Technical Notes

# NEC 78K Family In-Circuit Emulation

# Contents

# 1 Introduction

The NEC 78K development system is built around a powerful FPGA, which contains the 78K0 core and all the necessary debug logic in order to control the core and the peripherals. CPU peripherals are emulated by the 78K0 CPU (umbrella device), which contains all peripherals of the devices being emulated. The CPU is configured specific to every target CPU being emulated.

Note: The description in this document applies for the 78K0 ActivePRO PODs and does not apply for the old 78K0 Power PODs.

## Debug Features

- Unlimited execution breakpoints

- Access breakpoints

- 512KB overlay emulation memory

- Real-time access

- Fail-safe exceptions

- Trace

- Execution & Access Coverage

- Profiler

## 1.1 Differences from a standard environment

As soon as the POD is inserted in the target instead of the CPU, electrical and timing characteristics are changed. Different electrical and timing characteristics of used elements on the POD and prolonged lines from the target to the CPU on the POD contribute to different target (the whole system) characteristics. Consequently, signal cross-talks and reflections can occur, capacitance changes, etc.

Beside that, pull-up and pull-down resistors are added to some signals. Pull-up/pull-down resistors are required to define the inactive state of signals like reset and interrupt inputs, while the POD is not connected to the target. Because of this, the POD can operate as standalone without the target.

## 1.2 Common Guidelines

Here are some general guidelines that should be followed:

- Use external (target) Vcc/GND if possible (to prevent GND bouncing).
- Make an additional GND connection from the development system (iC3000HS) to the target to protect the emulator from being damaged due to different target and emulator ground potentials, at the time when the POD is plugged into the target.

# 2 Emulation Options

## 2.1 Hardware Options
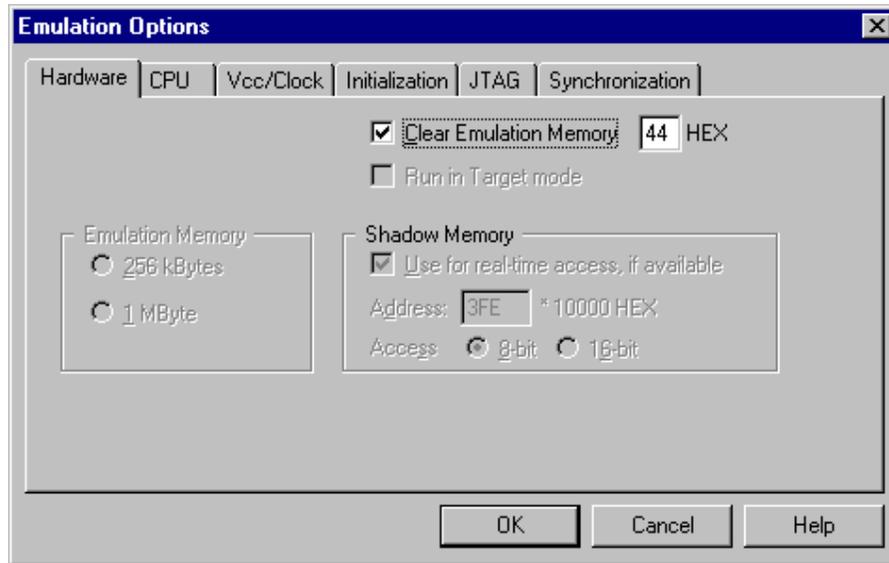


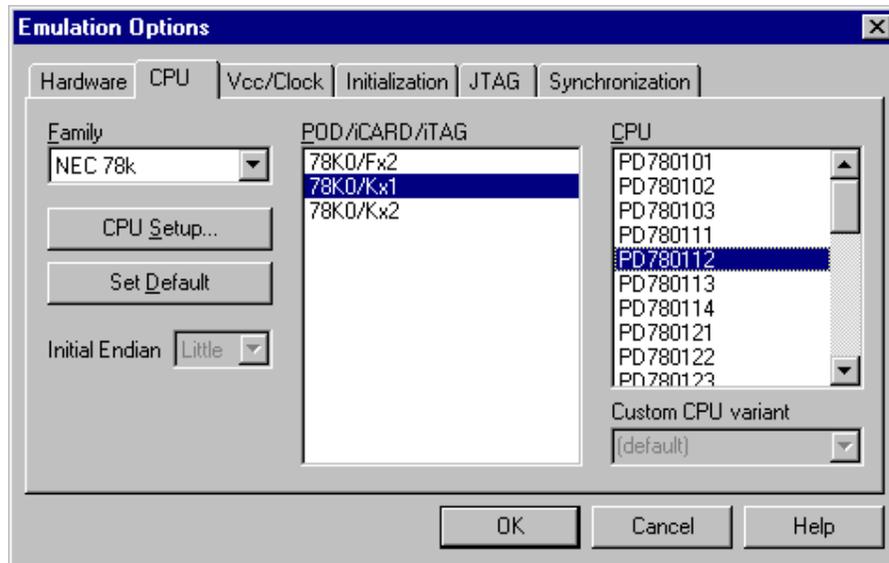*In-Circuit Emulator Options dialog, Hardware page*

### *Clear Emulation Memory*

This option allows you to force clearing (with the specified value) of emulation memory after the emulation unit is initialized.

Clearing emulation memory takes about 2 seconds per megabyte, so use it only when you want to make sure that previous emulation memory contents don't affect the current debug session.

## 2.2 CPU Configuration

With In-Circuit emulation besides the CPU family and CPU type the emulation POD must be specified (some CPU's can be emulated with different PODs).



*In-Circuit Emulator Options dialog, CPU Configuration page*

### CPU Setup

Opens the CPU Setup dialog. In this dialog, parameters like memory mapping, bank switching and advanced operation options are configured. The dialog will look different for each CPU reflecting the options available for it.
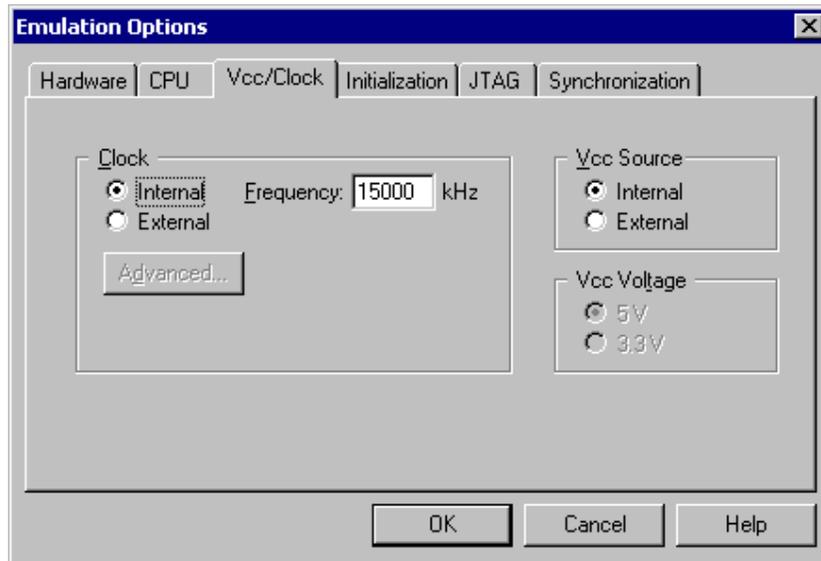
### Set Default

This button will set default options for currently selected CPU. These include:

- Vcc and clock source and frequency

- Advanced CPU specific options

Note: Default options are also set when the Family or a POD is changed.

## *2.3  Power Source and Clock*

The Vcc/Clock Setup page determines the CPU's power and clock source.



*In-Circuit Emulator Options dialog, Vcc/Clock Setup page*

Note: When either of these settings is set to External, the corresponding line is routed directly to the CPU from the target system.

### *Clock Source*

Clock source can be either used internal from the emulator or external from the target. It is recommended to use the internal clock when possible. When using the clock from the target, it may happen that the emulator cannot initialize any more.

It is dissuaded to use a crystal in the target as a clock source during the emulation.  It is recommended that the oscillator be used instead. Normally, a crystal and two capacitors are connected to the CPU's clock inputs in the target application as stated in the CPU datasheets. A length of clock paths is critical and must be taken into consideration when designing the target. During the emulation, the distance between the crystal in the target and the CPU (on the POD) is furthermore increased, therefore the impedance may change in a manner that the crystal doesn't oscillate anymore. In such case, a standalone crystal circuit, oscillating already without the CPU must be built or an oscillator must be used.

Note: The clock frequency is the frequency of the signal on the CPU's clock input pin. Any internal manipulation of it (division or multiplication) depends entirely on the emulated CPU.

### *Vcc Source*

Determines whether Emulator or the target system provides power supply for the CPU.

---

## *2.4 Initialization Sequence*

Typically, there is no need to use the initialization sequence on an In-Circuit Emulator, especially when the emulated target CPU is a single chip CPU. Primarily, the initialization sequence is used on On-Chip Debug systems to initialize the CPU after reset to be able to download the code to the target memory.

Besides enabling a disabled memory access upon reset, the initialization sequence can be used for instance to disable the CPU internal watchdog being active after reset or to modify any other CPU registers, when it's preferred to run the application with the modified CPU reset state.

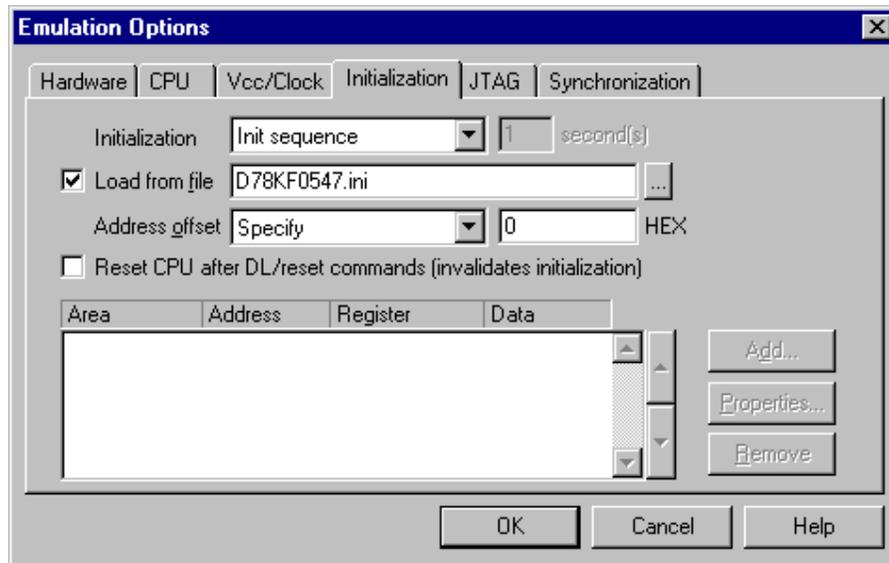The initialization sequence can be set up in two ways:

1.  Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.
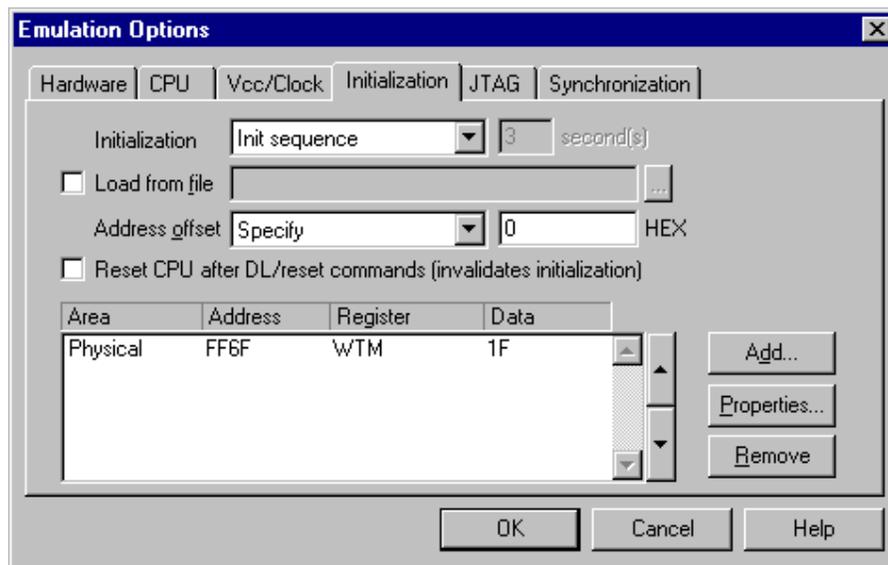


2.  winIDEA accepts initialization sequence as a text file with .ini extension. The file must be written according to the syntax specified in the appendix in the hardware user's guide.

    Excerpt from the sample D78KF0547.ini file:

    S WTM B 0x1F   //comment

---

The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

There is also a third method, which can be used too but it's not highly recommended for the start up. The user can initialize the CPU by executing part of the code in the target ROM for X seconds by using 'Reset and run for X sec' option.
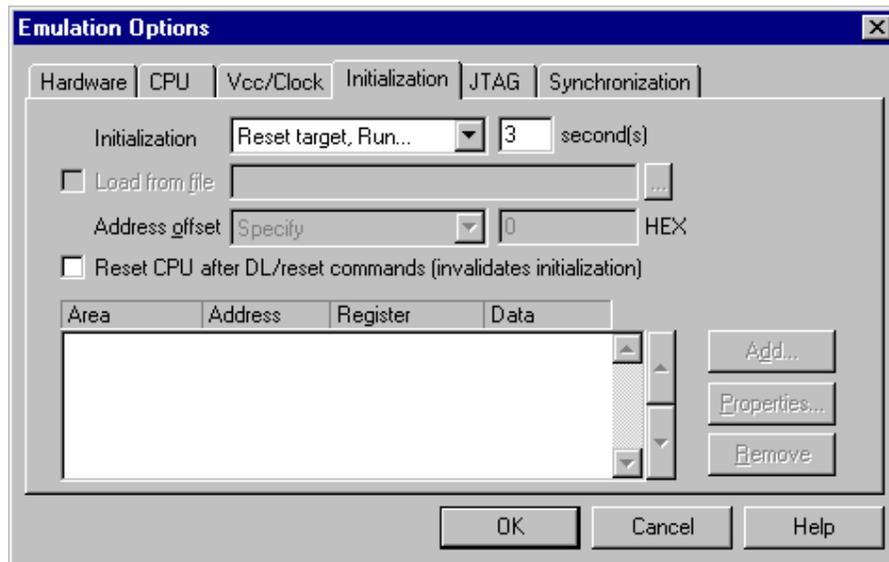
# 3  Setting CPU options

## 3.1  CPU Options

The CPU Setup, Options page provides some emulation settings, common to most CPU families and all emulation modes. Settings that are not valid for currently selected CPU or emulation mode are disabled. If none of these settings is valid, this page is not shown.
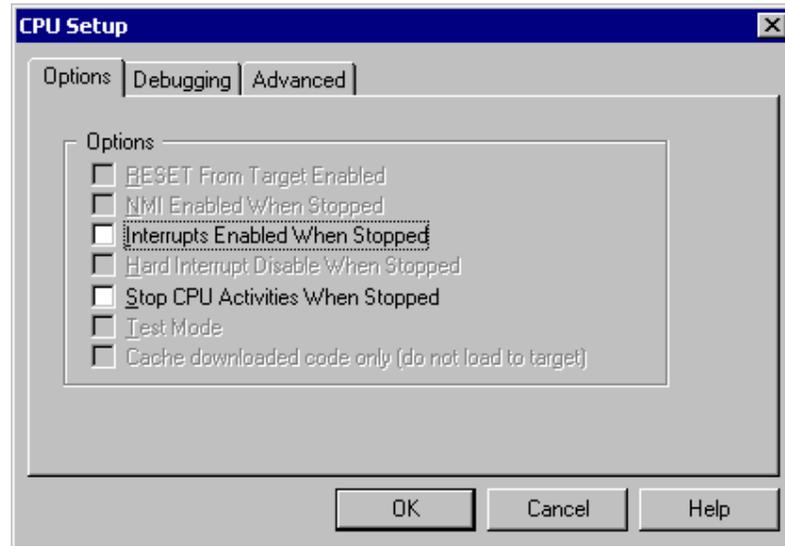


*CPU Setup, Options page*

### *Interrupts Enabled When Stopped*

The emulator itself doesn't support servicing interrupts while the application is stopped (interrupts in background). Setting of this option impacts only the CPU behaviour during the debug single step execution.

Default disabled option makes the emulator to mask the interrupts while the source step debug command is being executed, which yields more predictive behaviour in case of the application using interrupts. Interrupts are disabled through the IE flag in the PSW register hidden from the user.

If this option is enabled, the emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

### *Stop CPU Activities When Stopped*

When the option is checked, all internal peripheral modules, like timers and counters, are stopped when the user's program is stopped. Otherwise, timers and counters remain running while the program is stopped. Usually, when the option is checked, the emulation system behaves more predictable while stepping through the program. While being aware of the consequences, it is up to the user whether the option is checked or not.

## 3.2 Debugging Options
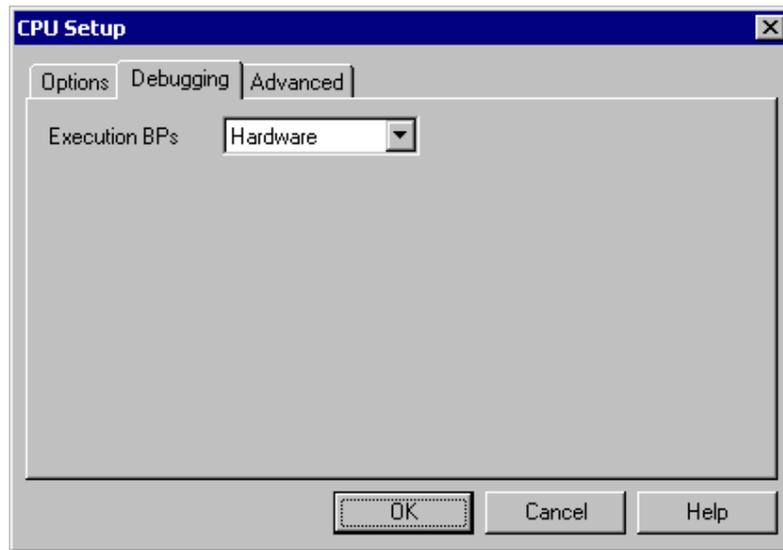


*NEC 78k Family Debugging Options*

### Execution Breakpoints

*Hardware Breakpoints*

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to sixteen. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writeable memory (ROM) or self-modifying code. If the option 'Use hardware breakpoints' is selected, only hardware breakpoints are used for execution breakpoints.

Note that the debugger uses execution breakpoints hidden from the user when executing the source step debug command. A single execution breakpoint is sufficient to implement the source step for almost all supported CPU families. NEC 78K microcontroller is an exception to this rule. In worst case, the debugger can use all 16 available hardware breakpoints in order to carry out the source step and thus leaving non to the user. In contrary, when all available hardware breakpoints are used as execution breakpoints, the debugger may fail to execute the source step. Software breakpoints should be used if the user finds this too annoying.

*Software Breakpoints*

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation.

When a software breakpoint is being used, the program first attempts to modify the source code by placing a break instruction into the code. If setting software breakpoint fails, a hardware breakpoint is used instead.

Note: It is recommended to use unlimited software breakpoints since the 78K emulator provides RAM overlay memory for the complete code area.

## 3.3  Advanced Options
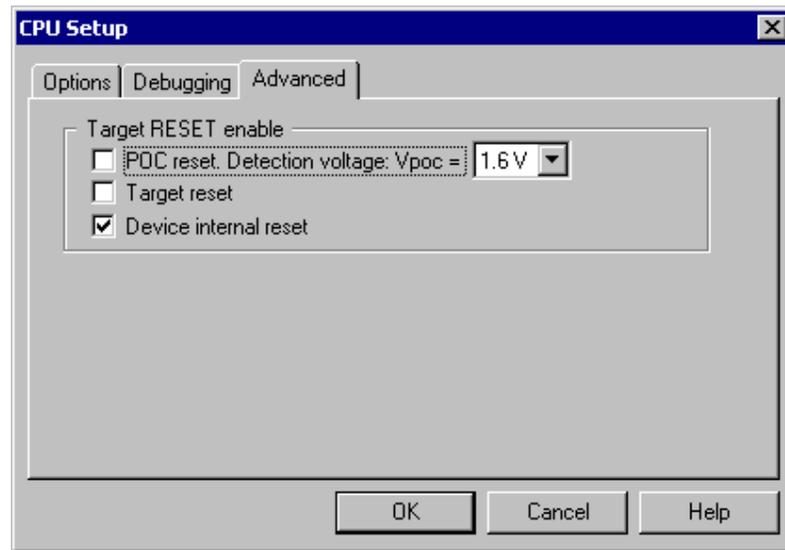


*NEC 78k Advanced Options*

### Target RESET enable

The NEC 78k emulation system allows different target RESET options which can individually be defined in this dialog. Also, a POC reset comparator is available, for which the voltages can be specified between 1.6V and 2.7V.

# 4   Memory Access

NEC 78k development tool features standard monitor memory access, which require user program to be stopped and real-time memory access based on a shadow memory, which allows reading the memory while the application is running.

## Real-Time Memory Access

The emulator features dual-port memory for the complete 78K0 memory space except for the SFR area, which is covered by the shadow memory. This allows all memory to be read and write in real-time without intrusion on the application execution, except for the SFRs, which can only be read in real-time.

## Monitor Access

When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

Since all accesses are performed using the CPU, all memory available to the CPU can be accessed. The drawback to this method is that memory cannot be accessed while the CPU is running. Stopping the CPU, accessing memory and running the CPU is an option, which, however, affects the real time execution considerably.

The time the CPU is stopped for is relative and cannot be exactly determined. The software has full control over it. It stops the CPU, updates all required windows and sets the CPU back to running. Therefore the time depends on the communication type used, PC's frequency, CPU's clock, number of updated memory locations (memory window, SFR window, watches, variables window), etc.

# 5 Access Breakpoints

The load/store breakpoints dialog is open by clicking Hardware breakpoints button in the Breakpoints dialog.
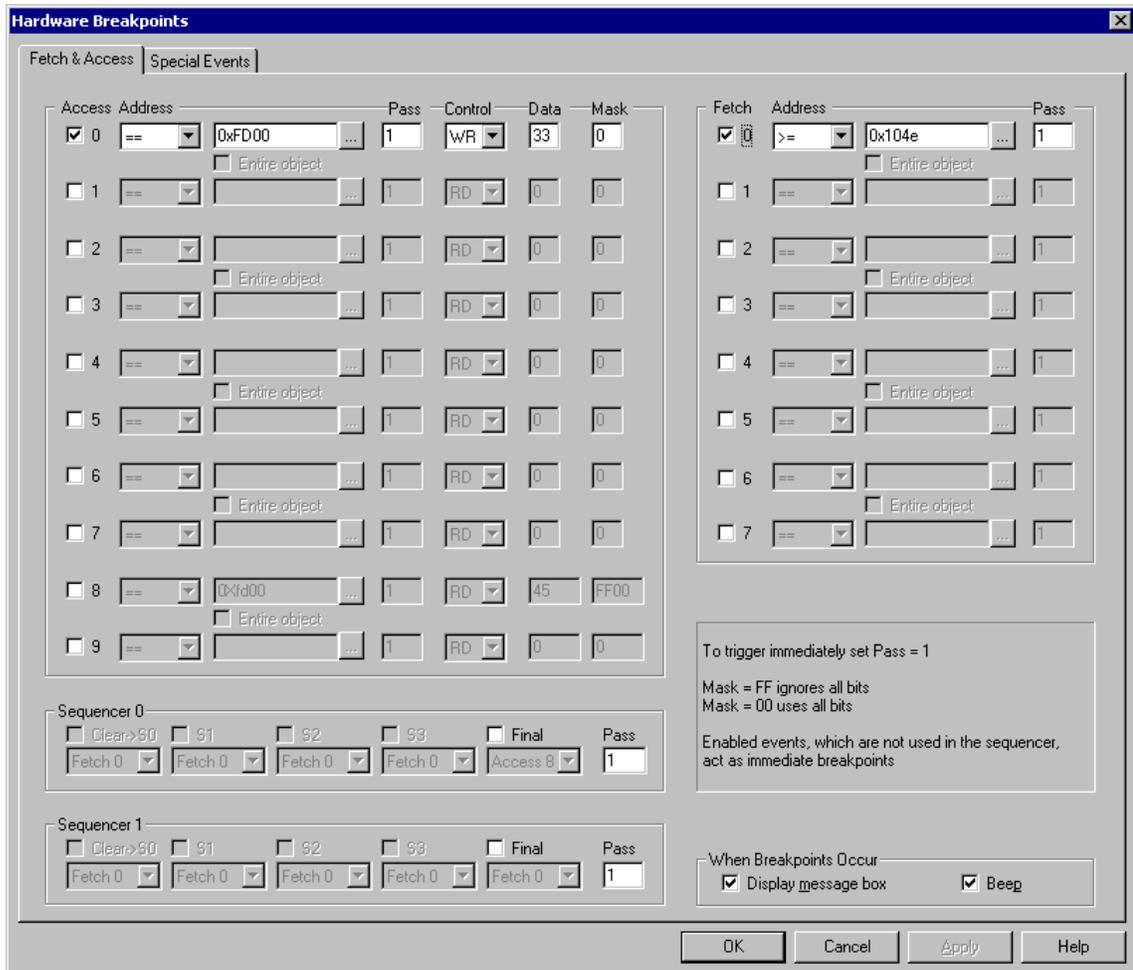


*78k Hardware Breakpoints dialog*

There are a number of access and fetch conditions which can be defined. For each condition the address can be specified (whether it is an exact address, inside or outside a specified range, higher or lower than the specified address or an entire object, specified with the '…' button), a counter of occurrences of a specified event can be defined, the access type (read, write, read/write) and the data and mask of the event.

Also, two sequencers are available to generate an as close as possible breakpoint event as possible.
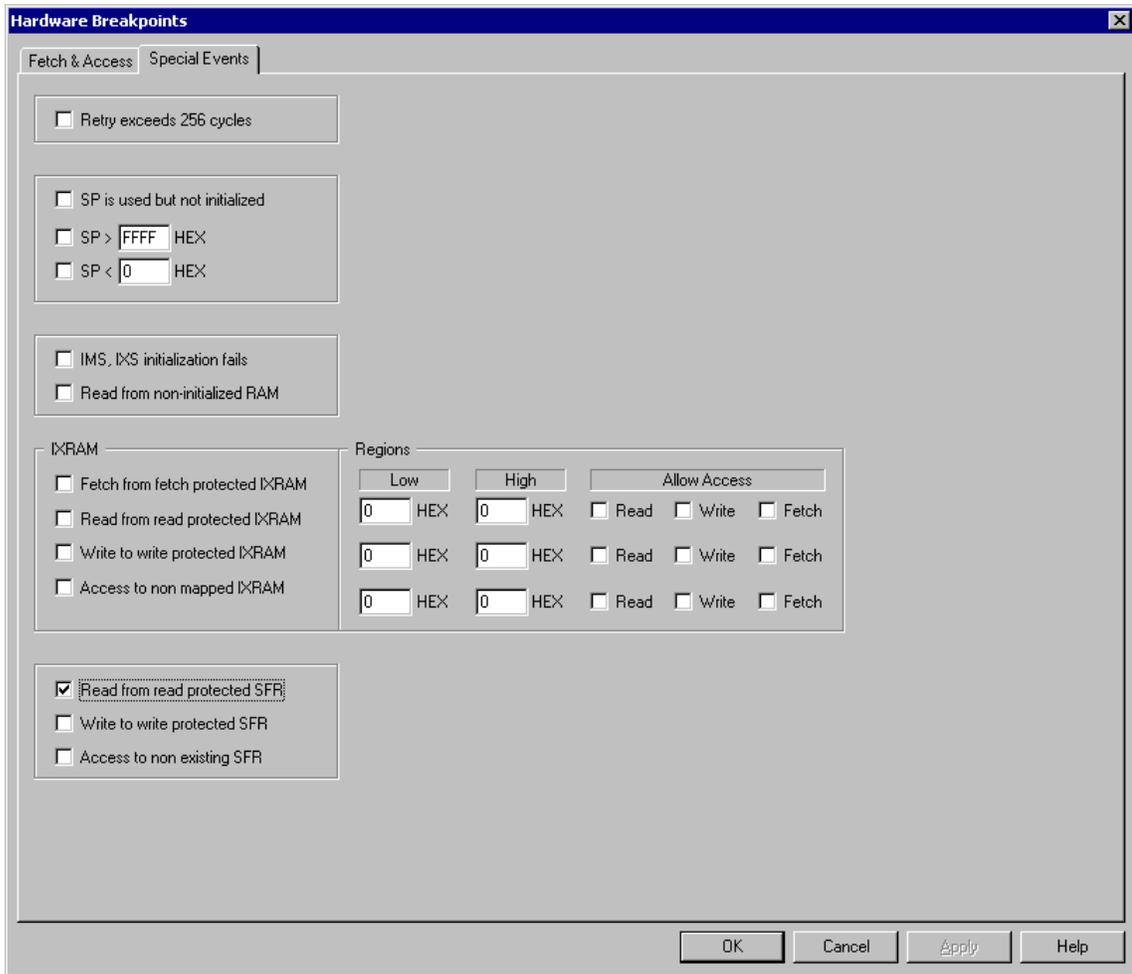
*78k Special Events dialog*

There are a number special events can also be specified as hardware breakpoints. Each special event selected will break the application.

Few useful settings:

- Check the 'SP is used but not initialized' option to break the application, when SP is not initialized before use.

- Specify valid SP range (SP>, SP<) to break the application when the stack use exceeds allocated memory space.

- Check the 'Read from non-initialized RAM' option to break the application when it reads the non-initialized variable.

# 6 Getting Started

1) Connect the system.

2) Power up the emulator and then power up the target.

3) Execute debug reset.

4) The CPU should stop on the address where the reset vector points to.

5) Open memory window at RAM location and check whether it is possible to modify its content.

6) If above steps passed successfully, the debugger is operational.

# 7 Trace

The NEC 78K development system features iSYSTEM proprietary ActivePRO trace.

## Trace Features

- 512K frames buffer depth

- Unlimited time stamp reach

- Supported Trace Configurations:

  ➢ Record Everything

  ➢ Plain Trigger/Qualifier

  ➢ Watchdog Trigger

  ➢ Duration Tracker

  ➢ Q between B and C events

  ➢ Pre/Post Qualifier

  ➢ Data Change

Refer to a separate document describing ActivePRO Trace features and use.
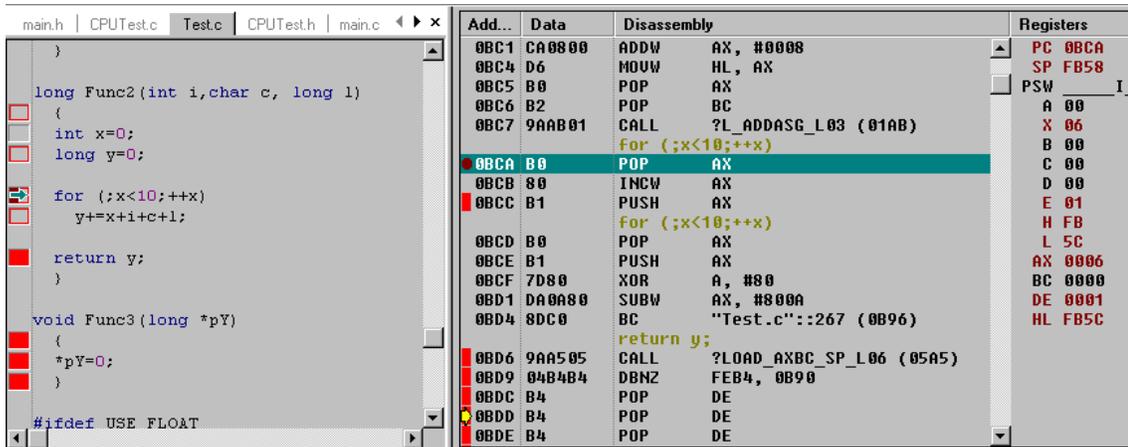
# 8 Execution Coverage

Execution coverage records the addresses from which the code is being executed, which allows the user to detect the code respectively memory areas not being executed. It can be used to detect a so called "dead code", the code that was never executed. The code may not be executed due to the error in the code or it may be a test code which was written temporarily for the debugging needs but later not removed from the final application. The code which is never executed represents an undesired overhead when assigning the necessary code memory resources.

Execution coverage covers complete CPU address space. It can run an infinite time, which means in practice that the application can run for days and then the results can be analyzed.

- Select 'Profiler/Coverage' window from the View menu and configure Execution Coverage settings. Normally, 'All Downloaded Code' option has to be checked only. The debugger extracts all the necessary information like addresses belonging to each C/C++ function from the debug info, which is included in the download file and configures the emulator accordingly. Refer to software user's guide for more details on configuring Execution Coverage and its use.
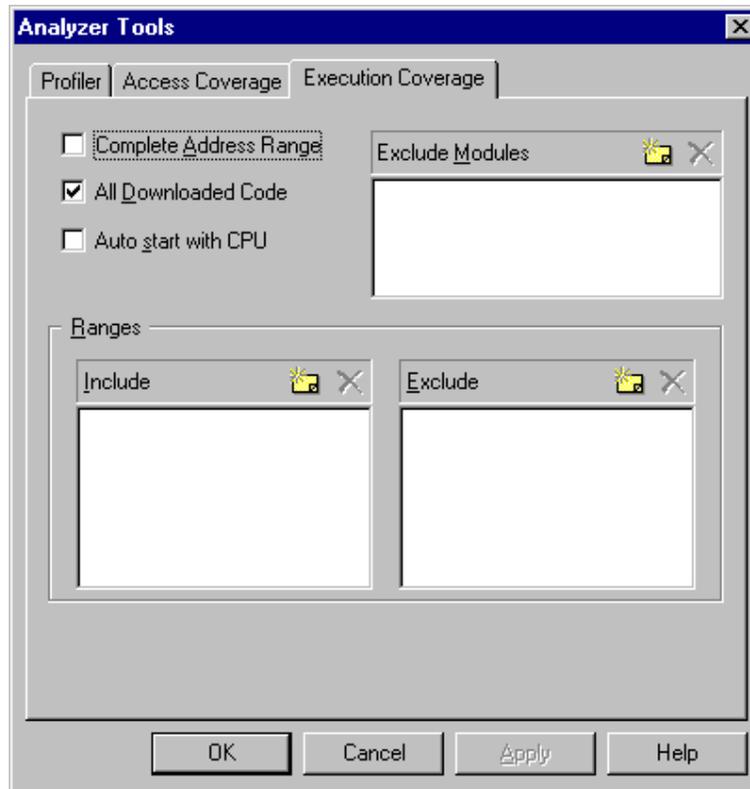
- Execution Coverage is configured. Reset the application, start Execution Coverage and then run the application. When it's assumed that the complete application code is executed, stop the Execution Coverage and inspect the results.

*Execution Coverage results displayed in the Source and the Disassembly window*

Red boxes on the left side in the source and the disassembly window depict the not executed code and Execution Coverage window shows which code was executed/not executed for selected modules.
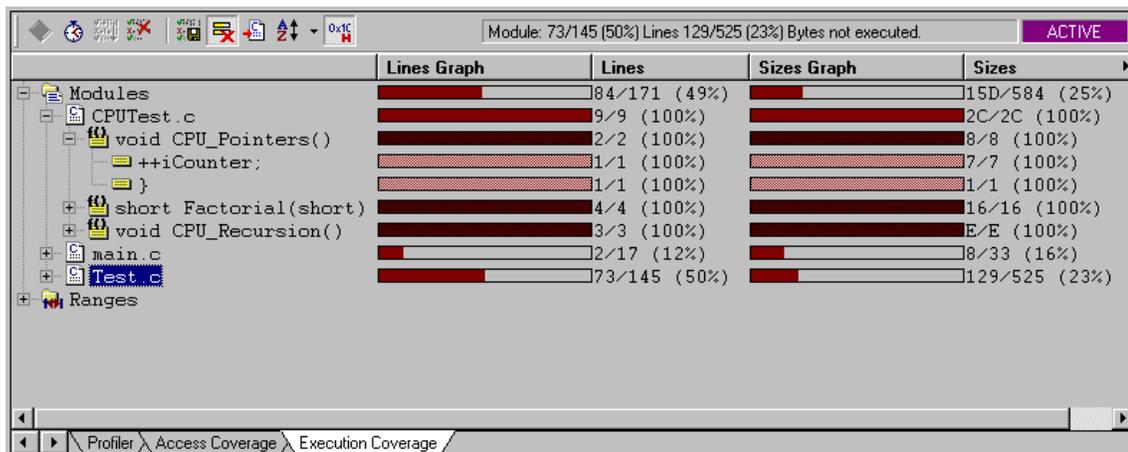


*Execution Coverage window*

# 9  Access Coverage

Access Coverage is a data access analysis technique, which records all CPU addresses being accessed and a type of the access, which can be: code read, data read or data write. It can be used to verify the stack consumption. For instance, when using an operating system, each task has its own stack allocated. After the program is stopped, it can be checked which tasks have used more bytes and which tasks less bytes than it was reserved for each task. From these results the user can re-allocate and re-assign the size of the stack memory for each task which at the end yields optimized memory resources.

Access Coverage covers complete CPU data address space. It can run an infinite time, which means in practice that the application can run for days and then the access coverage results can be analyzed.

- Select 'Profiler/Coverage' window from the View menu and configure Access Coverage settings. Normally, 'Complete Address Range' option, which covers complete data space, has to be checked. Additionally, 'Auto start with CPU' option can be selected. Refer to software user's guide for more details on configuring Access Coverage and its use.
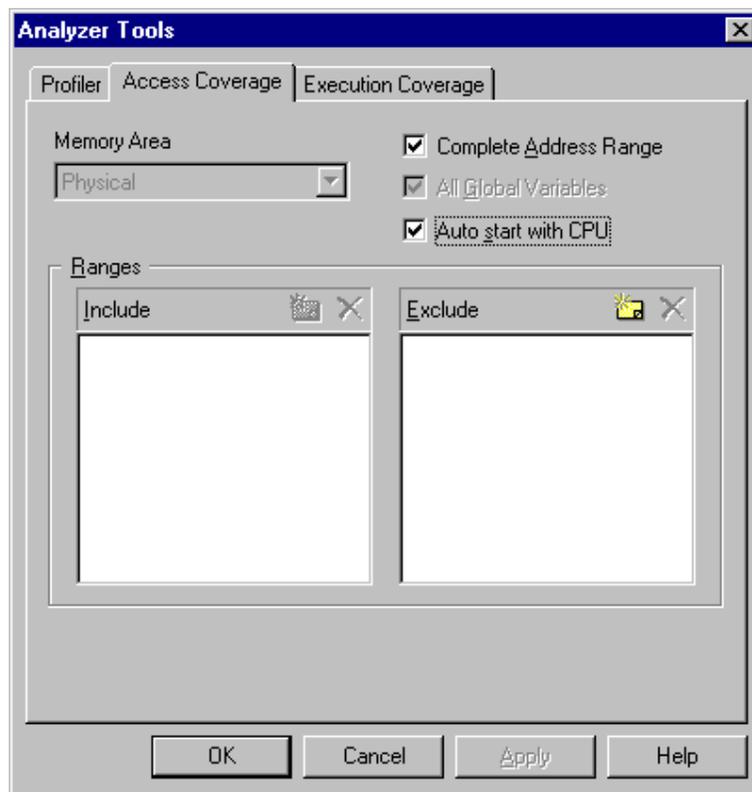


Access Coverage is configured. Reset the application, start Access Coverage and then run the application. When it's assumed that the complete application code was executed, stop the Access Coverage and inspect the results.

'Data Mode' toolbar in the Access Coverage window allows navigating among Code Read, Data Read and Data write access type.
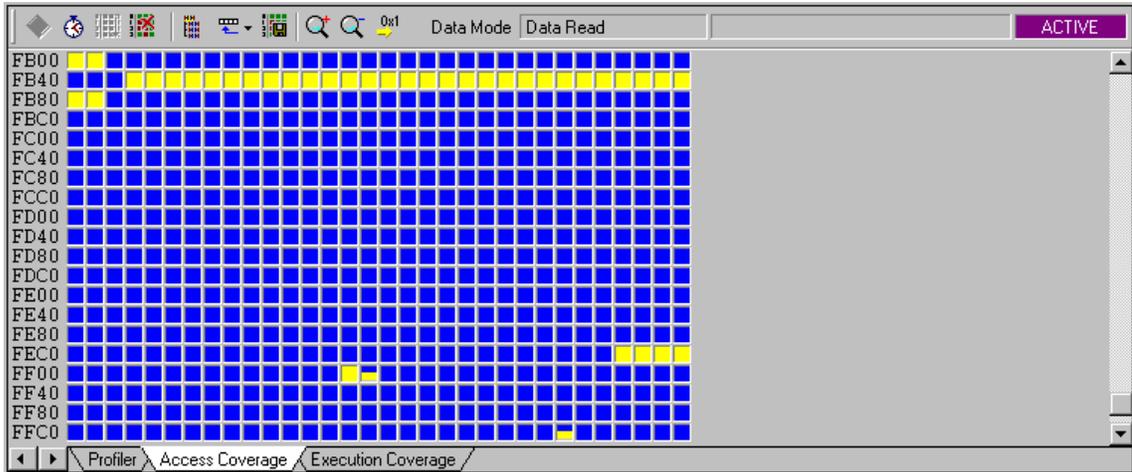
*Access Coverage window*

# 10 Execution Profiler

The Execution Profiler records executed function entry and exit points and then run time-analysis over the collected information. As a result it gives details on how much time (minimum, maximum, average) has the CPU spent in the particular function. Available information allows the user to optimize those parts of code, which are most time consuming or time critical.

The debug download file must contain accurate debug information when using Profiler to analyze C/C++ application. Normally Profiler extracts all the necessary information from the debug information and becomes useless if configured for wrong function entry and exit points.

- Select 'Profiler/Coverage' window from the View menu and configure Profiler settings. Select 'Functions' option in the 'Profile' field.

- Make sure that 'Keep history' option is checked if Code Execution view is going to be used during results analysis.

- Finally, profiled C/C++ functions are selected by pressing 'New…' button. It's recommended that 'All C Functions' is selected for the beginning. Additionally, 'Include lines' can be checked which will yield in time analysis of each source line belonging to the function.

The debugger extracts all the necessary information from the debug info, which is included in the download file and configure the hardware accordingly.

Refer to software user's guide for more details on configuring Profiler and its use.
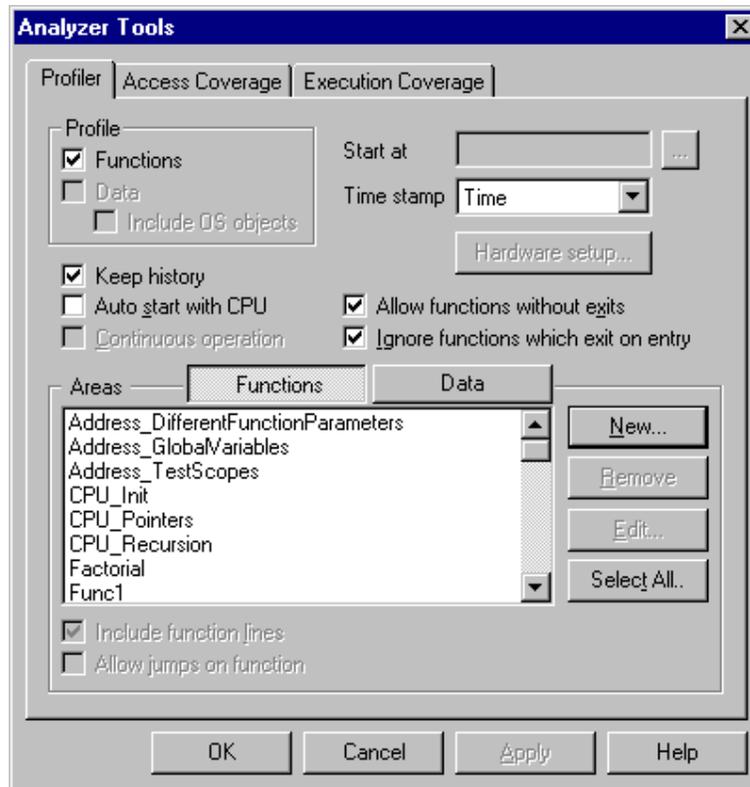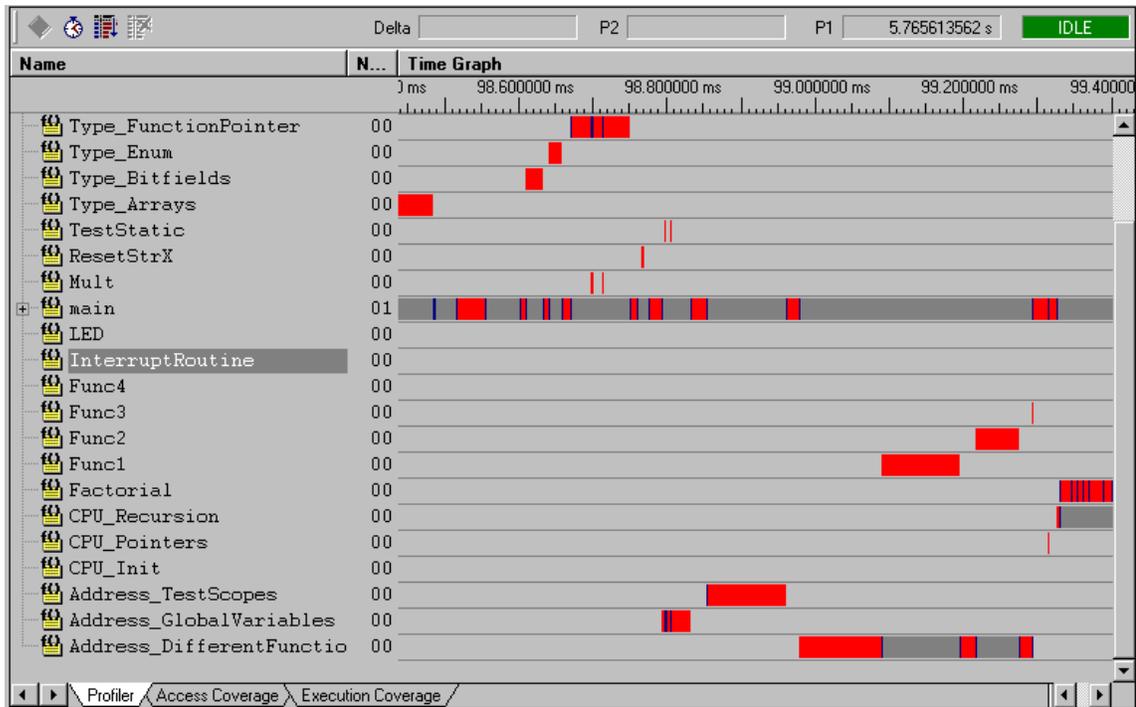
*Profiler configuration settings*

Profiler is configured. Reset the application, start Profiler and then run the application. The Profiler will stoop collecting information on a user demand or after the trace buffer becomes full. Then the recorded information is analyzed and profiler results displayed.



*Profiler results – Code Statistics view*

---

*Profiler results – Code Statistics view*