Freescale 68HCS12 Family In-Circuit Emulation

Contents

1 Introduction

Debug Features

- Unlimited breakpoints
- Access breakpoint
- Real-time access
- Trace
- Execution profiler
- Execution coverage

1.1 Differences from a standard environment

The In-Circuit Emulator emulates the target CPU, which is removed from the target, as good as possible. Beside the CPU, additional logic is integrated on the POD. The amount of additional logic depends on the emulated CPU and the type of emulation. A buffer on a data bus is always used (minimal logic) and when rebuilding ports on the POD, maximum logic is used. As soon as the POD is connected to 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. Consequentially, in worst case signal cross-talks and reflections can occur due to bad target connection, 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 you should follow.

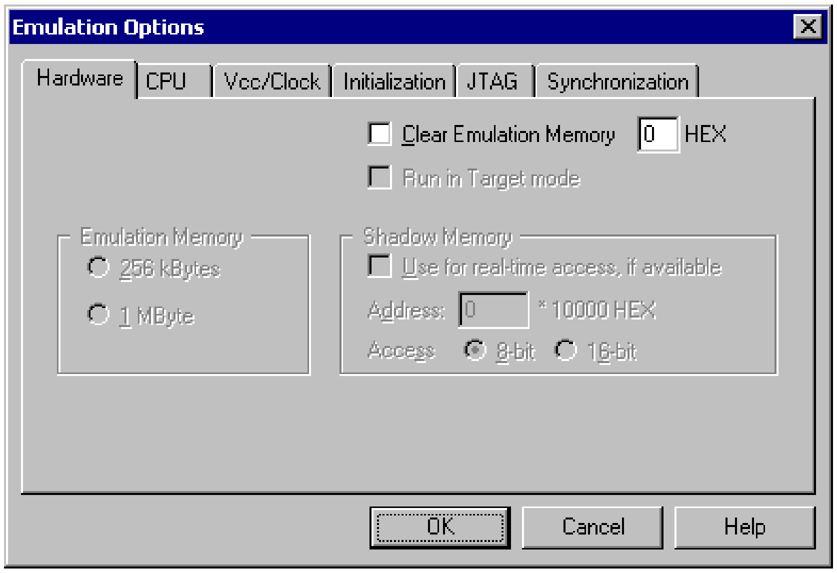
- Use external (target) Vcc/GND if possible (to prevent GND bouncing),
- Make an additional GND connection from POD to the target if the Emulator behaves strangely,
- Use the reset output line on the POD to reset the target whenever Emulator resets the CPU,
- Make sure the appropriate CPU is used on the POD. Please refer to the POD Hardware reference received with your POD.
- No on-chip or external watchdog timers can be used during emulation (unless explicitly permitted). Disable them all.
- When interrupts in background are enabled, take note that the interrupt routine must return in 25 ms, otherwise the Emulator will assume that the program is hung.

1.3 Port Replacement Information

In general, when emulating the single chip mode, some ports have to be rebuilt on the POD because original ports are used for emulation – typically ports used as address and data bus in extended mode. Standard integrated circuits are used to rebuild "lost" ports. Rebuilt ports are logically compatible with original CPU's ports, but electrical characteristics may differ. The differences may become relevant when standard integrated circuits are used and operating close to electrical limits, e.g. when input voltage level is close to specified maximum voltage for low input level ("0") or specified minimum voltage for high input level ("1") or if, for example, the target is built in the way that the maximum port input current must be considered.

2 Emulation Options

2.1 Hardware Options



Clear Emulation Memory: 0 HEX
Run in Target mode

Emulation Memory
- 256 kBytes
- 1 MByte

Shadow Memory
- Use for real-time access, if available
- Address: 0 * 10000 HEX
- Access: [✓ 8-bit] 16-bit

Active Emulator Options dialog, Hardware page

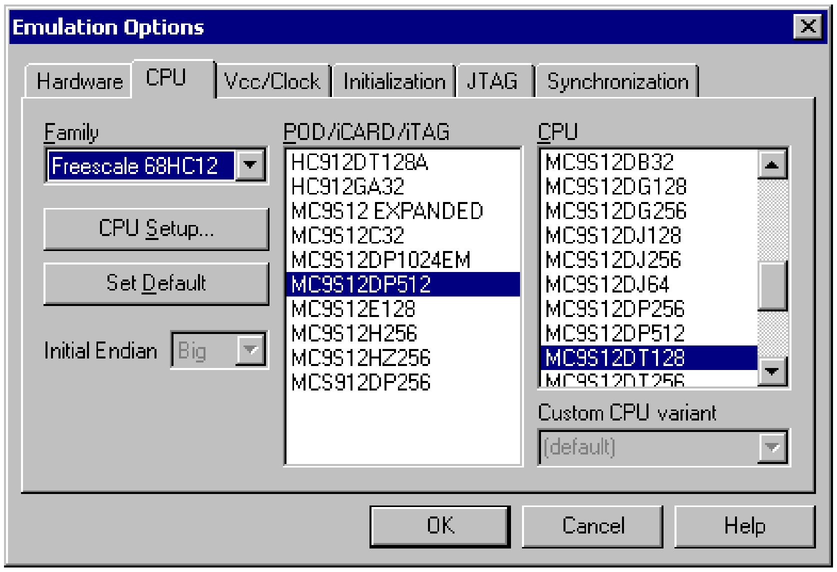
Clear Emulation Memory

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).



Family: Freescale 68HC12

Initial Endian: Big

POD/iCARD/iTAG:
HC912DT128A
HC912GA32
MC9S12 EXPANDED
MC9S12C32
MC9S12DP1024EM
MC9S12DP512
MC9S12E128
MC9S12H256
MC9S12HZ256
MCS912DP256

CPU:
MC9S12DB32
MC9S12DG128
MC9S12DG256
MC9S12DJ128
MC9S12DJ256
MC9S12DJ64
MC9S12DP256
MC9S12DP512
MC9S12DT128
MC9S12DT256

Custom CPU variant: (default)

Active 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

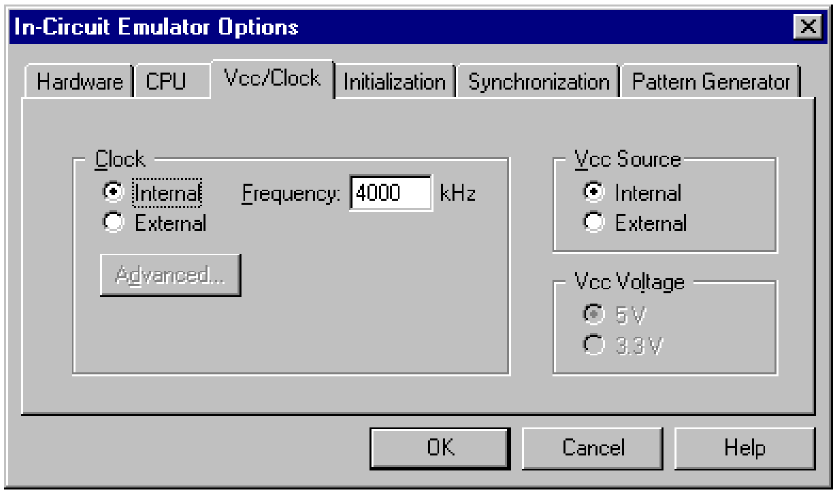
This button will set default options for currently selected CPU. These include:

- Vcc and clock source and frequency
- Advanced CPU specific options
- Memory configuration (debug areas, banks, memory mapping)

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.



Clock: [✓ Internal] External — Frequency: 4000 kHz

Vcc Source: [✓ Internal] External

Vcc Voltage: [✓ 5 V] 3.3 V

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 can happen that the emulator fails to initialize.

It is dissuaded to use a crystal in the target as a an external clock source during the emulation. It is recommended that the oscillator is 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 datasheet. 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 further more 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.

When external clock is selected, its clock frequency must be entered in the frequency field too otherwise the emulation fails. Note that this requirement applies only to MC9S12 EXPANDED Active POD, where a different emulation technique is used than on other HCS12 development systems.

When the clock source is set to Internal, the clock is provided by the emulator and its frequency can be set in steps of 1 kHz.

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

This setting determines whether the emulator or the target system provides a power supply for the CPU.

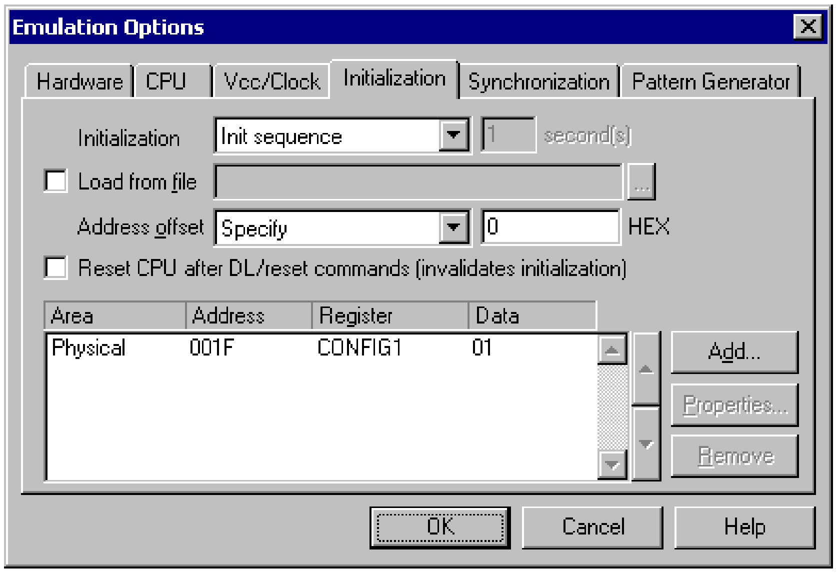
2.4 Initialization Sequence

Usually, there is no need to use initialization sequence when debugging with an In-Circuit Emulator (ICE) a single chip application. Primarily, 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 (CPU or CPU external) memory. With an ICE system, the initialization sequence may be required for instance to enable memory access to the CPU internal EEPROM or to some external target memory, which is not accessible by default after the CPU reset. The user can also disable CPU internal COP using initialization sequence if there is a need for that, etc.

Initialization sequence is executed immediately after the CPU reset and then the code is downloaded.

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.



Initialization: Init sequence; 1 second(s)

Load from file: unchecked

Address offset: Specify; 0 HEX

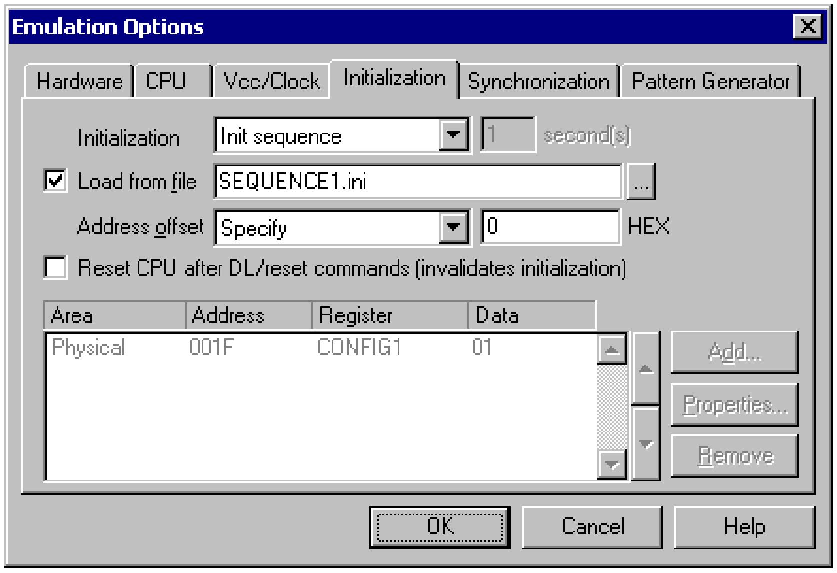
Reset CPU after DL/reset commands (invalidates initialization): unchecked

Area	Address	Register	Data
Physical	001F	CONFIG1	01

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 SEQUENCE1.ini file:

```
S PTBD B 12             //comment
S PTBDD B FF
```


Emulation Options

Hardware | CPU | Vcc/Clock | Initialization | Synchronization | Pattern Generator

Initialization: Init sequence 1 second(s)

[✓] Load from file: SEQUENCE1.ini

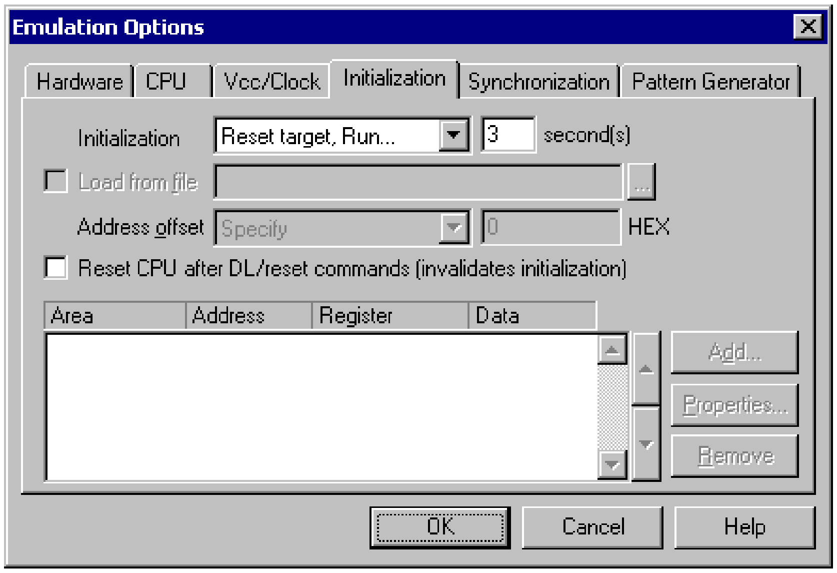
Address offset: Specify 0 HEX

[ ] Reset CPU after DL/reset commands (invalidates initialization)

Area	Address	Register	Data
Physical	001F	CONFIG1	01

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.


Emulation Options

Hardware | CPU | Vcc/Clock | Initialization | Synchronization | Pattern Generator

Initialization: Reset target, Run... 3 second(s)

[ ] Load from file

Address offset: Specify 0 HEX

[ ] Reset CPU after DL/reset commands (invalidates initialization)

Area	Address	Register	Data

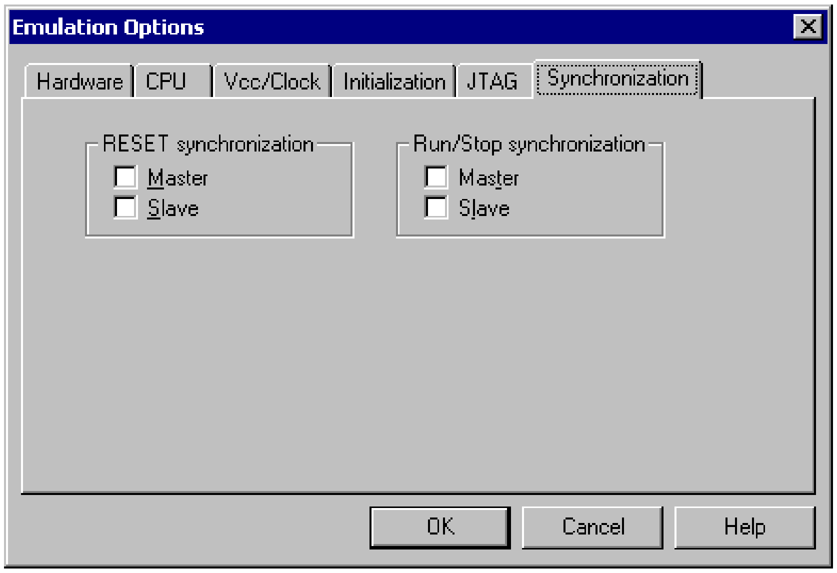
2.5 Synchronisation of Two or More Emulators

S12X development system allows synchronization of two or more emulators.

There are two special pins on the POD:

- SR - synchronous reset
- SS - synchronous stop

The corresponding pins of all Emulators must be connected together. This means that all SR pins must be connected together, and all SS pins must be connected together (but, of course, separated from each other).



Synchronization Options

The Emulator operation is specified in a special dialog, the 'Hardware/In-circuit emulation/Synchronization' tab. Both, SR and SS line can operate independent of each other as master or slave. The user must specify the operating mode in the 'Synchronization' tab.

Synchronous Reset (SR) line

- Master: when the CPU resets, the reset will also be broadcast on the SR line;
- Slave: only monitors the activity on the SR line.

Synchronous Stop (SS) line

- Master: when the program stops, this will be broadcast on the SS line;
- Slave: monitors the activity on the SS line and automatically stops or goes into running depending on the messages received on the line.

Both lines, synchronous reset and synchronous stop are open drain lines – it means that they can operate as input or output. If you define the Emulator as master, open drain line (SR, SS) operates as output and if you define the Emulator as slave it operates as input.

RESET	Emulator	Target
Entry (RUN→RESET)	Immediate	Immediate
Exit (RESET→RUN)	150 E cycles	Immediate

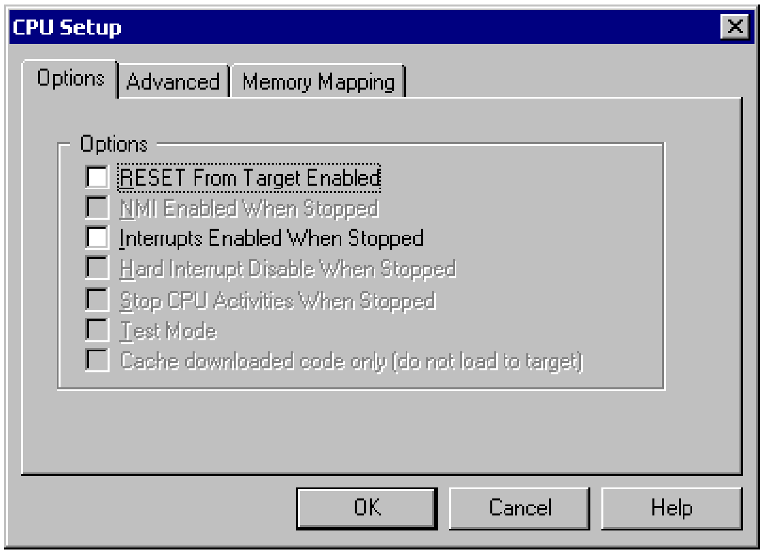
Note: If you are using two emulators with S12X running on the same clock, they will start synchronously. Also the RESET output line on the POD is delayed until the CPU starts. You can use this line to synchronize a third CPU.

Run/Stop	Emulator
Entry (RUN→WAITING)	On next source line
Exit (WAITING→RUN)	0-12 µs

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.



Options
- [ ] RESET From Target Enabled
- [ ] NMI Enabled When Stopped
- [ ] Interrupts Enabled When Stopped
- [ ] Hard Interrupt Disable When Stopped
- [ ] Stop CPU Activities When Stopped
- [ ] Test Mode
- [ ] Cache downloaded code only (do not load to target)

CPU Setup, Options page

RESET from Target Enabled

When checked, the target reset line is sensed, which can then reset the CPU while the CPU is running.

Interrupts Enabled When Stopped

Source step debug command is considered as a stop mode from the debugger's standpoint although it is implemented by setting breakpoints (hidden to the user) on adequate addresses and running the program up to them. This particular option refers to the stop mode, but it really impacts source step behaviour only. When the program is stopped, the CPU enters in so-called BDM mode in which the CPU cannot service any interrupt. Any pending interrupts are serviced after the program is resumed.

When this option is checked, I (Interrupt Mask) flag in the CCR register is not modified by the emulator. It means that when the user program is stopped or stepped through the sources, I flag is not affected by the emulator. For instance, if interrupts are enabled and there is an active interrupt request, it will be serviced during source step.

When this option is unchecked, the interrupts are disabled in stop mode. After the user program is stopped, the emulator, hidden from the user, memorizes the current I (Interrupt Mask) flag state and sets it, which result in disabled interrupts. When the user program is resumed, the emulator first restores original I flag and then resumes the program.

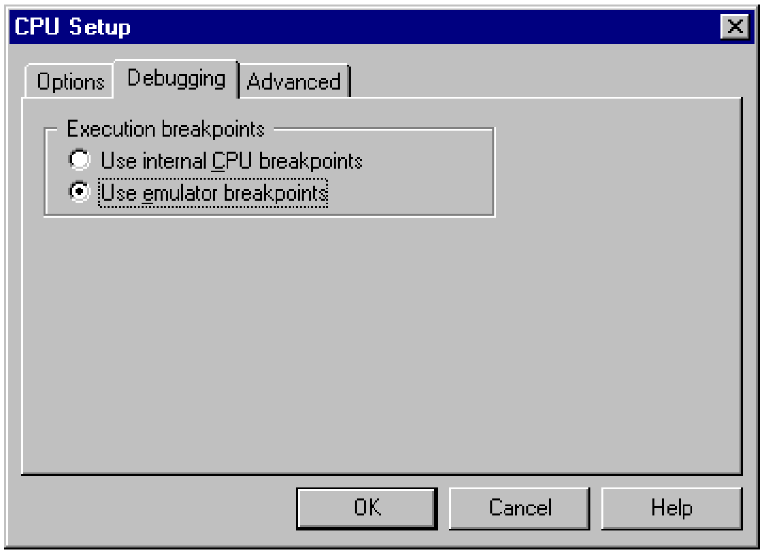
There is no problem when the 'Run' command is being used, but a problem can occur under certain conditions when a single step command is being used.

While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of I flag and handles it correctly. For example, interrupts are active and the program is stopped. The emulator memorizes the I flag state and disables interrupts. Now the user executes single steps in the disassembly window and, for example, once the SWI instruction (software interrupt) is stepped. At this moment, the CPU pushes the content of the CCR register to the stack, where the Interrupt Enable flag is stored and jumps to the address where the interrupt vector points to. Before the user's program was stopped (from running), the interrupts were active (I flag cleared) and after the program was stopped, they were disabled (I flag set) by the emulator. Therefore an incorrect I flag value (CCR register) is now pushed to the stack. Since the emulator can detect such an instruction it modifies the stack with the proper I flag value. If this would not be done, the program execution would be changed after RETI instruction is executed on software interrupt routine exit. Interrupts in the user's program would now be disabled and not enabled as before while the program was running.

But when using step in the source window (source step) the above problem becomes relevant and the user should never forget it. The source step is actually executed with RUN command with prior setting breakpoints on the required source lines. If for instance SWI (software interrupt) occurs during the source step execution, the CCR value with disabled interrupts will be pushed to the stack and after returning from the software interrupt routine (RETI) the same value is popped up from the stack. When the user resumes his program, interrupts are disabled and not enabled, as they were, before the program was stopped.

During the source step the emulator cannot detect instructions that changes the state of I flag as it is the case with single step in the disassembly window.

3.2 Debugging Options



Execution breakpoints

( ) Use internal CPU breakpoints

(●) Use emulator breakpoints

Debugging options

Execution Breakpoints

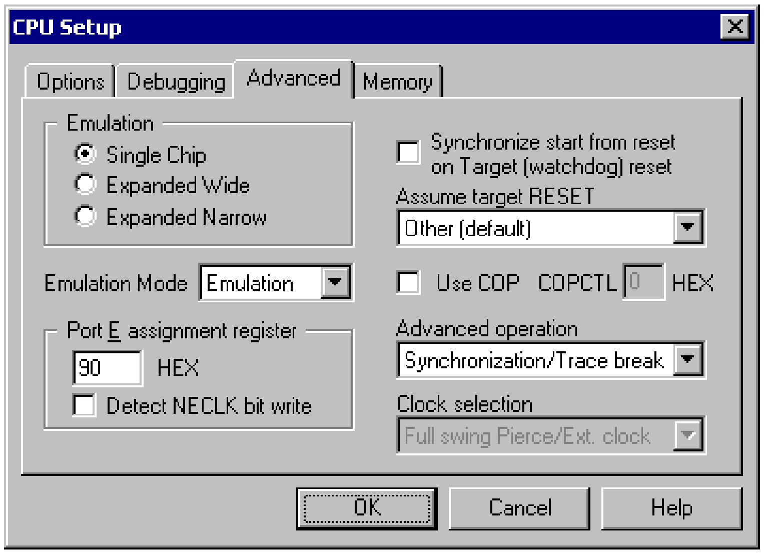
The development system can use two types of breakpoints:

- **'Use internal CPU breakpoints'** specifies the usage of the internal CPU breakpoint logic (also referred to as **hardware breakpoints**), which features two breakpoints except MC9S12Cxxx and MC9S12Exxx derivatives feature 3 hardware breakpoints. The main advantage of this option is that the code is not being modified; the program memory is only used for program storage.

Note that the debugger, when executing source step debug command, uses one breakpoint. Hence, when all available hardware breakpoints are used as execution breakpoints, the debugger may fail to execute debug step. The debugger offers 'Reserve one breakpoint for high-level debugging' option in the Debug/Debug Options/Debugging' tab to circumvent this. By default this option is checked and the user can uncheck it anytime.

- **'Use emulator breakpoints'** specifies the usage of emulator breakpoints (also referred to as **software breakpoints**). This is done by code modification, where break instructions are inserted, where a breakpoint is required. The number of breakpoints is unlimited and unnoticeable to the user, unless the code integrity is being checked.

3.3 Advanced Options



Emulation
- [✓ Single Chip]
- Expanded Wide
- Expanded Narrow

Emulation Mode: Emulation

Port E assignment register: 90 HEX

☐ Detect NECLK bit write

☐ Synchronize start from reset on Target (watchdog) reset

Assume target RESET: Other (default)

☐ Use COP COPCTL 0 HEX

Advanced operation: Synchronization/Trace break

Clock selection: Full swing Pierce/Ext. clock

Advanced Options

Emulation

The setting of this item specifies the emulation mode of the POD. The Expanded modes are only available for the MC9S12 EXPANDED ActivePOD.

Port E Assignment Register (PEAR)

Specify the value that you use in your program. Any configuration of this register in your program is ignored.

Note: Always specify the correct value of this register, even if you do not configure it in your program, in this case specify the default value.

The Emulator requires ECLK, RW and DBE signals for operation and enables them all.

This setting defines the signals that are required for proper target operation.

Note: Reading the PEAR register in the program can return a value different than the one specified here.

Detect NECLK Bit Write

NECLK bit selects port PE4 operation, which can operate either as IO port or ECLK output. When the CPU is running in Emulation expanded wide mode and the 'Detect NECLK Bit Write' option is checked, the emulator detects writes to the PEAR register and runtime selects PE4 operation. Note that this option cannot be used when Normal expanded wide mode is used for emulation.

Note: this option is not available for the MC9S12 EXPANDED ActivePOD.

Emulation Mode

The emulation mode used can be specified. Two emulation modes are available and both have a limitation:

Emulation Expanded Wide Mode

Because of a CPU flaw, in this mode the write to the IRQE bit in the INTCR register is impossible.

Normal expanded Wide mode

The PEAR register can be written any time and thus writable by the user's application. The application misbehaves when ECLK is turned off by the application since the emulator requires ECLK for its operation. The ECLK signal that goes from the emulator to the target is rebuilt on the POD and set according to the settings in the 'CPU Setup/Advanced' dialog (PEAR value). The POD emulates single chip mode (to the target) according to the CPU specification despite that the CPU on the POD operates in the expanded mode. When emulation mode is used, the register is write once register and first write is executed by the emulator, so the application cannot modify it later any more. The user needs to enter the PEAR value (that his application uses) in the 'CPU Setup' dialog. It is recommended to use the Emulation expanded wide mode always, except when the IRQE bit in the INTCR register is used.

Note: this option is not available for the MC9S12 EXPANDED ActivePOD.

Synchronize start from reset on

The target reset signal resets the CPU immediately. However, when the target reset line becomes inactive, the CPU reset line is belayed for few hundred milliseconds by the emulator.

If there is an active external watchdog, the CPU restart must be synchronized with the external watchdog. Then this option and 'RESET From Target Enabled' option must be checked. The watchdog timer event allows reset synchronization on the rising edge of external watchdog (target) reset. Note that the external watchdog must be a periodic signal (while forcing the CPU to a reset state). After the CPU starts, the external watchdog must be refreshed by the application, which ensures the target reset line not to be active.

Note: this option is not available for the MC9S12 EXPANDED ActivePOD.

Use COP

The CPU has an internal watchdog, that must be refreshed periodically, or the CPU resets. The COP can be disabled in STOP (while the CPU is stopped by BDM), which is necessary for debugging. Since the register, which turns off COP is a write-once register, the whole register must be written.

Use COP is a global option with, which the COP usage is selected. If this option is enabled, the option to insert COPCTL is available. If COP is enabled, the software presets the COPCTL register (address 0x003C) with the value, entered into the COPCTL field. It makes sure automatically that RSBCK (the bit that disables COP when the user's program stopped respectively when the CPU is in the BDM mode) is always active.

Assume target RESET

When the CPU is reset by an internal CPU reset source (e.g. COP), the CPU reset line is driven actively. This reset can be detected by the emulator which extends this reset to be able to initialize and start the emulation correctly. At that point, original reset source is no longer known to the emulator and there is no way for the emulator to figure it out. However, this information is required for the emulator to fetch the program counter from a proper vector. 'Assume target RESET' option allows the user to specify, which reset vector is fetched after the CPU reset. The emulator can assume that target reset is generated because of:

- external reset, which yields PC fetch from 0xFFFE
- Clock monitor failure, which yields PC fetch from 0xFFFC)
- COP Failure, which yields PC fetch from 0xFFFA)

The emulator allows testing of single internal reset source at the same time.

When the reset is issued by the emulator (debug reset), it will always fetch the PC from 0xFFFE.

Advanced operation

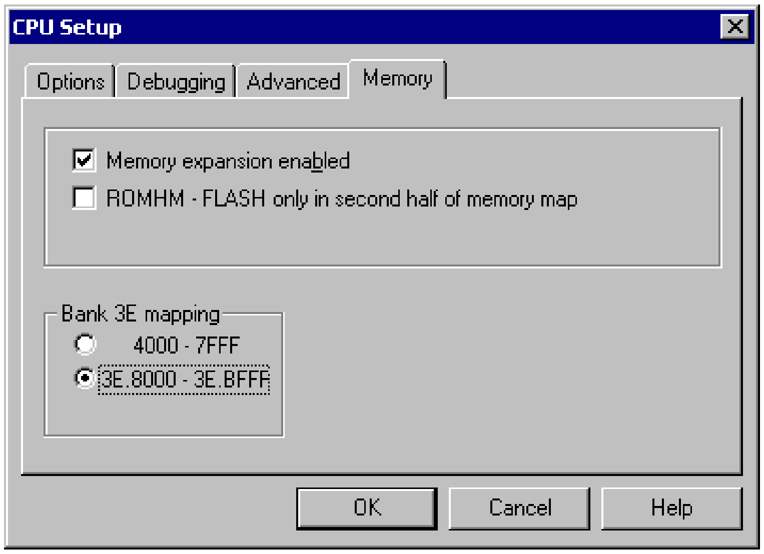
If software breakpoints are used, the program memory can be used in two ways:

- **Task debugging**. This operation mode enables task breakpoints. These are breakpoints, available only in a certain task, which can be set in the 'Debug/Operating System' dialog.
- **Synchronization/Trace break**. Normally, there is a delay when the stop command is used. This mode minimizes the time needed to stop the CPU and is used for quickly stopping the CPU.

Note: this setting is not available for the MC9S12 EXPANDED ActivePOD.

3.4 Memory Expansion

This page configures the mode of the on-chip memory expansion unit. It is shown only for CPUs with integrated memory expansion unit.



[✓] Memory expansion enabled
[ ] ROMHM - FLASH only in second half of memory map

Bank 3E mapping
( ) 4000 - 7FFF
(●) 3E.8000 - 3E.BFFF

Memory Expansion

Memory Expansion Enabled

If Memory Expansion is being used, this option must be checked. According to this selection, the mapping on the POD is automatically configured.

Bank 3E Mapping

HCS12 0x3E bank (bank address: 0x3E8000-0x3EBFFF) is a mirror picture of logic area 0x4000-0x7FFF. The same memory area can be accessed either as bank 0x3E or as logical 0x4000. The trace does not know automatically how to transform the adequate recorded address: into logical 0x4000 or into bank 0x3E8000. With this option proper source level display in the trace window is ensured.

ROMHM

This option sets the ROMHM bit in the MISC register. With this bit turned on, access to the Flash Memory in the area of 0x4000-0x7FFF is turned off.

ROMHM=0 - The 16K byte of fixed FLASH EEPROM in location 0x4000-0x7FFF can be accessed

ROMHM=1 - disables direct access to 16K byte FLASH EEPROM from 0x4000-0x7fff in the memory map. The physical location of this 16K byte FLASH can still be accessed through the Program Page window.

The option must be checked, if direct access to FLASH 0x4000-0x7FFF is disabled. The ROMHM must have the same value as the user uses in his application.

4 Debugging Interrupt Routines

An interrupt routine can only be debugged when the interrupt source for this routine has been disabled; otherwise you will keep reentering the routine and thus run out of system stack.

For example, there is an interrupt routine with 10 source lines. Let's assume that the interrupt routine is called periodically by a free running timer. A breakpoint is set on the first source line in the interrupt routine. Program execution stops at the breakpoint. Now source step is executed. Source step is actually executed using RUN command with prior setting of breakpoint on adequate source line. In this particular case, while source step is executed, the CPU executes the code and before the source step actually completes, a new interrupt call is generated by the timer. Consequentially new values are pushed onto the stack and the CPU stops on breakpoint again. If you repeat source steps in such interrupt routine new values are pushed to the stack and you can easily run out of stack.

An interrupt source can be disabled in two ways:

- Disable the interrupt process when the program is stopped (stop mode) – refer to chapter 3.1 describing 'Interrupts Enabled When Stopped' option. Stop mode is entered whenever the CPU is stopped and the emulator remains in stop mode until the Run command is executed. (During Step, Step over, etc. commands, stop mode persists).
- Do not place a breakpoint on any instruction in the interrupt routine where interrupts are not yet disabled. You should also not step over any instruction that re-enables the interrupt, but run the program only up to that point.

5 Memory Access

HCS12 development system feature standard monitor memory access, which require user program to be stopped and real-time memory access based on BDM interface, which allows reading and writing the memory while the application is running.

Real-Time Memory Access

All HCS12 development systems feature real-time memory access. The debugger can access CPU memory without disturbing the program being executed. Using hardware commands and on-chip BDM firmware, the debugger can access required resources in real time. In general, on-chip BDM firmware uses CPU dead cycles, when real-time read or write memory access is required. In worst case, some CPU cycles might be stolen.

Note: Real-time access is available only for memory, which can be accessed through single memory read/write access.

Watch window, memory window and SFRs window can be updated in real time.

Monitor Access

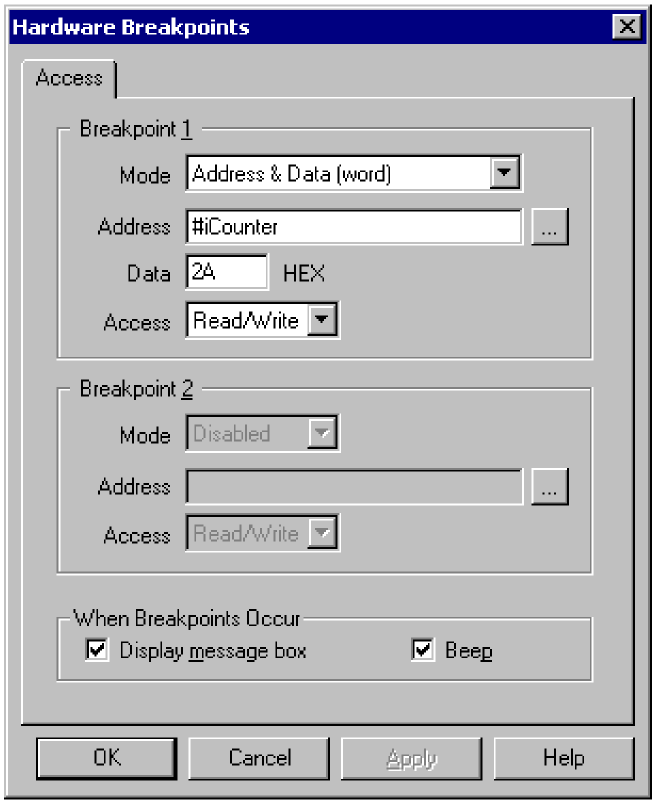
When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

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.

6 Access Breakpoints

The access breakpoints dialog is open by clicking Hardware breakpoints button in the Breakpoints dialog.



Breakpoint 1
Mode: Address & Data (word)
Address: #iCounter
Data: 2A HEX
Access: Read/Write

Breakpoint 2
Mode: Disabled
Address:
Access: Read/Write

When Breakpoints Occur
☑ Display message box ☑ Beep

HCS12 Hardware Breakpoints dialog

Two breakpoints are available and can be combined as two address breakpoints or one breakpoint, which breaks on the specified address and data value. Note that if Address and Data breakpoint is specified for Breakpoint 1, the Breakpoint 2 can not be used due to on-chip breakpoint logic limitations.

The mode is selected in the 'Mode' listbox. The different Address and Data options specify the access width, which can be Byte, Word, Word/High Byte or Word/Low Byte.

Address field accepts directly a number or a name of a selected variable. By pressing on the […] button a variable can be selected from the Source browser.

In the Data field, a data value can be entered.

The access type can also be specified, which can be Read, Write or Read/Write access.

7 Emulation Notes

7.1 Clock Setup

Even when an external clock source is used, you must specify its frequency to allow BDM synchronization (used by Emulator) after the CPU is released from reset.

7.2 PLL

The on-chip PLL is supported by the development system.

7.3 COP

The on-chip COP is supported by the development system. When experiencing problems with the emulator when using COP, check, whether you have a correct/valid interrupt vector table. If for instance COP interrupt vector is defined by mistake as a watchdog interrupt vector, it may look like the COP resets the CPU when the interrupt occurs.

7.4 STOP Instruction

STOP instruction is completely supported by the emulator. After the STOP instruction is being executed, the CPU is stopped and the debugger displays HALTED status. Note that the debug windows cannot be updated while HALTED status is displayed. When the CPU is awaken either by interrupt or target reset, the emulation/execution proceeds normally.

7.5 'Clear on read' register bits

Be careful, when the CPU has special function register bits that are cleared on read access. Do note that when such register (memory location) is accessed either by memory/watch window or SFR window, the flags are cleared and the application may behave different when using the emulator or the target CPU. It is recommended not to display such register or the associated memory location in the memory/watch window during final test. Otherwise, it may happen that target application doesn't work due to the bug in the code even though it works with the emulator.

For instance, the user makes a mistake and does not clear a flag in the application. Using the emulator, the application works correctly since the user uses a SFR window, which clears the flag when the window is updated.

7.6 Internal CPU FLASH

Note that internal FLASH is disabled during the emulation and cannot be used in any way. It must also not be enabled in the user program in any situation, since this would disable the emulation. Thereby, make sure that ROMON bit in the MISC register is never set to 1.

7.7 Internal EEPROM or RAM

Download to the internal EEPROM or the internal RAM must be done through the ‘Target download’.

Additionally, when downloading the code to the internal EEPROM, the user is required to write the ECLKDIV register to divide the oscillator clock down to within the 150kHz to 200kHz range. Refer to the CPU datasheet for more details. ECLKDIV register can be configured prior to download using initialization sequence (chapter 2.4).

7.8 Miscellaneous

Remove all emulator breakpoints when performing any kind of checksum since they may impact the checksum result.

8 Trace

HCS12 development system features a powerful trace named Active Trace, which is implemented externally to the CPU.

Active Trace ‘Break on trigger’ functionality is available only when the development system uses emulator breakpoints for execution breakpoints.

Refer to a separate document describing Active Trace features and use.

9 Execution Coverage

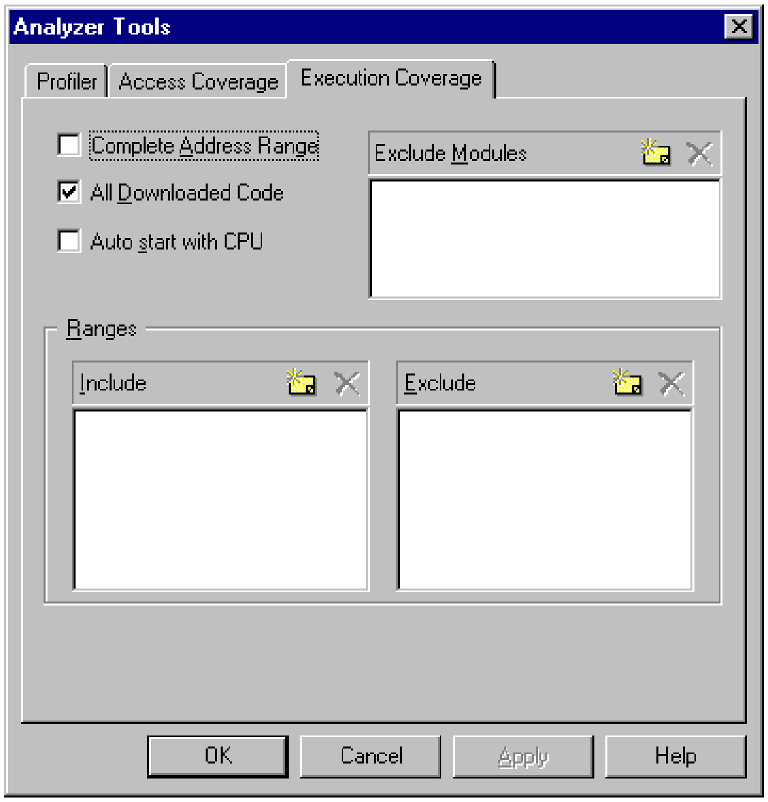
Execution coverage records all addresses being executed, which allows the user to detect the code or memory areas not executed. It can be used to detect a so called "dead code", the code that was never executed. Such code represents undesired overhead when assigning code memory resources.

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

- First, make sure that 'Active' Trace is selected in the Hardware/Analyzer Setup dialog.

- Next, select 'Execution 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 configure hardware accordingly.

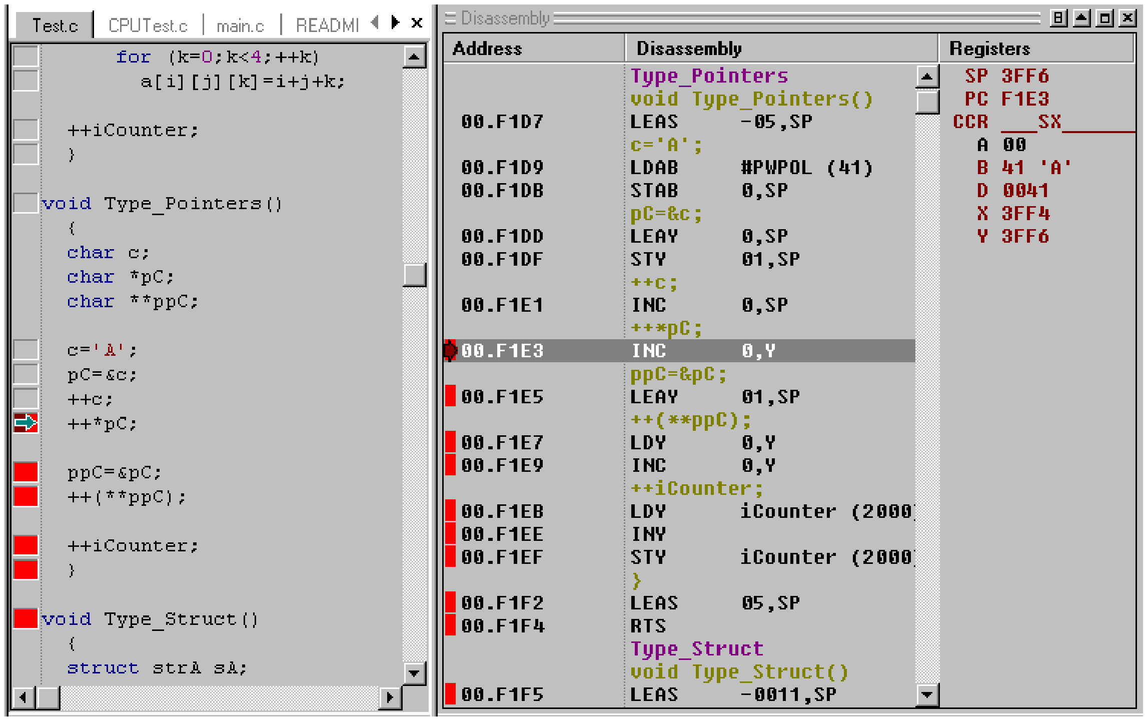
Refer to software user's guide for more details on configuring Execution Coverage and its use.



[ ] Complete Address Range
[✓] All Downloaded Code
[ ] Auto start with CPU

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

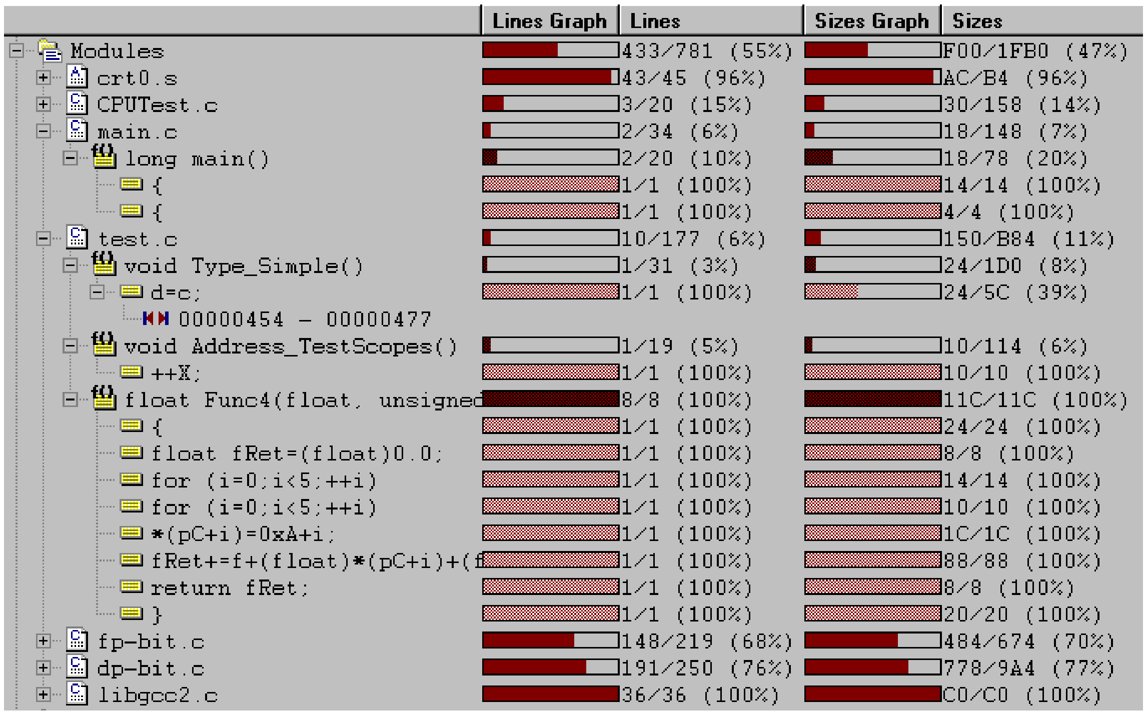
Red boxes on the left in the source window and disassembly window depict not executed code. Execution Coverage window shows which code was executed/not executed for selected modules.



```
Test.c | CPUTest.c | main.c | READMI

        for (k=0;k<4;++k)
          a[i][j][k]=i+j+k;

    ++iCounter;
    }

void Type_Pointers()
    {
    char c;
    char *pC;
    char **ppC;

    c='A';
    pC=&c;
    ++c;
    ++*pC;

    ppC=&pC;
    ++(**ppC);

    ++iCounter;
    }

void Type_Struct()
    {
    struct strA sA;
```

```
Disassembly
Address     Disassembly                      Registers
            Type_Pointers                     SP  3FF6
            void Type_Pointers()              PC  F1E3
00.F1D7     LEAS      -05,SP                 CCR ___SX___
            c='A';                              A  00
00.F1D9     LDAB      #PWPOL (41)               B  41 'A'
00.F1DB     STAB      0,SP                      D  0041
            pC=&c;                              X  3FF4
00.F1DD     LEAY      0,SP                      Y  3FF6
00.F1DF     STY       01,SP
            ++c;
00.F1E1     INC       0,SP
            ++*pC;
00.F1E3     INC       0,Y
            ppC=&pC;
00.F1E5     LEAY      01,SP
            ++(**ppC);
00.F1E7     LDY       0,Y
00.F1E9     INC       0,Y
            ++iCounter;
00.F1EB     LDY       iCounter (2000
00.F1EE     INY
00.F1EF     STY       iCounter (2000
            }
00.F1F2     LEAS      05,SP
00.F1F4     RTS
            Type_Struct
            void Type_Struct()
00.F1F5     LEAS      -0011,SP
```

Source & Dissasembly Window: Execution Coverage results



	Lines Graph	Lines	Sizes Graph	Sizes
Modules		433/781 (55%)		F00/1FB0 (47%)
crt0.s		43/45 (96%)		AC/B4 (96%)
CPUTest.c		3/20 (15%)		30/158 (14%)
main.c		2/34 (6%)		18/148 (7%)
long main()		2/20 (10%)		18/78 (20%)
{		1/1 (100%)		14/14 (100%)
{		1/1 (100%)		4/4 (100%)
test.c		10/177 (6%)		150/B84 (11%)
void Type_Simple()		1/31 (3%)		24/1D0 (8%)
d=c;		1/1 (100%)		24/5C (39%)
00000454 - 00000477				
void Address_TestScopes()		1/19 (5%)		10/114 (6%)
++X;		1/1 (100%)		10/10 (100%)
float Func4(float, unsigned		8/8 (100%)		11C/11C (100%)
{		1/1 (100%)		24/24 (100%)
float fRet=(float)0.0;		1/1 (100%)		8/8 (100%)
for (i=0;i<5;++i)		1/1 (100%)		14/14 (100%)
for (i=0;i<5;++i)		1/1 (100%)		10/10 (100%)
*(pC+i)=0xA+i;		1/1 (100%)		1C/1C (100%)
fRet+=f+(float)*(pC+i)+(f		1/1 (100%)		88/88 (100%)
return fRet;		1/1 (100%)		8/8 (100%)
}		1/1 (100%)		20/20 (100%)
fp-bit.c		148/219 (68%)		484/674 (70%)
dp-bit.c		191/250 (76%)		778/9A4 (77%)
libgcc2.c		36/36 (100%)		C0/C0 (100%)

Execution Coverage Window results

10 Execution Profiler

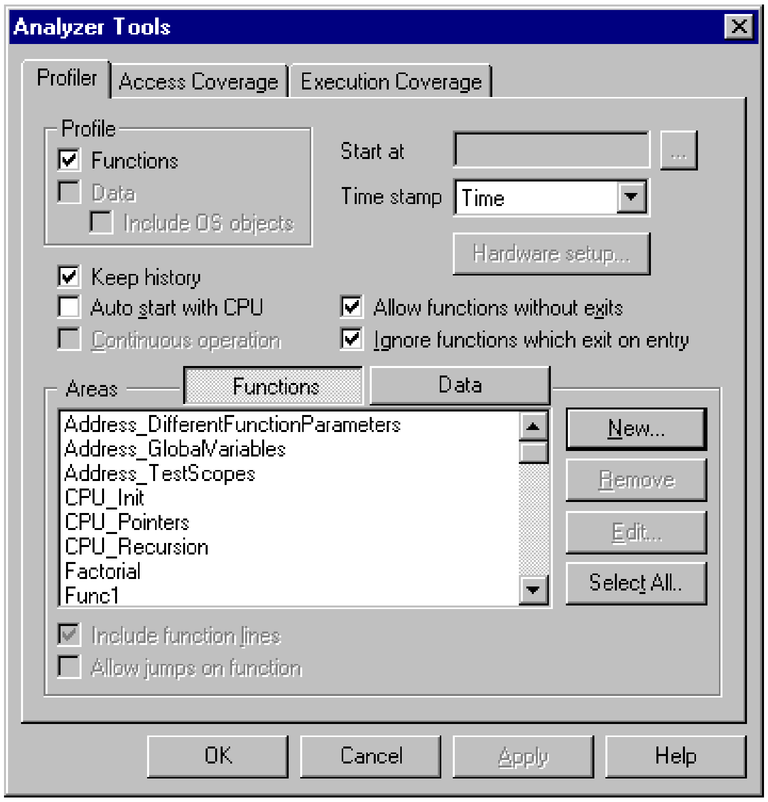
Profiler records executed function entry and exit points and then run time-analysis over the collected information. As a result it gives details on how much time (minimum, maximum, average) has the CPU spent in the particular function. Available information allows the user to optimize those parts of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

- First, make sure that 'Active' Trace is selected in the Hardware/Analyzer Setup dialog.
- Next, select 'Profiler' 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.



Profile: [✓] Functions, [ ] Data, [ ] Include OS objects

Start at:

Time stamp: Time

[✓] Keep history

[ ] Auto start with CPU

[ ] Continuous operation

[✓] Allow functions without exits

[✓] Ignore functions which exit on entry

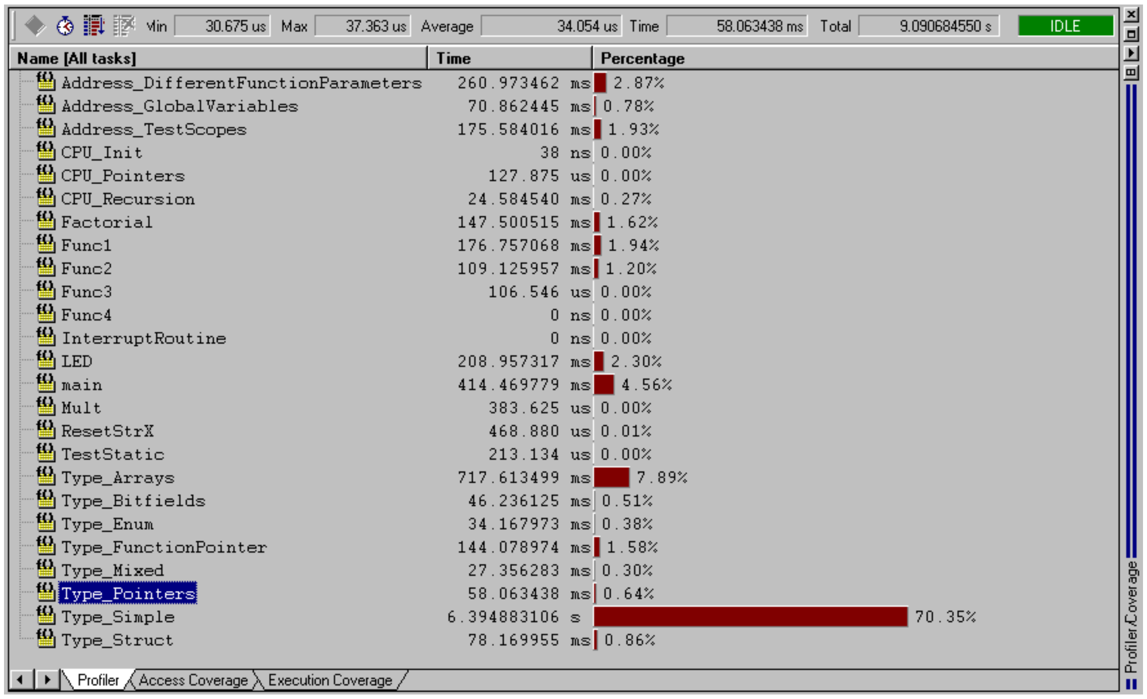
Address_DifferentFunctionParameters
Address_GlobalVariables
Address_TestScopes
CPU_Init
CPU_Pointers
CPU_Recursion
Factorial
Func1

[✓] Include function lines

[ ] Allow jumps on function

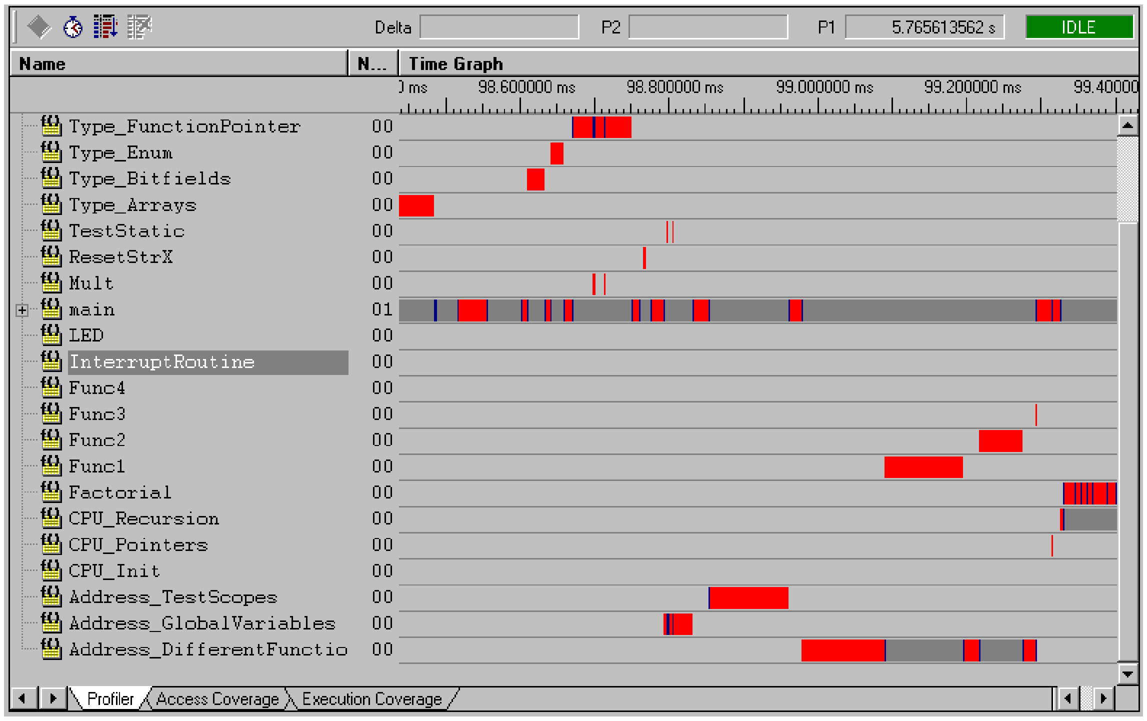
Profiler configuration settings

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.



Name [All tasks]	Time	Percentage
Address_DifferentFunctionParameters	260.973462 ms	2.87%
Address_GlobalVariables	70.862445 ms	0.78%
Address_TestScopes	175.584016 ms	1.93%
CPU_Init	38 ns	0.00%
CPU_Pointers	127.875 us	0.00%
CPU_Recursion	24.584540 ms	0.27%
Factorial	147.500515 ms	1.62%
Func1	176.757068 ms	1.94%
Func2	109.125957 ms	1.20%
Func3	106.546 us	0.00%
Func4	0 ns	0.00%
InterruptRoutine	0 ns	0.00%
LED	208.957317 ms	2.30%
main	414.469779 ms	4.56%
Mult	383.625 us	0.00%
ResetStrX	468.880 us	0.01%
TestStatic	213.134 us	0.00%
Type_Arrays	717.613499 ms	7.89%
Type_Bitfields	46.236125 ms	0.51%
Type_Enum	34.167973 ms	0.38%
Type_FunctionPointer	144.078974 ms	1.58%
Type_Mixed	27.356283 ms	0.30%
Type_Pointers	58.063438 ms	0.64%
Type_Simple	6.394883106 s	70.35%
Type_Struct	78.169955 ms	0.86%

Profiler results – Code Statistics view



Name	N...
Type_FunctionPointer	00
Type_Enum	00
Type_Bitfields	00
Type_Arrays	00
TestStatic	00
ResetStrX	00
Mult	00
main	01
LED	00
InterruptRoutine	00
Func4	00
Func3	00
Func2	00
Func1	00
Factorial	00
CPU_Recursion	00
CPU_Pointers	00
CPU_Init	00
Address_TestScopes	00
Address_GlobalVariables	00
Address_DifferentFunctio	00

Profiler results – Code Statistics view

Notes:

Notes: