trigger statement in the Defined Hardware Event Terms field at the bottom of the dialog box as you make your selections. Take a moment to look at the TRHE0 (e.g.: `TermsforHardwareEvent0`) terms for this hardware event.



**Figure 4-14.** Example of Defined Hardware Event Terms

14. Click the OK button to save your Defined Hardware Event Terms and close the Hardware Event dialog box.
15. Click the OK button in the Breakpoints dialog box to save the association of the hardware event with the current breakpoint.

### Set Up the Trace Function

1. Open the Tracing window - if it is not already open.
2. Click the Collect... button at the bottom of the Tracing window.
3. In the Tracing Controls dialog box, make sure that a check *does not* appear in the Clear Trace on Each Execution box, then click the OK button.
4. Make sure that Tracing is turned off, because we want the Trace function to be initiated when the Hardware Event occurs. If the Start/Stop button in the lower left of the Tracing window says Stop, then click the button to turn off Tracing. If it says Start, then tracing is already off.
5. Select Debug→Go from the Main window menu. A break will occur - because you have specified in your Defined Hardware



Event Terms that a break *should* occur *after* one cycle has transpired.

The display should look something like the example below. Note only the write cycle corresponding to the address equal to 0x424 (Symbol = msec) is displayed for this example, based on a 186ES target CPU. *Note that this address is dependent upon the target CPU.*



**Figure 4-15.** Tracing a Memory Write Cycle

This exercise continues into the one that follows, when you will create Defined Hardware Event Terms that count write cycles and break execution on their tenth occurrence. If it's possible, don't discard your current tutorial settings.



## Exercise 4: Multiple Breakpoints and Multiple Defined Hardware Event Terms

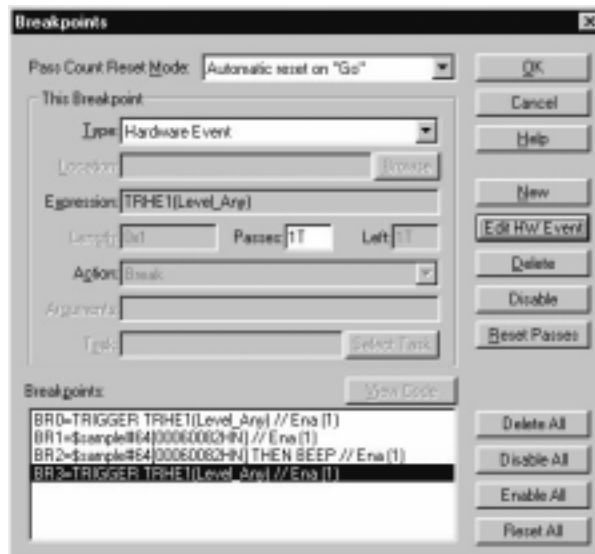
*In this exercise you will create multiple breakpoints; select a breakpoint for editing; create a list of Defined Hardware Events; and change their associations with the currently-selected breakpoint.*

*To run this tutorial in VP Q.E.D., use the 186 sample code file [Timerc.omf](#) included on your BeaconSuite installation CD.*

This exercise is a continuation of the preceding exercise. It demonstrates how multiple Breakpoints and Defined Hardware Events can be selected for editing, and helps illustrate the association between them.

VisualProbe Q.E.D. maintains your breakpoints in an ordered list that appears at the bottom of the Breakpoints dialog box. When you wish to edit the settings for any existing breakpoint you must first select (highlight) that particular breakpoint in the list by clicking on it.

Any subsequent changes that you make in the Breakpoints dialog box or in the Hardware Events dialog box *will be assigned to that selected breakpoint and will be saved to your project when you click OK.*



**Figure 4-16.** A List of Existing Breakpoints



## Multiple Hardware Event Terms

VisualProbe makes it possible to create a *list* of multiple Defined Hardware Event Terms in the Hardware Event dialog box. Only *one* line from this list of triggers can be associated with the breakpoint. Therefore, when you have a list of multiple triggers in the Hardware Event dialog box, be certain that you have selected the correct set of Defined Hardware Event Terms from the list before you associate it with the breakpoint by pressing the OK button.

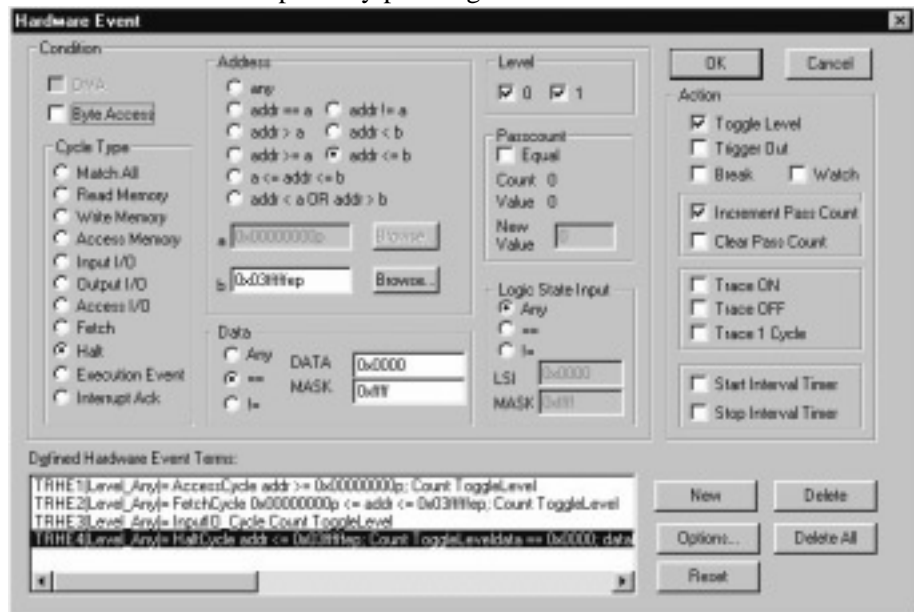


Figure 4-17. A List of Defined Hardware Event Terms (Triggers)



If your Hardware Event dialog box contains multiple Defined Hardware Events - the hardware event definition that is highlighted when you click the OK button is the one that becomes associated with the breakpoint. If the hardware event definition that is selected when you *close* the Hardware Event dialog box has been changed from the one that was associated upon *opening* the dialog box, VisualProbe displays the following message:



**Figure 4-18.** Change of Association Warning - Hardware Events

1. Reopen the Breakpoints dialog box. Note that the Type menu defaults to the last type of breakpoint chosen—Hardware Event.
2. To create a *new* Hardware Event, click New.
3. With BR1 highlighted in the Breakpoints field, click the Edit HW Event button.
4. Click the New button in the Hardware Event dialog box.

**Note:** The Defined Hardware Event Terms window now has two TRHE definitions. This window allows you to monitor the Hardware Event resource use in your debug session.

This example will cause a break in execution on the 10th occurrence of the previous event. Set up the new Defined Hardware Event Terms:

5. Click the Match All radio button under Cycle Type and the Any radio buttons under Address, Data, and Logic State Input.
6. Check the Equal box under Passcount and enter 10 into the New Value field.
7. Check the Break box under Action.
8. Check the Clear Passcount box under Action.



9. Uncheck the Trace 1 Cycle box under Action.

**Note:** Notice that the Trace 1 Cycle box already contained a check? When a new Hardware Event is created, it is duplicated from the currently selected Defined Hardware Event Terms.

10. Click the Add button and notice the change in the Defined Hardware Event Terms field.

11. Click the OK button to close the Hardware Event dialog box and return to the Breakpoints dialog box.

12. Select BR0 in the Breakpoints field.

13. Click the Edit HW Event button.

14. In the Action section of the Hardware Event dialog box, check the Increment Pass Count box.

15. Click the Modify button. The dialog box should look like this example:



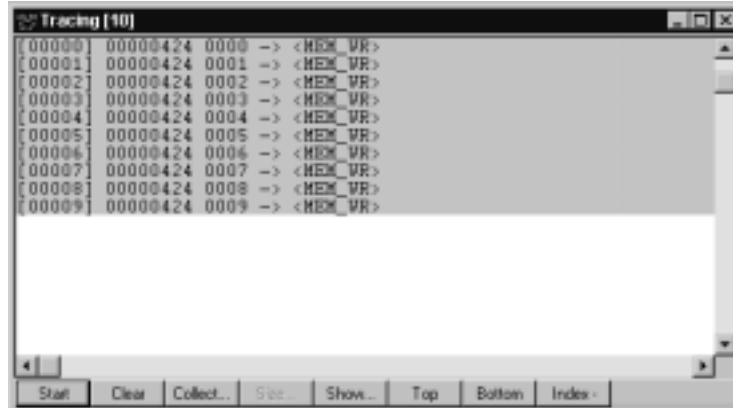
Figure 4-19. The Modified Defined Hardware Event Terms



16. Click the OK button to close the Hardware Event dialog box.
17. In the Breakpoints dialog box, click the OK button.
18. From the tool bar, click the Reset button and then the Go button.

Execution stops after the first 10 occurrences of the data write to msec. The Tracing window should look like the example shown in Figure 4-20.

**Note:** Your tutorial results *may* include one extra trace cycle, due to the nature of the 186 processor.



**Figure 4-20.** The Tracing Window After Complex Breakpoint Execution



## Exercise 5: Real Time Watch

*In this exercise you will learn what a Real Time Watch can do for your debugging technique - and how to set one up in VisualProbe Q.E.D.*

*Use this 186 sample code file:  
[Timerc.OMF](#)*

You may find that it is helpful to observe the values of the symbols contained in your code as it executes. Real Time Watch can provide you with the information you need.

Real Time Watches are based on regularly-occurring comparisons that are made between:

- A *value at a specific hardware address* (a symbol you want to watch) and
- *Data* that is flowing through the bus

These comparisons result from Defined Hardware Events that you will create. Defined Hardware Events let you select a particular type of data and/or cycle - then define a logical comparison between that data and your choice of one or two variables. Both variables' values can be set by browsing for any of the symbolic functions or variables that are contained in your code.

When the terms of your Hardware Event Definition are satisfied (true), it triggers an action: "Watch". The result of the logical comparison is then displayed in the Real Time Watch window.

You must also specify the frequency at which this comparison (*polling*) occurs. Therefore at runtime, the logical comparisons in the Hardware Event Definition occur at the Polling Frequency that you have specified in the Real Time Watch window.

Creating a Real Time Watch has two basic steps:

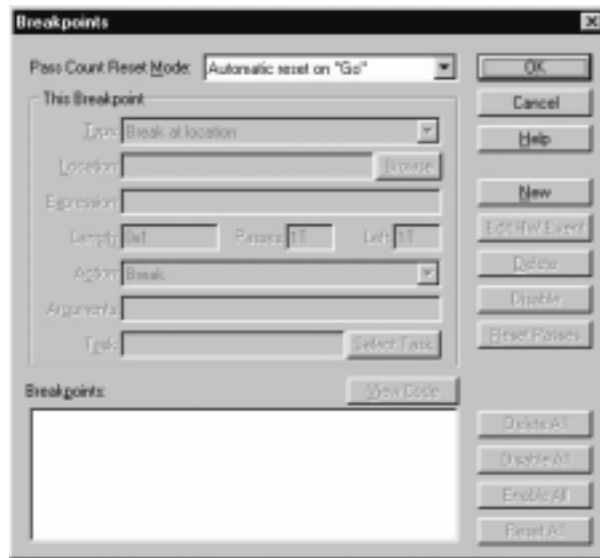
1. Create a Hardware Event Definition
2. Set the Polling Frequency



## Create the Hardware Event Definition

The purpose of this Hardware Event Definition is to create a Real Time Watch that identifies whether *any* bus cycle contains *any* address data that is greater than the value of the symbol *msec*.

1. Click View>Breakpoints to open the Breakpoints dialog box.

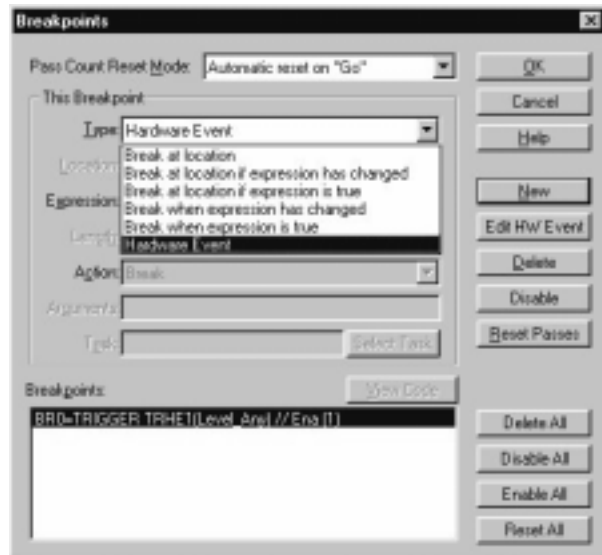


**Figure 4-21.** The Breakpoints Dialog Box (Default)

2. Click New to create a new Hardware Event Definition.



3. Select Hardware Event from the Type: drop-down list box.



**Figure 4-22.** Creating a New Hardware Event

4. Click Edit HW Event to open the Hardware Event dialog box.
5. Select the Conditions that describe your Hardware Event. Click the radio buttons that correspond to these terms:

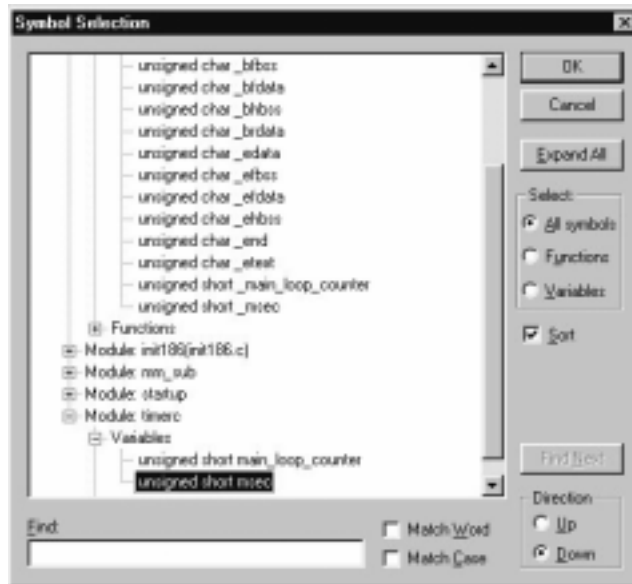
*Cycle Type:* Match All

*Address:* addr → a

*Note that when you click this radio button, a definition box becomes un-grayed in which you must enter the value of variable a.*



- a: To enter the value of a symbol in the data entry box associated with variable *a*, click the Browse... button and the Symbol Selection dialog box opens.

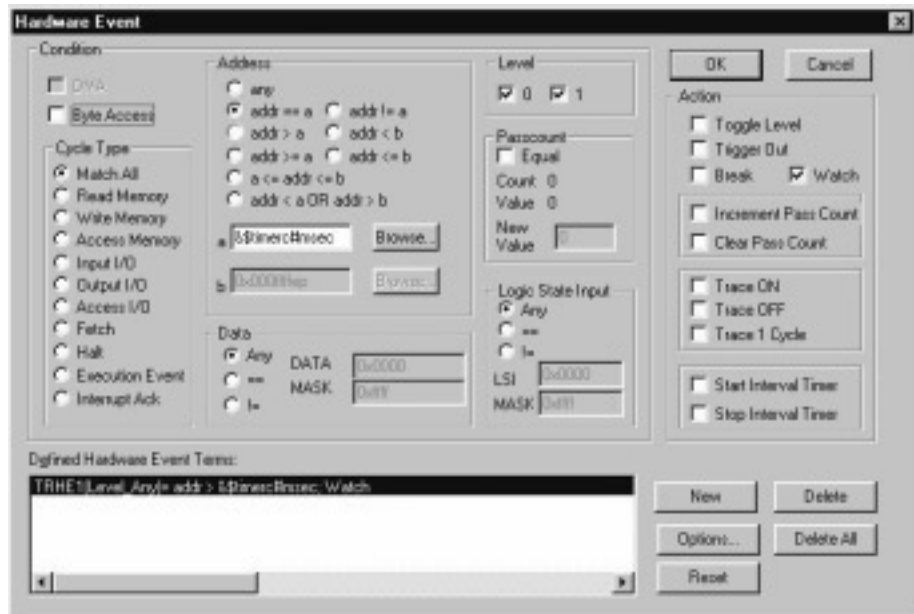


**Figure 4-23.** The Symbol Selection Dialog Box

6. From the hierarchy of symbols and functions represented in the Symbol Selection dialog box, you must choose the symbol `unsigned short msec` from the module **timer.c**. Symbols preceded by a box can expand [+] or collapse [-] to reveal more or less of the structure, a function that operates in a way that is similar to Windows Explorer's file management application.
  - Click to expand Program > Module Timerc[timer.c] > Variables
  - Then highlight the item `unsigned short msec`; then
  - Click OK to select `msec` as the value of variable *a*.



**Note:** The symbol `msec` from the `timerc` module now appears in the data entry box as the value of variable `a`.



**Figure 4-24.** The Real Time Watch Window

- Complete the Action (the function that VisualProbe Q.E.D. will carry out subsequent to the Condition settings in your Hardware Event Definition. Choose the following settings for the remaining sections of the dialog box.

**Data:** Any - VisualProbe examines Any type of data on the bus

**Level:** Place a checkmark in both boxes (1 & 2) to include all program levels

**Passcount:** Leave blank

**Logic State Input:** Any

**Action:** Because we are creating settings for a Real Time Watch, place a checkmark in the Watch box, and leave the remaining Action checkboxes blank



**Note:** Observe that the *Defined Hardware Event Terms* box now contains the logical statements that support your Real Time Watch.

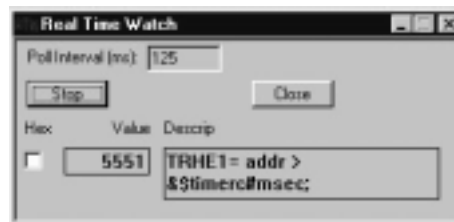
**Warning!** It is possible to create multiple Hardware Event Definition Terms - but only ONE may be associated with the current Breakpoint. Therefore **ALWAYS** make sure that the appropriate Hardware Definition Terms are highlighted before you close the Hardware Event Definition dialog box.

8. Click OK to associate the Defined Hardware Event terms that are currently highlighted with the Breakpoint function.

**Note:** The breakpoint with which a Hardware Event is associated *is not indicated* by a red dot in the Code window, because a Hardware Event can occur independently of CS:IP.

### Set the Polling Frequency

The polling interval for Real Time Watch can be set in 1ms increments.



**Figure 4-25.** Real Time Watch Window

1. Open the Real Time Watch dialog box by selecting View→Real Time Watch.
2. Click in the Poll Interval (ms) field and enter 125ms as the frequency setting.
3. If you wish to observe the Value as a hexadecimal number, click to place a check in the Hex checkbox. To toggle back to decimal values, click within the checkbox to remove the check.



4. Click Debug→Go to start the debugger.
5. Click Start in the Real Time Watch window start the Real Time Watch function. Notice that the Start button's label has toggled and reads: Stop.
6. You may stop the Real Time Watch while the debugger continues to execute your code. To do so, click Stop.



## Exercise 6: Interrupt Registers

*In this exercise, you will open the Interrupt Control Unit window, set the view to binary, and mask an interrupt register.*

*To run this tutorial in VP Q.E.D., use the 186 sample code file [Timerc.omf](#) included on your BeaconSuite installation CD*



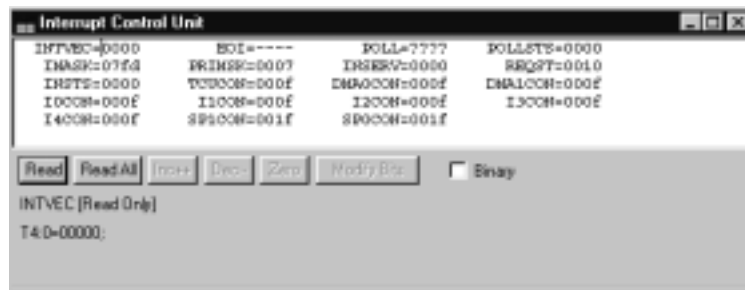
VisualProbe Q.E.D. provides access to the 186 processor's Interrupt Mask Registers through the Interrupt Control Unit.

Keep in mind that there is a wide range of variation in implementation of interrupt registers among the chips supported by VisualProbe Q.E.D. In particular, protected-mode processors have inherent capabilities that may require additional inspection and management techniques available through the Interrupt Descriptor Table view provided in VisualProbe Q.E.D.386.

To view the Interrupt Control Unit for your 186 processor:

1. Open the Interrupt Control Unit window:
  - Select View→Interrupt Control Unit in the main menu. Or,
  - Click the Interrupt Control Unit button in the toolbar.

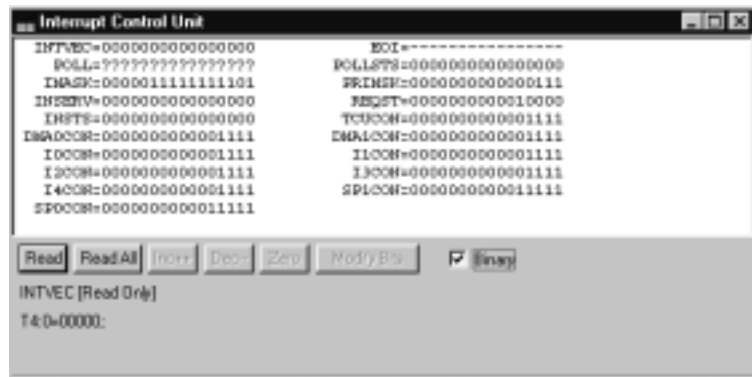
The default view shows the values within each register in hexadecimal form:



**Figure 4-26.** The Interrupt Control Unit - Hexadecimal Display

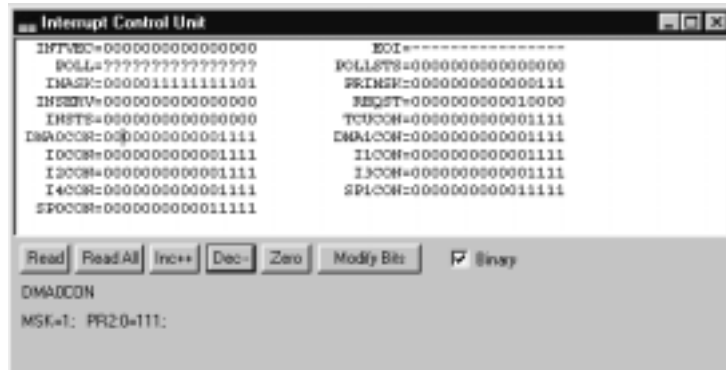
2. You can select a binary view of the registers. Check the Binary check box.





**Figure 4-27.** Binary View in the Interrupt Control Unit Window

3. You can modify register values.
  - Click within the DMA0CON register’s value field to “select” it. There is no highlighting to indicate selection - placing your cursor in the register’s field directs the current selection focus. Note that additional controls are now activated in the bottom half of the Interrupt Control Unit.



**Figure 4-28.** Masking an Interrupt Register

**Read** Causes VisualProbe to read the contents of the currently-selected register.



- Read All Causes VisualProbe to read the contents of ALL registers.
- Inc++ Causes an incremental change in value based on the default boundaries for the selected register.
- Dec-- Causes an incremental change in value based on the default boundaries for the selected register.
- Zero Causes the value of the register to become 0 (zero).
- Click the Modify Bits button, and the Modify Register Bits dialog box appears, with the current settings for the DMA0CON register. This dialog box opens and displays controls that affect only the register that was selected in the Interrupt Controller Unit dialog box.



**Figure 4-29.** Setting a Register Mask for DMA0CON

4. To mask this register, select the MSK item from the Bits drop-down listbox. The only possible value for the MSK bit is “1” - which sets the mask. Click OK to set the mask.
- Note:** Note that the bit selection available relates directly to the register you have chosen to modify. You must consult the chip manufacturer’s documentation for register-specific information.
5. Notice that when you make changes to the registers’ values, they become highlighted - they are displayed in blue text.



## Exercise 7: The Interval Analyzer

Learn about how the Interval Analyzer works to help you improve the performance characteristics of your code!

To run this tutorial in VP Q.E.D., use the 186 sample code file [Timerc.omf](#) included on your BeaconSuite installation CD.

Would you be interested in statistical performance analyses of your applications? For instance, would you like to know what percentage of runtime does a CPU expend performing a particular routine within your code? If you think this might be important, and if you would like to know more about how to analyze the timing characteristics of your code, then this tutorial is for you!

Many developers understand that maintaining good application performance analysis marks is as important as any other part of the debugging process. Excellent performance time goes hand-in-hand with clean-running code, and in many quarters slow application performance is now recognized as a bug. To help you identify instances in which your code may need some streamlining, VisualProbe Q.E.D.'s Interval Analyzer provides a “two-timing” approach to performance analysis that can measure two very significant intervals of time:

- The *InInterval* = the interval of time when your routine is running.
- The *OutInterval* = the interval of time when your routine is *not* running.

Let's say that you will use *Timer A* to measure all of the *OutIntervals* - those intervals when your routine is not running; and you will use *Timer B* to measure all of the *InIntervals* - which are those intervals when your routine *is* running. You will instruct the Interval Analyzer to use hardware events to measure these intervals:

- Turn off *Timer A* and turn on *Timer B* when your routine starts
- Turn off the *Timer B* and turn on *Timer A* when your routine stops

So, how is this useful? By adding the amounts of time recorded by *Timers A* and *B*, we know the total amount of time. By dividing the value of *Timer B* (your routine's running time) by the total amount of time, you can know the *interval ratio*. The interval ratio represents the percentage of CPU time spent processing your routine.



### How to Set Up the Interval Analyzer

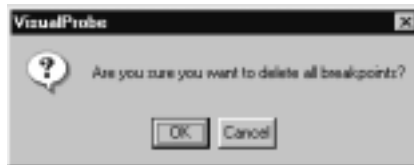
Operation of the Interval Analyzer is intimately tied to *Hardware Events*, and from our earlier exercises you will recall that you must define hardware events using the Breakpoints dialog box. Every hardware event must be associated with a breakpoint.

Before any results can be viewed in the Interval Analyzer display, you must create two separate hardware events - one for each timer. Therefore you must also create two breakpoints; one for each timer in this two-timer system. Begin by opening the Breakpoints dialog box:



1. Open the Breakpoint... dialog box, using the Breakpoints... icon, or select the Breakpoints... item in the View menu.
2. If any breakpoints exist from earlier exercises, click the Delete All button to get rid of them. Especially those that your co-workers created while you were out at lunch.

*VisualProbe Q.E.D. inquires whether you're sure that this is what you want to do. Of course it is. We just told you to do it!*



**Figure 4-30.** A Polite Message from VisualProbe Q.E.D.

3. Click OK.
4. Click New to create a new breakpoint.
5. Click the drop-down listbox button for the breakpoint Type:, and select Hardware Event from the list.
6. Click Edit HW Event to open the Hardware Event dialog box.
7. If any Defined Hardware Event Terms exist from earlier exercises, click the Delete All button to get rid of them.

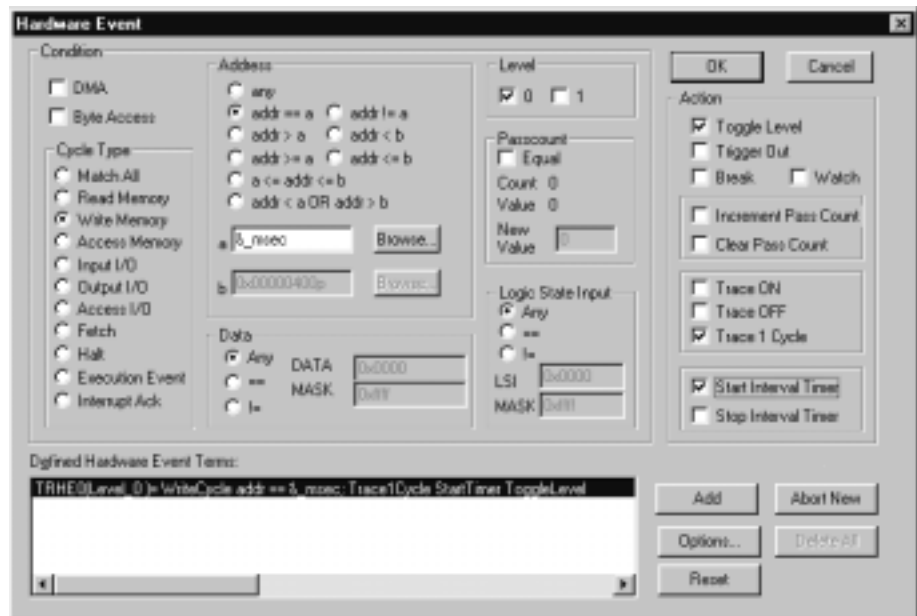


In the example code that you have loaded for this tutorial for your 186 processor, there is a GLOBAL variable called `msec`. You will set up the first of two sets of Defined Hardware Event Terms (TRHE0) that fires whenever a Write Memory cycle with a value equal to the value of the variable `msec` flows through the bus. These terms (TRHE0) cause the following Actions:

- Toggle Level (start on Level 0)
- Trace 1 Cycle
- Start Interval Timer

Your Hardware Event dialog box will look like this example when you have entered this information.

**Note:** Use the Hardware Event dialog box's Browse... button to find the Global variable `msec` with the Symbol Browser.

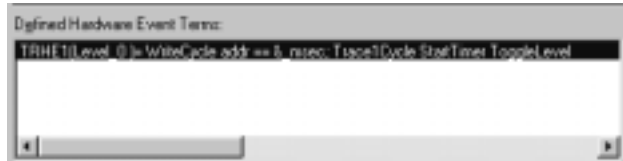


**Figure 4-31.** Defined Hardware Event Terms (TRHE0)



8. When you have carefully entered these settings, click the Add button to finalize this set of Defined Hardware Event Terms.

**Note:** Notice that the designation TRHE0 (0=unconfirmed) for this set of terms has changed to TRHE1, and is now a valid set of terms that can be assigned to a breakpoint.

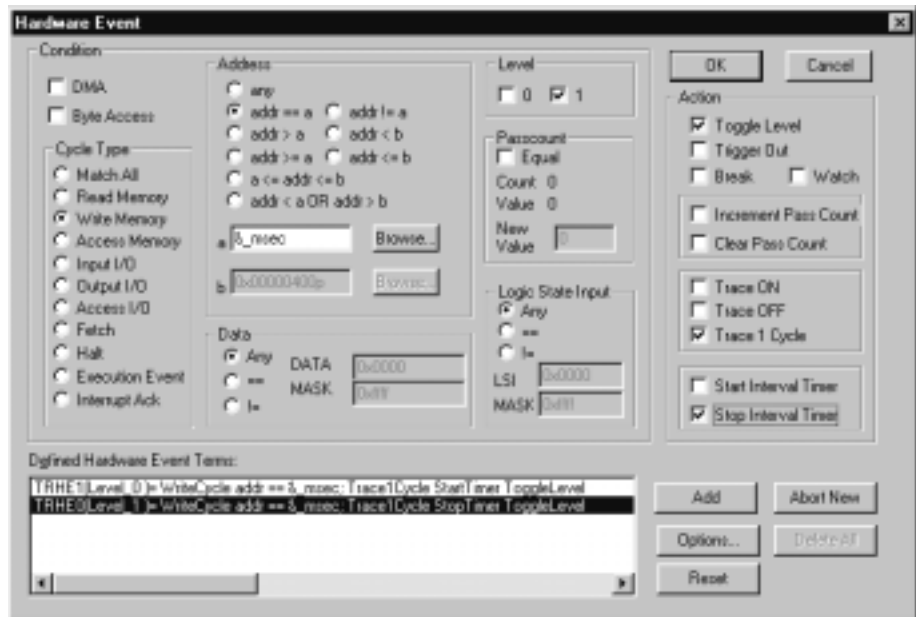


**Figure 4-32.** Defined HWE Terms are Valid When  $TRHE_n > 0$

9. Now you must create a second set of Defined Hardware Event Terms for the second timer. Click the New button to create a new set of Defined Hardware Event Terms. A new unconfirmed set of terms (TRHE0) appears in the Defined Hardware Event Terms list. Notice that this new set is based on the previous set of Terms. Terms that are labeled (TRHE0) are always the newest set of terms. This label indicates that they have not yet been validated or “added” to the list.
10. Edit the settings of the Hardware Event dialog box for the new set of terms (TRHE0). The Conditions and Actions for this second set of terms must define: Any Write Memory cycle having a value equal to the variable *msec* occurring on Level 1 (the first HWE was for Level 0) will cause the following Actions: Toggle Level, Trace 1 Cycle, and STOP the Interval Timer.



Your Hardware Event dialog box will look like the following example. Notice that the new set of terms is still labeled TREH0 - which indicates that it is new, and that it has not yet been validated as part of the list.



**Figure 4-33.** New Defined Hardware Event Terms are Labeled TREH0 - Before Being “Added”

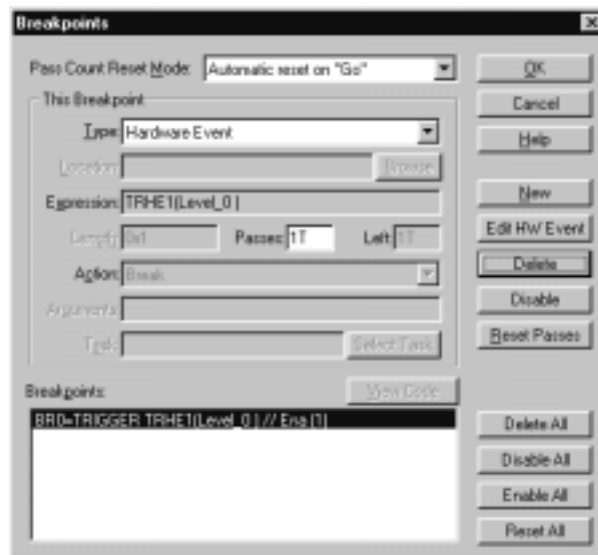
11. Click the Add button to validate the new Defined Hardware Event Terms. The set of terms becomes assigned to TRHE2.



**Figure 4-34.** The Completed Set of Defined Hardware Event Terms



12. Now that you have created the two sets of Defined Hardware Event Terms you must assign them to breakpoints.
13. Click (highlight) the first set of Hardware Event Terms (TREH1). Click the OK button, and the Hardware Events dialog closes. The Breakpoints dialog box remains open. *Notice that there is still only one breakpoint in the list (BR0), and it is associated with Hardware Event Term (TREH1).*



**Figure 4-35.** A Second Breakpoint is Needed

- You must create the second breakpoint (BR1) to which you will assign the second set (TRHE2) of Defined Hardware Event Terms
14. Click New to create the second breakpoint. The first breakpoint is already associated with TRHE1, therefore, we need to associate this new breakpoint with TRHE2.
  15. Click Edit HW Event to reopen the Hardware Event dialog box.

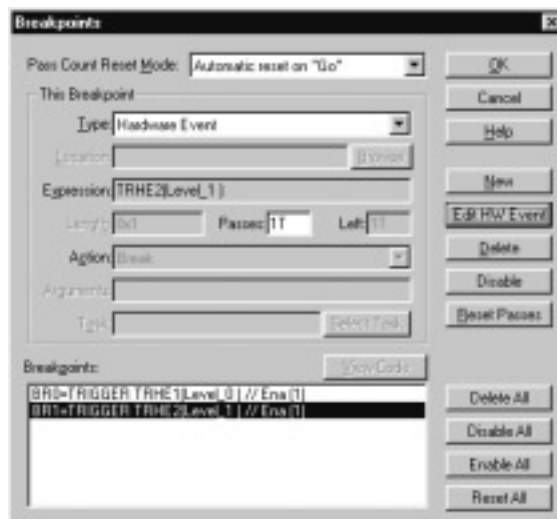


16. In the Defined Hardware Event Terms list, click on TRHE2, then click OK to close the Hardware Events dialog box. The second hardware event is now associated with the second breakpoint.
17. Click OK to close the Breakpoints dialog box. Both breakpoints are now associated with a different Hardware Event.

BR0 is associated with TRHE1

BR1 is associated with TRHE2

as shown in the example below:



**Figure 4-36.** Completed Association of Hardware Events with Breakpoints



## Open the Interval Analyzer and Set the Polling Interval

Now that the hardware events that control the operation of the timers have been associated with their breakpoints, you can open the Interval Analyzer Control, set the Polling Interval, and select a Time Base by which measurements will be taken.

Open the Interval Analyzer Control:



- In the View menu, select Interval Analyzer. *Or*
- Click the Interval Analyzer Control icon in the toolbar



**Figure 4-37.** The Interval Analyzer Control Dialog Box

The Interval Analyzer Control contains several components which you can tailor your analysis. They are:

- Poll Interval (ms): The frequency with which the Interval Analyzer checks for changes in states of the timers, based on the Timebase settings.

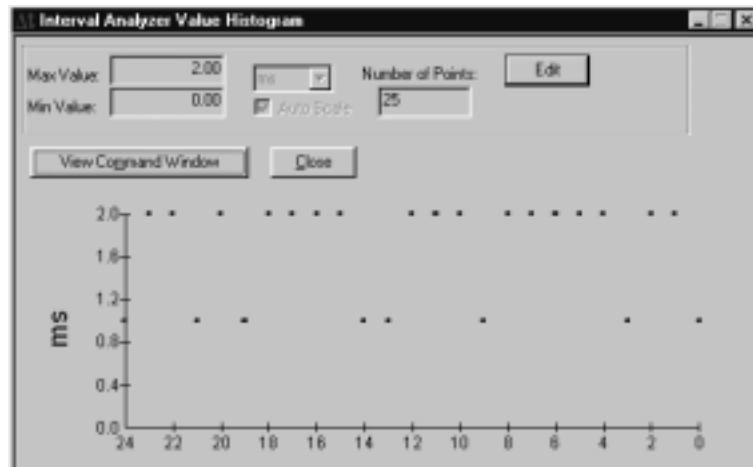


- Time Base: Check the box corresponding to the units of time on which you wish to base your analyses.
- Last Sample Value: This value is not editable by the user, it results directly from the operation of the Interval Analyzer, based on the validity of the last polled sample. Last Sample Value reports the value (in ms) of the most recent valid sample. A sample is reported as invalid if the timer is running when the poll begins or if the timer is not running and a previous poll has already reported the value. *If the polling interval is much less than the typical durations of both timer states, most of the polls will be invalid.*
- All Samples: Sample Size shows the number of valid samples on which computation of the results displayed in Max:, Min:, and Avg: are based.
- Moving Window Statistics: This section displays the same information as the All Samples section, but instead of being based solely on the total number of valid samples taken during the entire time that the Interval Analyzer was running, the results are based on user-selected sections of the Interval Analyzer's running time that are displayed in the Graphs section. The Interval Ratio represents the percentage of CPU time (based on the Sample Size) spent processing the *InInterval* - in the case of this example, *Timer A*.
- Graphs: These are moving-window graphic representations of polling as it occurs while the debugger is running your code. Three graphical statistic representations are available:



## The Interval Analyzer's Value Histogram

You can use this graph to visually discern patterns in the sample data.



**Figure 4-38.** The Interval Analyzer's Value Histogram

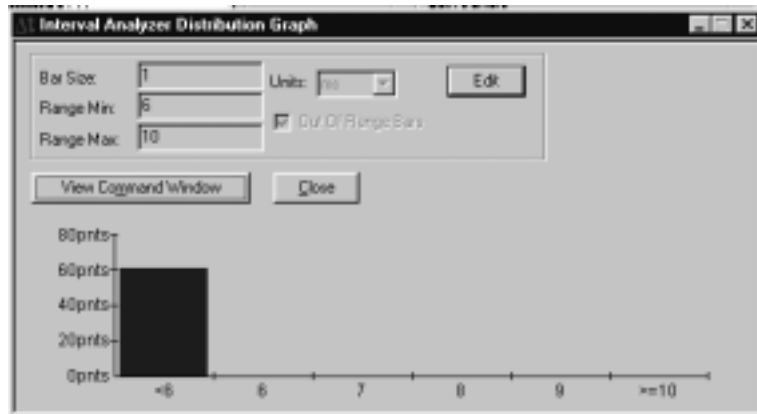
The Value Histogram plots the most recent samples' values on a graph. The graph's vertical axis shows results in milliseconds, and the horizontal axis reveals the number of Polls. The most recent sample is displayed on the right-hand side of the graph. All preceding values are moved one position to the left as each Poll occurs. As the number of samples taken exceeds the value you have entered in Number of Points:, the values move off the left-hand edge and are discarded.

**Note:** You can use the View Command Window button to conveniently recall the Interval Analyzer to the top layer of your current screen view.



## The Interval Analyzer's Distribution Graph

This type of graph will be useful for illustrating probability distributions for interval times.



**Figure 4-39.** The Interval Analyzer's Distribution Graph

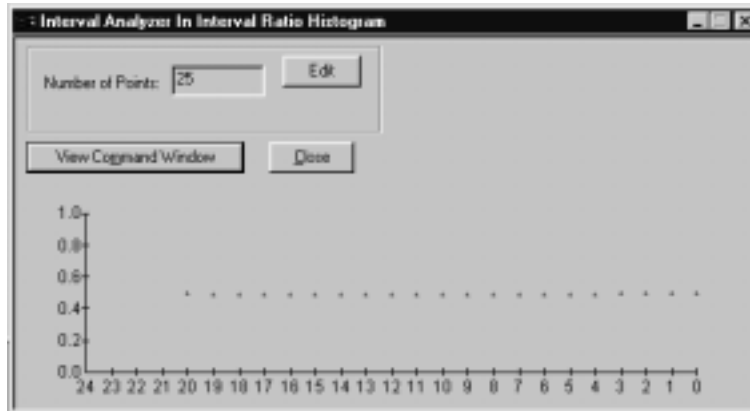
Note that the sample size can be increased beyond that which you have set in the Interval Analyzer Control window, to run continuously and collect as many data points as necessary for a credible result. You may edit the settings for Bar Size:, Range Min:, Range Max:, and Units:. The vertical axis displays the number of Polls, and the horizontal axis displays time in the Units that you have specified.

**Note:** You can use the View Command Window button to conveniently recall the Interval Analyzer to the top layer of your current screen view.



### The Interval Analyzer's Interval Ratio Histogram

This graph reveals the most recent values of the Interval Ratio in histogram form. It is useful for graphical displays of variations in CPU usage over time.



**Figure 4-40.** The Interval Analyzer's Interval Ratio Histogram

You may edit the number of Polls that will be included in the displayed results.

**Note:** You can use the View Command Window button to conveniently recall the Interval Analyzer to the top layer of your current screen view.

