
Technical Notes

Freescal ColdFire Family On-Chip Emulation

Contents

Contents.....	1
1 Introduction	2
2 Emulation Options.....	3
2.1 Hardware Options	3
2.2 Initialization Sequence	5
3 CPU Setup	7
3.1 General Options	7
3.2 Debugging options	8
3.3 Advanced Options.....	10
3.3.1 ColdFire V2, V3, V4 and V4e core	10
3.3.2 ColdFire V1 core	11
4 Real-Time Memory Access	12
5 Hot Attach	12
6 Internal FLASH programming	13
6.1 ColdFire V2, V3, V4 and V4e core.....	13
6.2 ColdFire V1.....	14
7 Access Breakpoints and Trace Trigger.....	16
7.1 ColdFire V2, V3, V4 and V4e core.....	16
7.2 ColdFire V1.....	21
8 Trace.....	23
9 Profiler.....	29
10 Execution Coverage.....	33
11 Emulation Notes	35
12 Getting Started.....	37
13 Troubleshooting.....	38

1 Introduction

The ColdFire family implements a low-level system debugger in the microprocessor hardware. Communication with the development system is handled through a dedicated high-speed serial command interface. The ColdFire architecture implements the BDM controller in a dedicated hardware module. Although some BDM operations, such as CPU register accesses, require the CPU to be halted, other BDM commands, such as memory accesses, can be executed while the processor is running.

The external debug hardware uses a three-pin serial, full-duplex BDM communication protocol based on 17-bit data packets for all ColdFire cores except for the ColdFire V1 core. The V1 ColdFire core supports BDM functionality using the HCS08's single-pin interface based on 8-bit data packet.

ColdFire device can feature V1, V2, V3, V4 or V4e core and a debug module Revision A, B, B+, C or D. Refer to the Reference Manual of the CPU for the core type and the debug module Revision.

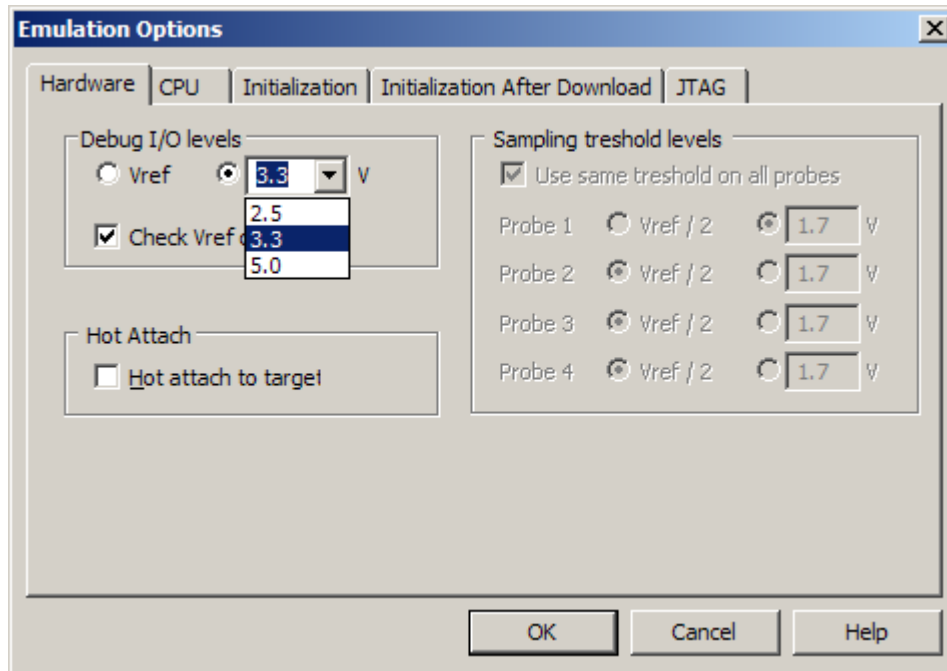
Rev. B debug module has no longer shared debug sources comparing to Rev. A, which restricted real-time access use together with hardware execution breakpoints, access breakpoints or trace trigger at the same time. Rev. B+ adds three more hardware execution breakpoints (4 total). Rev. C extends access breakpoints resources and combines trace port into a single 8-bit port. Rev. D adds MMU support. The version 1 ColdFire core features HCS08 physical interface and rev. B+ of the ColdFire debug architecture.

Debug Features

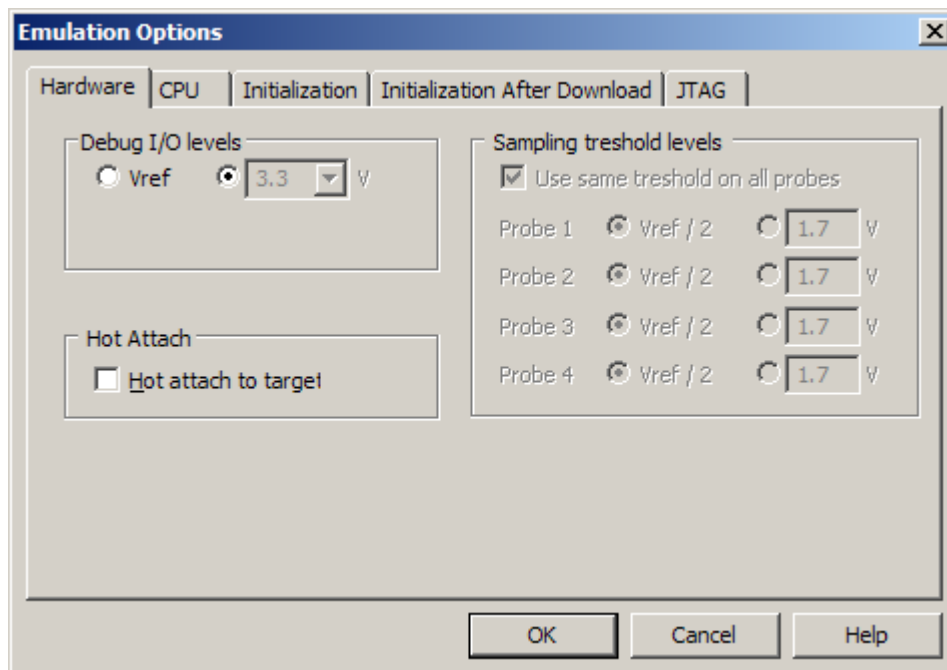
- Limited hardware execution breakpoints (Debug module Rev. A/B = 1, Rev. B+/C/D = 4)
- Unlimited internal FLASH breakpoints
- Unlimited software breakpoints
- Access Breakpoints
- Real-time access
- Hot Attach
- Fast internal/external flash programming
- Trace
- Profiler
- Execution Coverage

2 Emulation Options

2.1 Hardware Options



iC5000 Emulation Options, Hardware page



iC3000 Emulation Options, Hardware page

Debug I/O levels

The development system can be configured in a way that debug BDM signals are driven by the emulator or by the target voltage (Vref).

When 'Vref' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin on the target debug connector is used as a reference voltage for voltage follower, which powers buffers, driving the debug BDM signals. The user must ensure that the target power supply is connected to the Vref pin on the target BDM connector and that it is switched on before the debug session is started. If these two conditions are not met, it is highly probably that the initial debug connection will fail already. However in some cases it may succeed but then the system will behave abnormal.

In case of ic3000 iCARD based development system, emulator firmly drives When a voltage is selected from the combo box

Sampling threshold levels (iTRACE PRO/GT only)

Sampling threshold level can be set for the CPU trace (output) port. Default Vref/2 setting should work unless the trace port signals are shifted. In case of problems with the trace recording, try different voltage levels, e.g. 1.5V, 1.2V, etc.

Hot Attach

Option must be checked when Hot Attach is used. Refer to the [Hot Attach](#) chapter for more details on Hot Attach use.

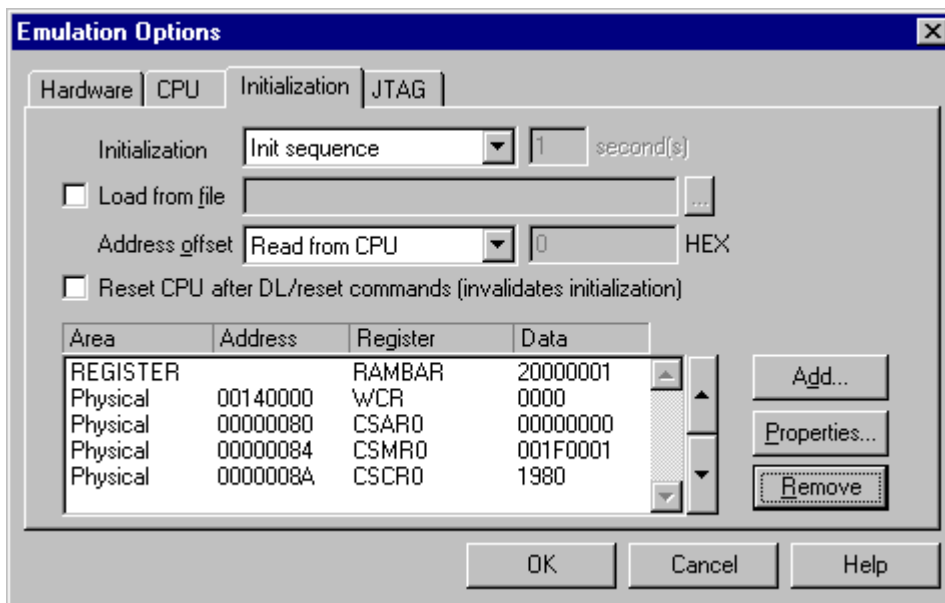
Note: Hot Attach function cannot be used for any flash programming or code download!

2.2 Initialization Sequence

The user must properly configure the CPU before the debug download (including the flash programming) can take place to the memory area, which is not accessible upon the CPU reset. This is essential for the applications using memory resources, for instance external RAM or external flash, which are not accessible after the CPU reset. In such case, the debugger executes a so-called initialization sequence immediately after the CPU reset, which writes to the CPU registers configuring the CPU memory interface to the physical memory and then the debug download is executed. Note that the initialization sequence must be set up specific to the application. 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.



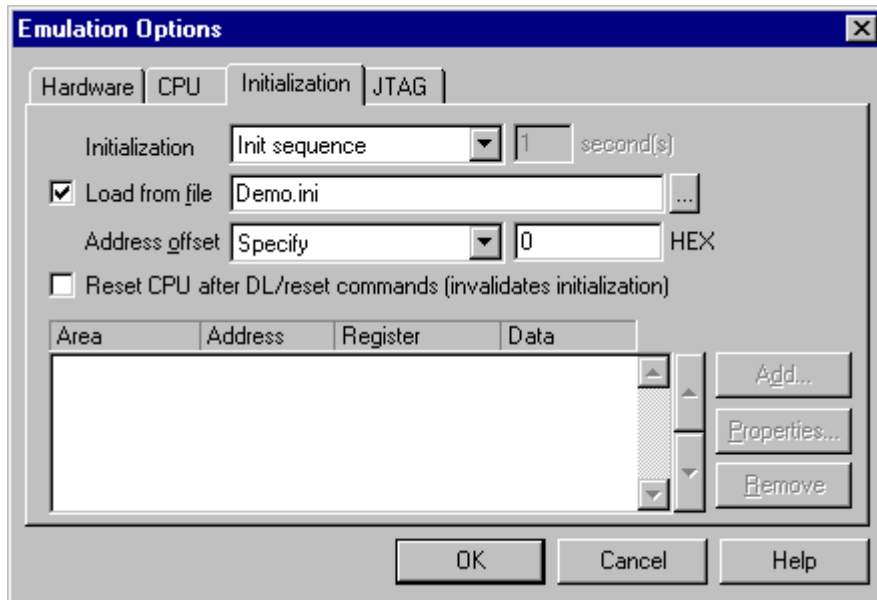
Note that registers displayed in the disassembly window (e.g. D0-D7, A0-A7, ACC0, ACR0, RAMBAR, ...) are CPU core registers and are accessed through the debug interface while other peripheral registers are memory mapped registers. Make sure that correct access type is selected for the register when configuring it through the initialization sequence.

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

Excerpt from Demo.ini file:

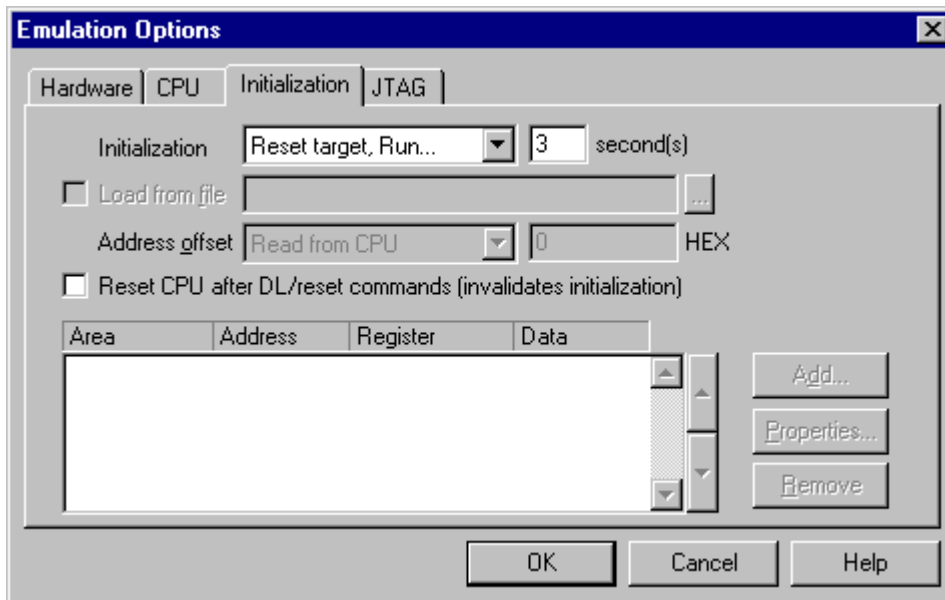
```
R RAMBAR L 0x20000001
S WCR W 0
```

```
//ext. flash at CS0 (address 0x00000000)
S CSAR0 W 0x00000000
S CSMR0 L 0x001F0001
S CSCR0 W 0x1980
```



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.

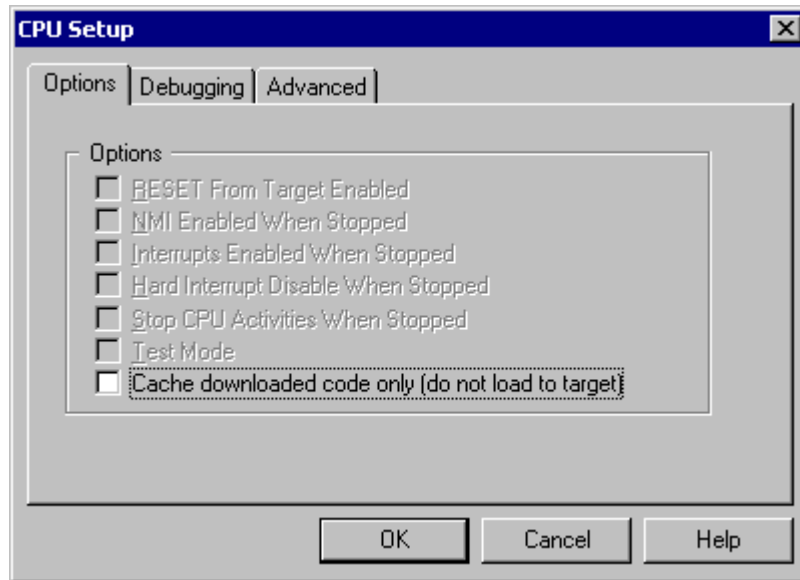
There is also a third method, which can be used too but it's not highly recommended for the start up. The user can initialize the CPU by executing part of the code in the target ROM for X seconds by using 'Reset and run for X sec' option.



3 CPU Setup

3.1 General 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.



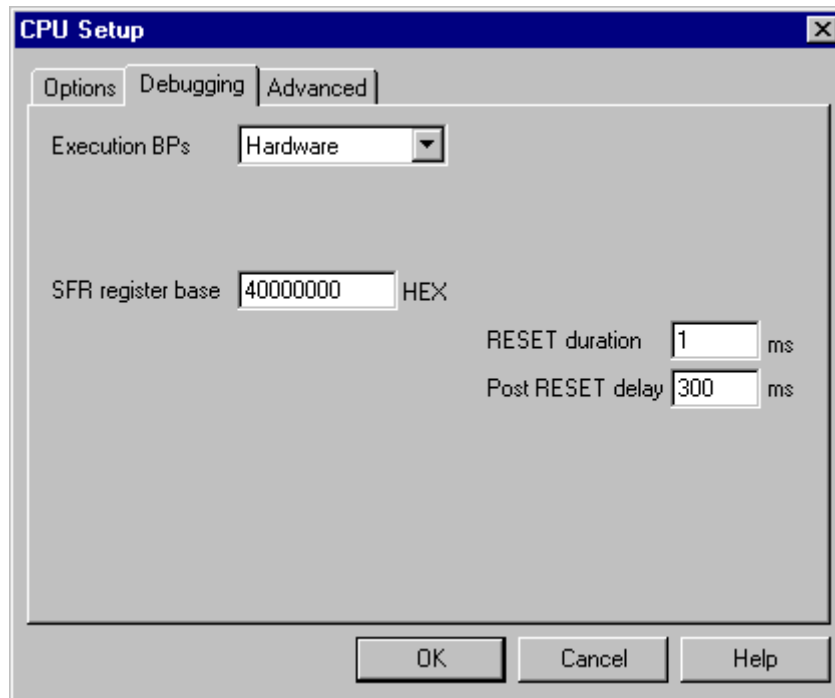
General Options

Cache downloaded code only (do not load to target)

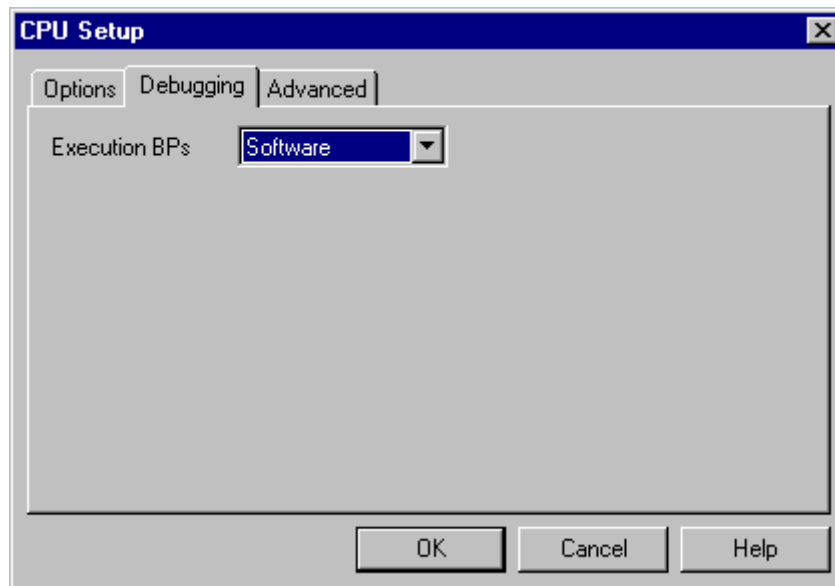
When this option is checked, the download files will not propagate to the target using standard debug download but the Target download files will.

In cases, where the application is previously programmed in the target or it's programmed through the flash programming dialog, the user may uncheck 'Load code' in the 'Properties' dialog when specifying the debug download file(s). By doing so, the debugger loads only the necessary debug information for high level debugging while it doesn't load any code. However, debug functionalities like the on-chip trace will not work then since an exact code image of the executed code is required as a prerequisite for the correct trace program flow reconstruction. This applies also for the call stack on some CPU platforms. In such applications, 'Load code' option should remain checked and 'Cache downloaded code only (do not load to target)' option checked instead. This will result in debug information and code image loaded to the debugger but no memory writes will propagate to the target, which otherwise normally load the code to the target.

3.2 Debugging options



ColdFire Debugging options (V2, V3, V4, V4e core)



ColdFire Debugging options (V1 core)

Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to four. 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-writable 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, 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.

Software Breakpoints

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation. Note that the debugger features unlimited software breakpoints in the CF5211, CF5212, CF5213, CF5214, CF5216, CF52210, CF52211, CF52212, CF52213, CF52221, CF52223, CF52230, CF52231, CF52233, CF52234, CF52235, CF5281, CF5282 and ColdFire V1 internal flash too.

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.

Using flash software breakpoints

A flash device has a limited number of programming cycles. Belonging flash sector is erased and programmed every time when a software breakpoint is set or removed. The debugger sets breakpoints hidden from the user also when a source step is executed. In worst case, a flash may become worn out due to intense and long lasting debugging using flash software breakpoints.

SFR register base

IPS value (SFR register base) used by the application must be entered for the debugger to be able to display SFRs since IPS value cannot be read by the debugger after it is relocated. SFR register base is 0x40000000 after reset. Note that this option does not apply for all CPUs and is disabled accordingly.

Note: This option is not available for ColdFire V1.

RESET Duration

The width of the RESET pulse is specified here.

Note: This option is not available for ColdFire V1.

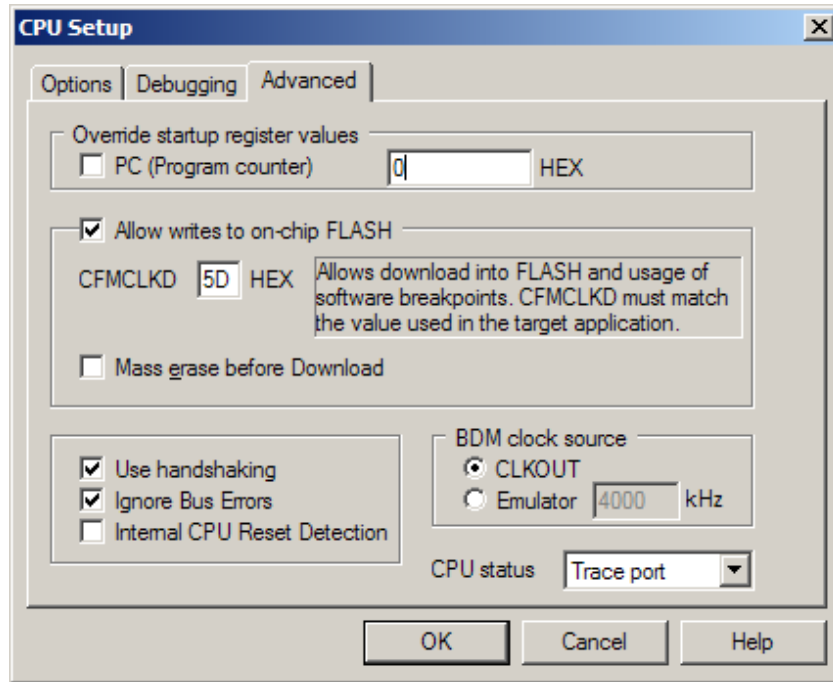
Post RESET Delay

Typically, the on-chip debug module is reset concurrently with the CPU. After the CPU reset line is released from the active state, the on-chip debug module requires some time (delay) to become operational. The default delay value normally allows the debugger to gain the control over the CPU. If a first debug connection fails already try different delay values to establish the debug connection.

Note: This option is not available for ColdFire V1.

3.3 Advanced Options

3.3.1 ColdFire V2, V3, V4 and V4e core



ColdFire Debugging options (V2, V3, V4, V4e Core)

Override startup register values

If required, the debugger can change the CPU program counter after the CPU is released from reset.

Allow writes to on-chip FLASH

This option applies only for the CF5211, CF5212, CF5213, CF5214, CF5216, CF52210, CF52211, CF52212, CF52213, CF52221, CF52223, CF52230, CF52231, CF52233, CF52234, CF52235, CF5281, CF5282 devices.

If this option is checked, the writes to the on-chip FLASH are allowed. In this case, the CFMCLKD value, normally used in the target application, must be written in the dialog. Note that the register is a write-once register and can not be modified later in the application.

When 'Mass erase before download' option is checked, the debugger performs a mass erase prior to debug download into the internal CPU flash. Note that a mass erase is only possible when no PROTECT bits (CFMPROT register) are set for that block.

Use handshaking

By default, this option is checked ensuring successful download. Each memory write BDM command is checked for successful completion. When no handshaking is used, download speed is increased by up to 3 times. Verify download can be used to verify whether the "no handshaking" download was successful. If verify errors are reported, revert to the default settings (the 'Use handshaking' option checked).

Ignore Bus Errors

Bus Errors are ignored if this option is checked.

Internal CPU Reset Detection

A CPU reset line (RSTI pin) is connected to the target debug connector and is input to the CPU only. The CPU provides a separate CPU reset output (RSTO pin) but per default it's not connected to the target debug connector. Due to this, the debugger has no way to detect internal CPU reset over the default target debug connection.

In order for the debugger to detect also the internal CPU reset, the user can connect the CPU RSTO to pin 1 of the target debug connector (otherwise not used) and check the 'Internal CPU Reset Detection' option.

Note: ColdFire iCARD (IC30114) must be revision I3 or newer when debugging through iC3000 and ColdFire iCARD. With older iCARDS this option is not supported.

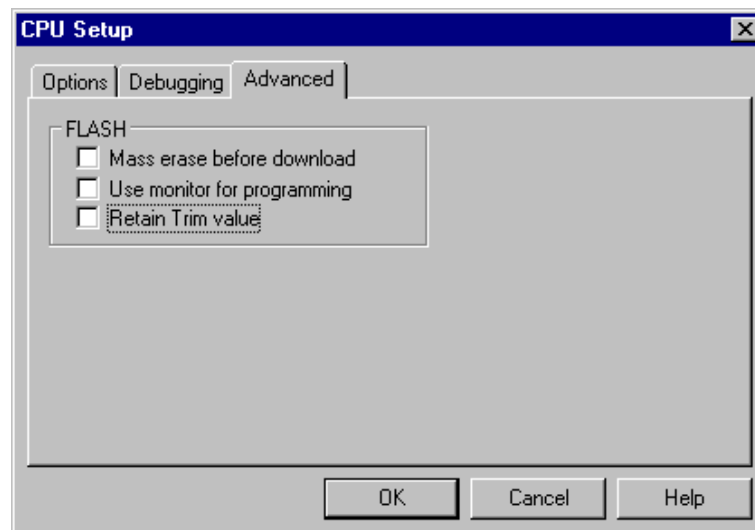
BDM clock source

This setting applies only for ColdFire iCARD and is disabled for iTRACE PRO/GT development system. When CLKOUT (default) is selected, the debugger's BDM engine is synchronized with the CPU output clock CLKOUT (sometimes named PSTCLK), which is present on the target BDM debug connector. However, some CPUs runs at 150, 200MHz and higher frequencies and in such cases the debugger fails to synchronize with the CLKOUT and then 'Emulator' clock must be selected and BDM frequency specified. Frequency should be less than half of the reset CPU clock. Try different values until you find a working setting. Higher frequency yields better debug performance.

CPU status

The debugger can recognize CPU status (run, stop) by inspecting the trace PST port or the CSR debug register. The disadvantage of the CSR debug register is that the status flag is cleared after it is read. It is recommended to use the trace port (normally connected to a standard 26-pin BDM debug connector) by default. However, there are CPUs without the trace PST port, where the debug register must be selected for the CPU status.

3.3.2 ColdFire V1 core



ColdFire Debugging options (V1 Core)

Mass erase before download

Check the option if complete flash should be erased prior to the debug download. Otherwise, only the necessary sectors are erased.

Use monitor for programming

If this option is checked, the FLASH is programmed through a special monitor loaded in the internal RAM; otherwise programming will be performed via BDM interface. It is recommended to check the option for better performance.

Retain Trim value

The CPU has a function which allows trimming the internal period of the internal reference clock. The trim value is stored in the internal flash and is internally read when the CPU is powered.. Refer to the CPU user manual for more details.

If this option is checked, the Trim value is read before the flash programming and restored back after the flash programming completes. This option is not available when 'Mass erase before download' is checked.

4 Real-Time Memory Access

ColdFire debug module supports real-time memory access. Watch window's Rt.Watch panes can be configured to inspect memory with minimum intrusion while the application is running. Optionally, memory and SFR windows can be configured to use real-time access as well.

Real-time access operation: BDM debug commands which reference memory must first request the bus from the core in order to perform the access. Included in this concurrent operation is an internal bus arbitration scheme which effectively schedules bus cycles for the debug module by stalling the processor's instruction fetch pipeline and then waiting until all operand requests have been serviced before granting the bus to the debug module. The debug module completes one bus transaction before releasing the bus back to the processor. There is an unverified information that there is a 4..6 cycles delay for every BDM memory access.

In general it is not recommended to use real-time access for Special Function Registers (SFRs) window. In reality, real-time access still means stealing some cycles. As long as the number of real-time access requests stays low, this is negligible and doesn't affect the application. However, if you update all SFRs or memory window via real-time access, you may notice different application behavior due to stealing too many CPU cycles.

When a particular special function register needs to be updated in real-time, put it in the real-time watch window (don't forget to enable real-time access in the SFRs window but keep SFRs window closed or open but with SFRs collapsed). This allows observing a special function register in real-time with minimum intrusion on the application.

Using "alternative" monitor access to update a memory location or a memory mapped special function register while the application is running works like this: the application is stopped, the memory is read and then the application is resumed. Hence the impact on real time execution is severe and use monitor access for 'update while running' only if you are aware of the consequences and can work with them.

5 Hot Attach

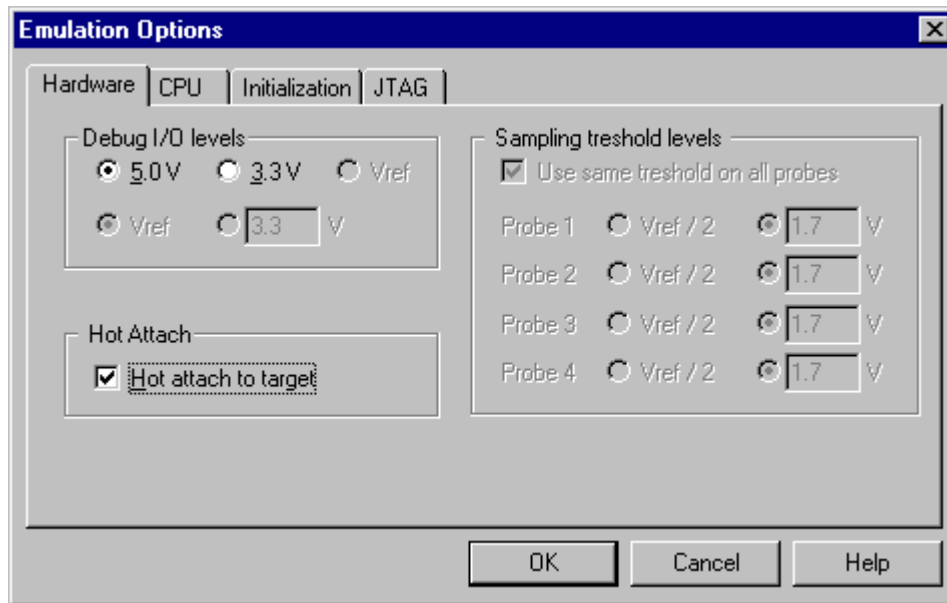
ColdFire BDM debug support includes a Hot Attach function, which allows the emulator to connect to a working target device and have all debug functions available. As such, it is a very convenient troubleshooting tool when the application misbehaves after a longer time. The user can connect to and inspect such an application after the problem pops-up.

Requirements:

- 1K ohm pull-down must be added on the DSCLK (BDM clock) CPU pin, which keeps the DSCLK pin at low level while the debugger is not connected to the target.

To hot attach to a running target with no debugger connected:

- Check the 'Hot attach to target' option in the 'Hardware/Emulation Options/Hardware' tab.
- Execute Download debug command.
- Connect the BDM cable to the target system
- Select the 'Attach' debug command in the 'Debug' menu to attach to the target system.



Now, the debugger should display run status and the application can be stopped and debugged.

Select 'Detach' debug command in the 'Debug' menu to disconnect from the target application. If the CPU was stopped before detach, it will be set to running.

Note: Hot Attach function cannot be used for any flash programming or code download!

6 Internal FLASH programming

6.1 ColdFire V2, V3, V4 and V4e core

Internal CPU flash programming directly through the debug download is supported on CF5211, CF5212, CF5213, CF5214, CF5216, CF52210, CF52211, CF52212, CF52213, CF52221, CF52223, CF52230, CF52231, CF52233, CF52234, CF52235, CF52252(C), CF52254(C), CF52255C, CF52256(C), CF52258(C), CF52259C, CF5281, CF5282 (check with iSYSTEM for the latest list) and devices based on ColdFire V1 core. Software breakpoints in the flash and flash modification through the memory window are supported too. Flash is modified by loading a small flash programming monitor into the internal CPU RAM, where it's then executed. Complete operation is hidden to the user and does not restrict any available CPU resource for the application.

In order to download the code into the internal flash, the user must (does not apply for ColdFire V1):

- Configure the FLASHBAR register using winIDEA initialization sequence unless the internal flash is already visible and enabled at the correct address after the CPU reset. It is recommended that the FLASHBAR register value from the application is used or a conflict will occur when flash modification is required while debugging the application. Flash is modified when using software breakpoints or modifying its content in the memory window.
- Configure the RAMBAR register using winIDEA initialization sequence unless the internal RAM is already visible and enabled at the correct address after the CPU reset. It is recommended that the

RAMBAR register value from the application is used or a conflict will occur when flash modification is required while debugging the application. Flash is modified when using software breakpoints or modifying its content in the memory window.

- Check the 'Allow writes to on-chip FLASH' option in the 'CPU setup/Advanced' tab.
- Calculate and enter the CFMCLKD register value in the 'CPU setup/Advanced' tab depending on the target CPU clock.

Double check that the CFMCLKD value matches with the target CPU clock during the debug download when the flash is being programmed. To issue any flash command, CFMCLKD register must be written to divide the flash input clock to within 150kHz and 200kHz range. Setting CFMCLKD to a value such that $FCLK < 150kHz$ can destroy the flash memory due to overstress. Setting CFMCLKD to a value such that $FCLK > 200kHz$ can result in incomplete programming or erasure of the flash memory array cells.

- Configure the CFMPROT register (sector protection) register using winIDEA initialization unless the necessary flash logical sectors are already not protected after the CPU reset.

Flash programming may fail when any of the above settings is not done or is configured improperly.

Refer to the CPU datasheet for more details on the ColdFire Flash Module (CFM) configuration.

The FLASH programming is executed automatically during the debug download and hidden from the user.

Maximum FLASH programming speed is achieved by unchecking 'Use handshaking' option in the 'Hardware/Emulation Options/CPU Setup/Advanced' tab but it may not work always.

Note: CFMDACC register must be configured properly to be able to run the code from the internal flash. By default the flash is mapped into the data address space. Refer to Microcontroller Reference Manual for more details on CFMDACC register.

Unsecure flash

Flash security can be enabled through the CFM Security register (CFMSEC). Enabling flash security disables BDM debug communication, which also means that the debug connection can no longer be established. Security protection can be removed only through the JTAG interface by erasing the entire CPU flash.

During BDM debug session, JTAG_EN pin is tied to low and CPU PSTCLK/TCLK pin is connected to pin 24 of the target debug connector. To unsecure the flash, the target must be reconfigured for the JTAG operation. JTAG_EN pin must be tied to high and CPU PSTCLK/TCLK pin must be connected to pin 6 of the target debug connector. Refer to the Microcontroller Reference Manual for more details. After a complete system including the debug tool is powered up, press the Hardware/Flash->Unsecure button. This operation takes approximately 40s. When it's done, reconfigure the target back for the BDM operation and new debug session can be started.

6.2 ColdFire V1

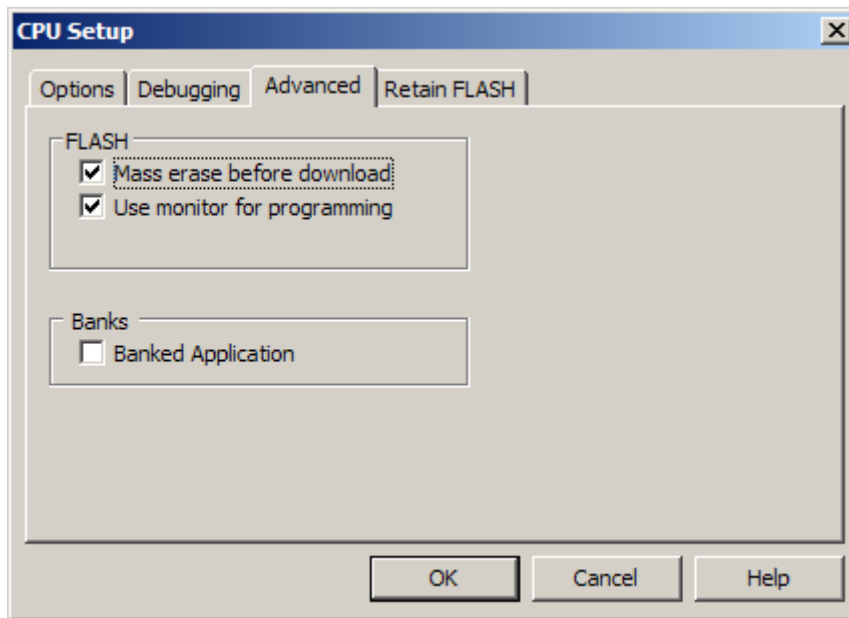
Coldfire V1 internal flash is programmed through the standard debug download. The debugger identifies which code from the download file fits in the flash and programs it accordingly. All necessary flash programming settings are done in the 'Hardware/Emulation Options/CPU Setup/Advanced' dialog.

Internal flash can be programmed in two ways:

- use default programming through BDM debug interface

When the regular BDM mode is used, the debugger executes flash programming algorithm and feeds the data to be programmed via the BDM debug interface. This method is relatively slow since BDM debug interface is a single wire serial communication.

- Use monitor for programming

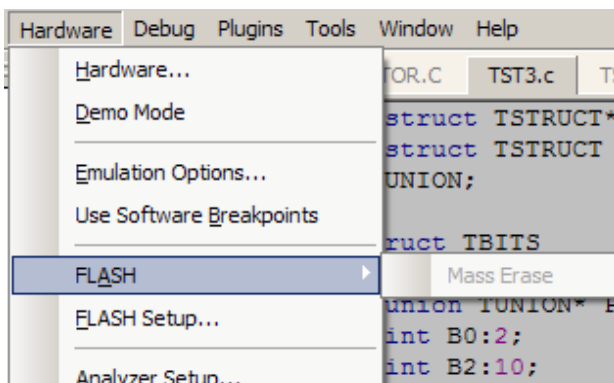


CPU Setup/Advanced dialog

First a small programming monitor is loaded in the internal RAM and then BDM debug interface is used only to feed the data to be programmed into the allocated monitor data buffer. Flash programming algorithm is executed fast since the monitor is executed by the CPU. All this operation is performed hidden from the user, which doesn't need to pre-configure anything except for checking the 'Use monitor for programming' option in the 'CPU Setup/Advanced' dialog. Note that no restrictions apply to the application.

Per default, the debugger erases only sectors to be programmed. If there is a need that a complete flash is erased prior to programming, check the 'Mass erase before download' option in 'CPU Setup/Advanced' dialog.

User can also manually perform mass erase via the 'Hardware/FLASH/Mass Erase' button.



When 'Use monitor for programming' is selected, the emulator requires internal CPU RAM for flash programming (but not later when the application is run). During the debug download, only the code which fits into internal FLASH is loaded. If you want to load a part of code to the internal CPU RAM, 'target download' must be used. During the target download, only code which doesn't fit into internal FLASH is loaded to the CPU. When regular BDM mode (the 'Use monitor for programming' option unchecked) is selected, the emulator loads the code to the internal RAM also during the download and the target download is not required.

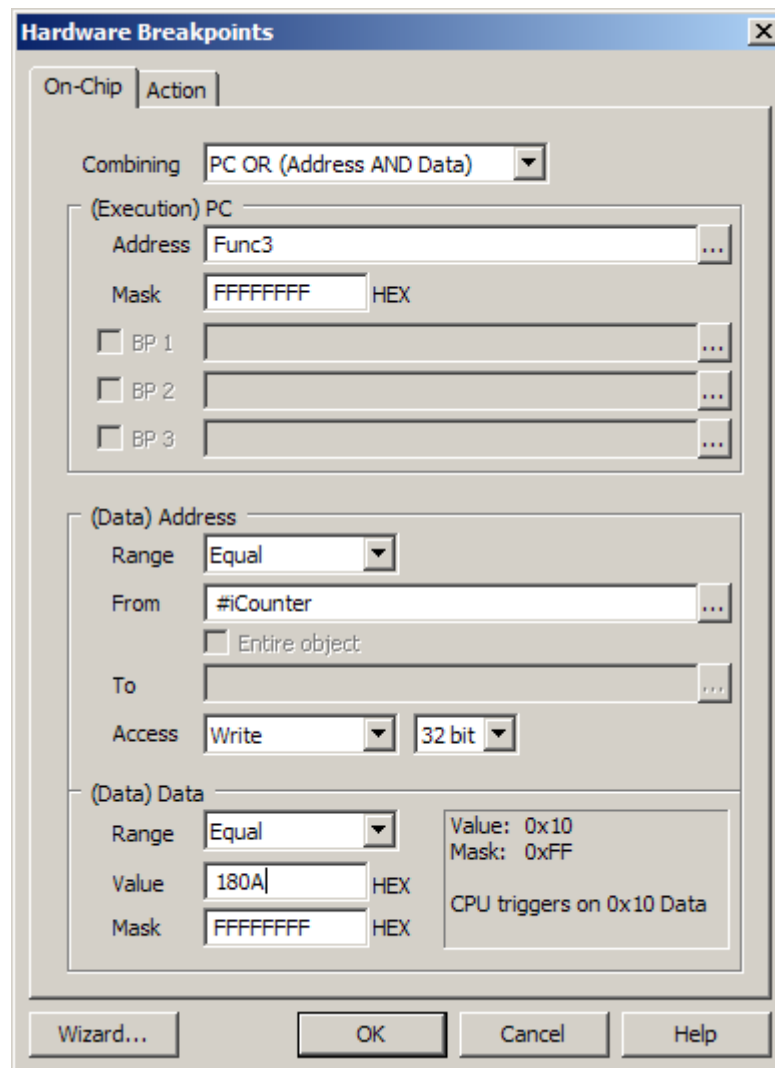
7 Access Breakpoints and Trace Trigger

Access breakpoints and on-chip trace trigger share the same on-chip debug resources, which means that the same CPU event can be either configured for access breakpoint or trace trigger. Consequentially, trace and access breakpoints cannot be used at the same time.

7.1 ColdFire V2, V3, V4 and V4e core

In the Rev. A debug module, certain hardware structures are shared between BDM and breakpoint/trigger functionality. Loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, a BDM command to access memory overwrites an address breakpoint/trigger. **This means that the user must not use real-time access (RT watch pane) while using access breakpoints or trace trigger! Double check that 'Update when running' option is disabled for real-time access while using trace!** Rev. B, B+, C and D debug modules do not have this restriction.

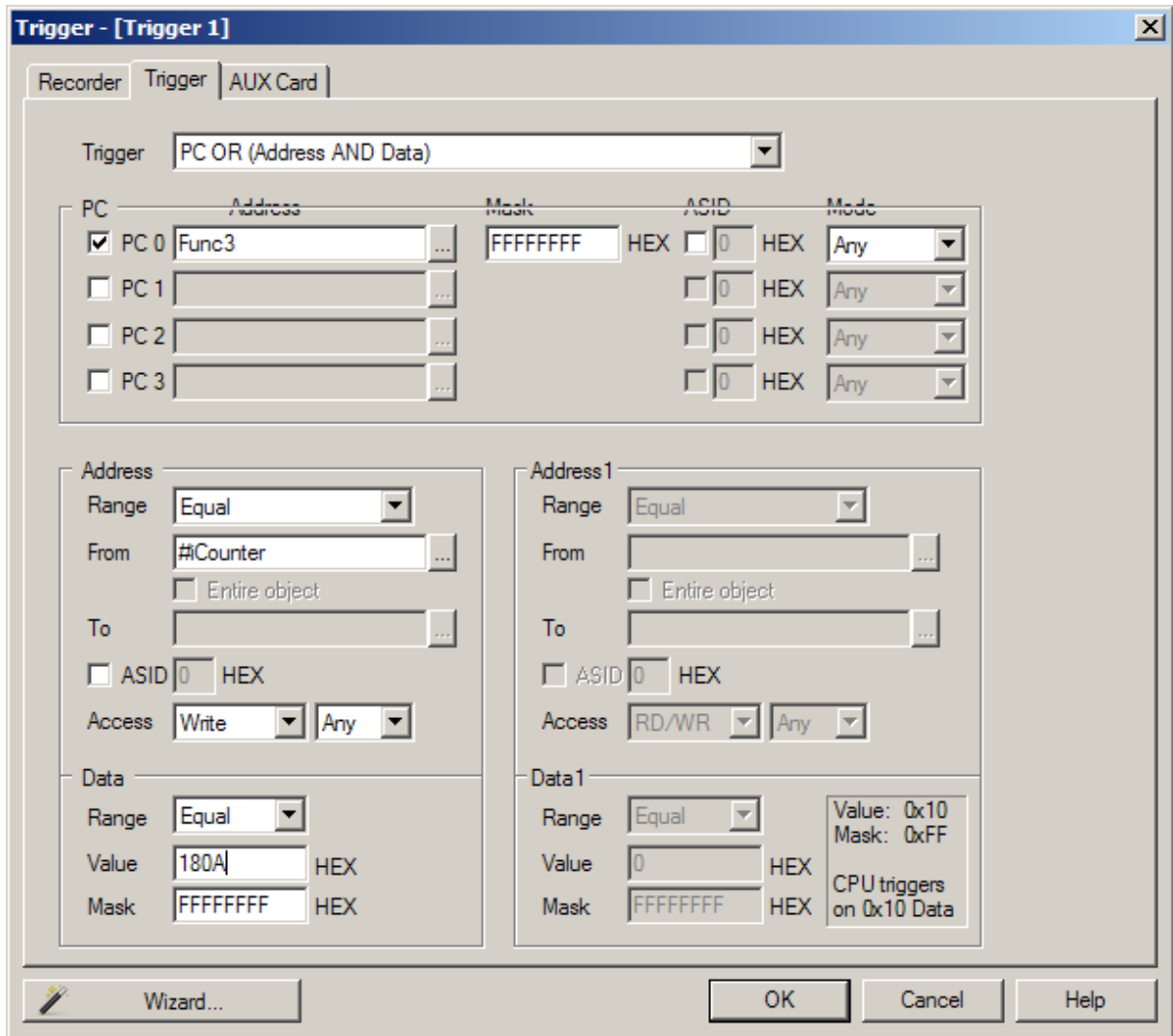
Depending on the integrated debug module revision, there are two types of breakpoint/trace configuration dialogs, and not all options are enabled on all CPUs, depending on the on-chip resources available for the selected CPU.



Access Breakpoint Configuration: Debug module Rev. A, B and B+

For CPUs featuring Rev. A, B and B+ debug module, Access Breakpoint / Trigger can be configured as 1 or 2-level logical combination of PC (execution address), Address (address being accessed) and Data (data being accessed).

For CPUs featuring Rev. C and D debug module, Access Breakpoint / Trigger can be configured as 1 or 2-level logical combination of PC (execution address), Address/Address1 (address being accessed) and Data/Data1 (data being accessed).



Trace Trigger Configuration: Debug module Rev. C and D

Trigger / Access Breakpoint mode combining

Anything

Triggers on anything

PC

Trigger when PC condition matches.

Address

Trigger when (Data) Address condition matches

Data

Trigger when (Data) Data condition matches.

Address and Data

Trigger when configured address, data and access type match within the same bus cycle.

PC OR Address

Trigger when PC is reached or (Data) Address condition is matched within the same bus cycle.

PC OR (Address AND Data)

Trigger when PC is reached or (Data) Address and (Data) Data conditions are matched within the same bus cycle.

PC THEN Address

Trigger when first PC is reached and then (Data) Address condition is matched.

PC THEN Data

Trigger when first PC is reached and then (Data) Data condition is matched.

PC THEN (Address AND Data)

Trigger when first PC is reached and then (Data) Address and (Data) Data conditions are matched.

Address THEN PC

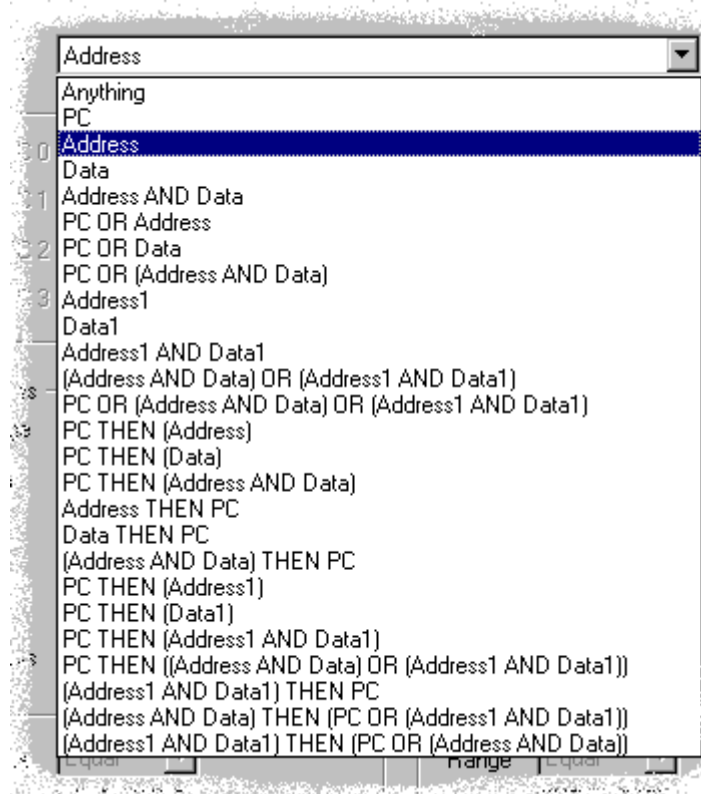
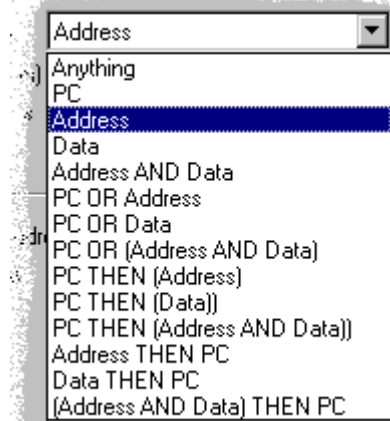
Trigger when first the (Data) Address condition is matched and then PC is reached.

Data THEN PC

Trigger when first the (Data) Data condition is matched and then PC is reached.

(Address AND Data) THEN PC

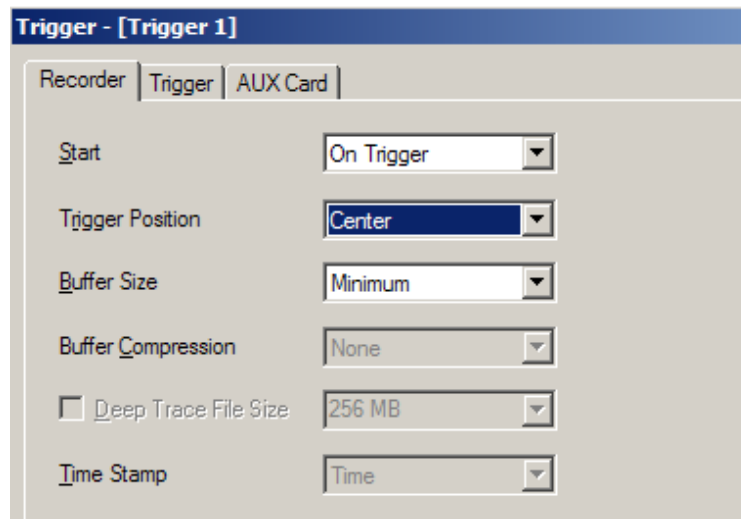
Trigger when first both the (Data) Address and (Data) Data conditions are reached and then PC is reached.



Extended configuration options

The extended configuration options of Rev. C and Rev. D on-chip debug resource logic implements additional options to configure breakpoint and trigger events. The improved resources include Address1 and Data1 conditions and consequently there are more combinations of the conditions available.

Recorder (Trace)



Start

Depending on the setting the Trace can start recording immediately or on trigger or trace can be configured for a so called Continuous Mode.

Typical use case is start trace recording on a trigger event where focus is around the configured trigger event. Alternatively the trace can be configured to stop recording on a program stop where focus is on program just before application stop. 'Continuous mode' use allows roll over of the trace buffer, which results in the trace recording up to the moment when the application stops. In practice, the trace displays the program flow just before the program stops. For instance, due to the breakpoint hit, due to the stop debug command issued by the user or due to the erratic state of the CPU which initiated the application stop.

Trigger Position

Depending on the needs, trigger can be located at the beginning (located at $1/64^{\text{th}}$ of the trace buffer), in the center (located in the middle of the trace buffer) or at the end of the trace buffer (located at $63/64^{\text{th}}$ of the trace buffer). Note that the trigger position may not configurable for minimum buffer. If the user intends to analyze the trace record after the trigger, it makes sense to use 'Begin' trigger position and 'End' trigger position when the trace pre-history that is program behavior before trigger is required.

Buffer Size

Minimum setting allocates $1/64$ of physical trace buffer for tracing, Medium allocates $1/4$ and Maximum allocates complete trace buffer. Primarily trace capture time depends on this setting and later on trace upload time as well.

Configuring 'Address AND Data' condition

Below table shows relationships between processor address, access size, and location within the 32-bit data bus.

A[1:0]	Access Size	Operand Location
00	Byte	D[31:24]
01	Byte	D[23:16]
10	Byte	D[15:8]
11	Byte	D[7:0]
0x	Word	D[31:16]
1x	Word	D[15:0]
xx	Longword	D{31:0}

When configuring trigger or access breakpoint on a data with specific data value, the debugger considers above table and configures on-chip debug resource accordingly.

(Data) Address

Range: Equal

From: (#g_c).c3

Entire object

To:

Access: Write 8 bit

(Data) Data

Range: Equal

Value: 94 HEX

Mask: FF HEX

Value: 0x10
Mask: 0xFF
CPU triggers on 0x10 Data

8-bit data value configuration

(Data) Address

Range: Equal

From: (#g_c).s1

Entire object

To:

Access: Write 16 bit

(Data) Data

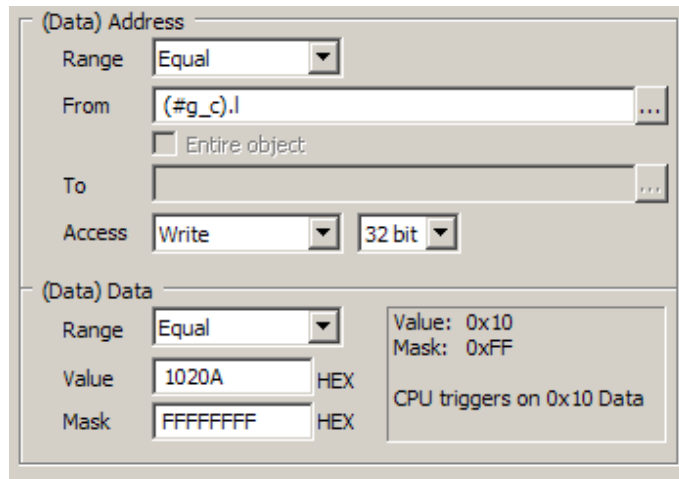
Range: Equal

Value: 0A12 HEX

Mask: FFFF HEX

Value: 0x10
Mask: 0xFF
CPU triggers on 0x10 Data

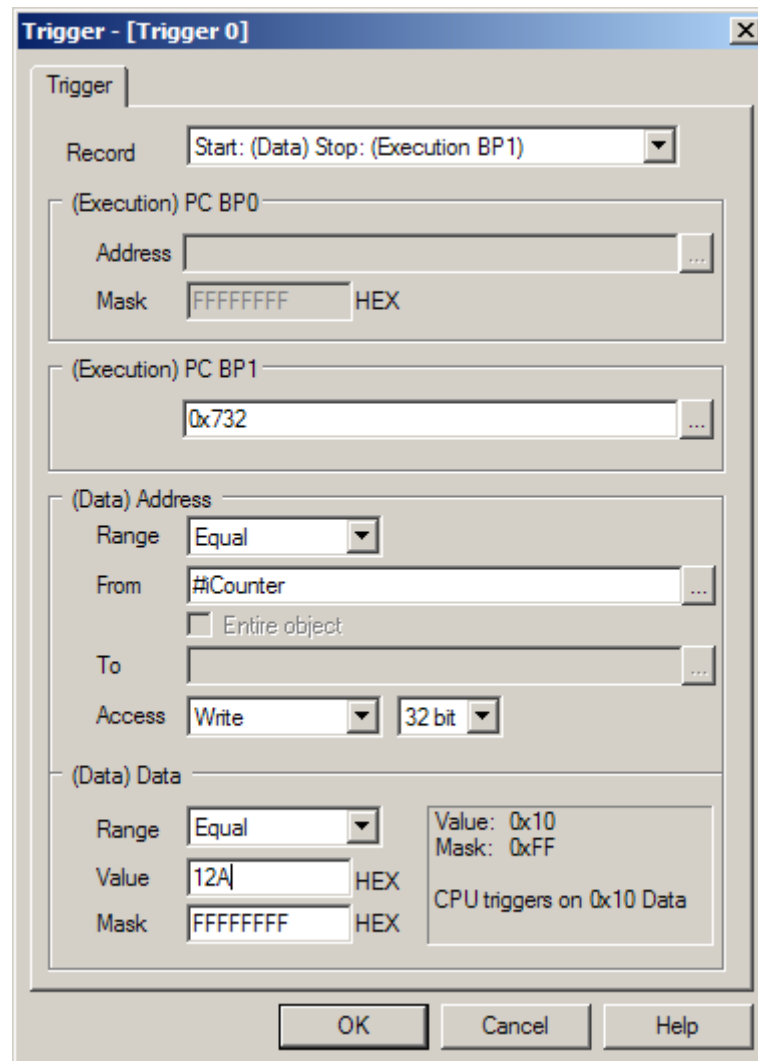
16-bit data value configuration



32-bit data value configuration

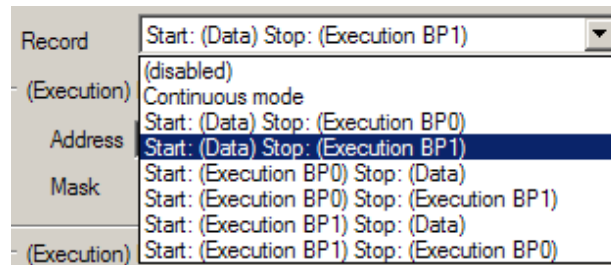
7.2 ColdFire V1

ColdFire V1 core has completely different on-chip trace comparing to other ColdFire cores. First of all, it has on-chip trace buffer and no trace port. On chip trace buffer can store 64 PST records, which are read out via the BDM debug interface.



Trace Trigger configuration

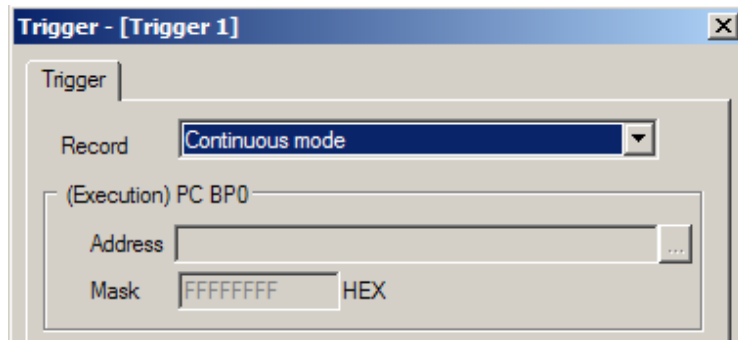
For the trace, start and stop condition is defined instead of the trigger event. Start and stop event can either be one of the two hardware execution breakpoints or a data access.



Trace Start/Stop condition

Note that the on-chip trace buffer cannot be uploaded until the trace start and stop condition are met. For instance, if start condition is met but stop condition hasn't met yet, trace status remains in sampling.

The alternative is to use Continuous mode, where the trace records until the program is stopped either due to execution breakpoint hit or what ever other reason.



8 Trace

ColdFire CPUs feature real-time trace, which allows the user insight into the program execution flow. The trace port is not available on all ColdFire devices. Typically, devices in small pin-count package don't have the trace port.

The ColdFire debug architecture ((V2, V3, V4, V4e core) implements an 8-bit parallel output bus that reports processor execution status and data to an external emulator system, which records the trace information in the trace buffer. Note that the processor status may not be related to the current bus transfer.

Devices based on ColdFire V1 core have no trace port, but there is limited on-chip trace buffer, which is read via debug interface.

External development system can use the trace output stream with an external image of the program to completely track the dynamic execution path. An accurate code image (download file) is a precondition for correct execution path reconstruction. This tracking is complicated by any change in flow, especially when branch target address calculation is based on the contents of a program-visible register. Trace outputs can be configured to display the target address of such instructions in sequential nibble increments across multiple processor clock cycles.

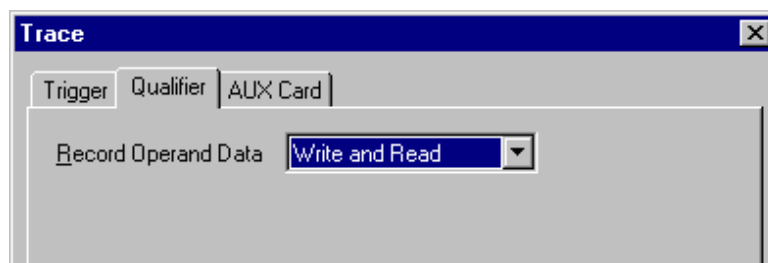
- Rev. A, B and B+ debug module

Two 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PST[3:0] and DDATA[3:0] trace port. Execution speed is affected only when both storage elements contain valid data to be dumped to the trace output. The core stalls until one FIFO entry is available.

- Rev. C and D debug module

Four 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PSTDDATA[7:0] trace port. Execution speed is affected only when three storage elements contain valid data to be dumped to the trace output. This occurs only when two values are captured simultaneously in a read-modify-write operation. The core stalls until two FIFO entries are available.

The real-time trace can additionally capture operand data. 'None', 'Write', 'Read' or 'Write and Read' can be selected for the Record Operand Data in the Qualifier tab in Trace trigger configuration dialog,



Operand Data Trace configuration

Note: Only operand data acting on the memory allocated outside of the CPU can be traced (e.g. external RAM). If the application data is allocated in the CPU internal RAM, no operand data is reported to the trace port in such case.

Additionally, ColdFire has a dedicated WDDATA instruction. This instruction fetches the operand (byte, half word or word size) defined by the effective address and captures the data in the ColdFire debug module for display on the trace port output pins. Refer to ColdFire Programmers Reference Manual for more details on the WDDATA instruction.

Below screenshot shows WDDATA instruction execution, which reports half word (16-bit) value 0x2233 to the trace port. Traced WDDATA points added to the application represents an equivalent technique to the well known printf() debug technique.

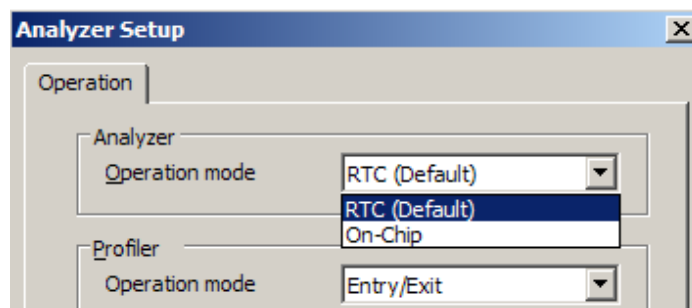
			4E75 RTS Instruction
-18.0	00001082	48782233	WDDATA16(0x2233); 48782233 PEA.L (2233).W Instruction
-17.0	00001086	4EB90000	4EB900001CA2 JSR (00001CA2).L Instruction
-17.1	0000108A	1CA2588F	Instruction
-13.0	00001CA2	FB6F0006	wddata.w (6,%a7) FB6F0006 WDDATA.W (0006,A7) Instruction
-13.1	00000000	00002233	Pulse wddata (0x00002233)
-10.0	00001CA6	4E75FB2F	rts 4E75 RTS Instruction
-8.0	0000108C	588F4EB9	588F ADDQ.L #04,A7

ColdFire V1 devices implements on-chip trace buffer that records processor execution status and data, which can be subsequently accessed by the external emulator hardware to provide program (and optional partial data) trace information. On-chip trace buffer provides programmable start/stop recording conditions.

The option 'Break on trigger' should not be used on ColdFire (except on V1 core). If the option is used, last part of the program executed prior to the trigger event is not displayed in the trace window. "Missing" program part cannot be reconstructed because the CPU stops sending the trace information as soon as the CPU is stopped even though there is still trace information to be sent out from the internal trace FIFOs. Note also that there is latency between the CPU bus transfer and belonging trace output information.

Real-Time Compression (RTC)

iC5000 and iC3000 based iTRACE GT development system feature a unique compression technology, which compresses the trace information captured on the microcontroller trace port before it's stored in the trace storage buffer. RTC allows capturing longer trace session with the same physical trace buffer comparing to the standard ColdFire on-chip trace.



'RTC' or standard 'On-chip' trace operation mode is configurable in the Hardware/Analyzer Setup dialog. By default 'RTC' use is recommended since it yields longer trace, profiler and coverage sessions. In case of any problems with the trace results, try also the 'On-Chip' trace operation mode to see if it makes any difference.

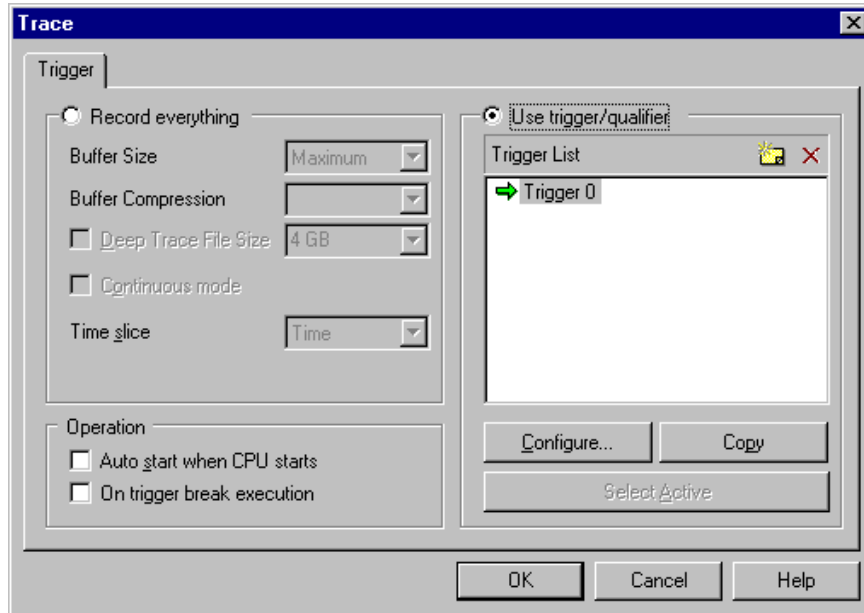
Note: RTC technology is supported only on ColdFire microcontrollers featuring debug module Rev. A, B or B+.

ColdFire V2 Trace Examples

Example 1: Trace triggers on a function `Type_Pointers`. Using trace, the program flow before the trigger event and after the trigger event can be inspected.

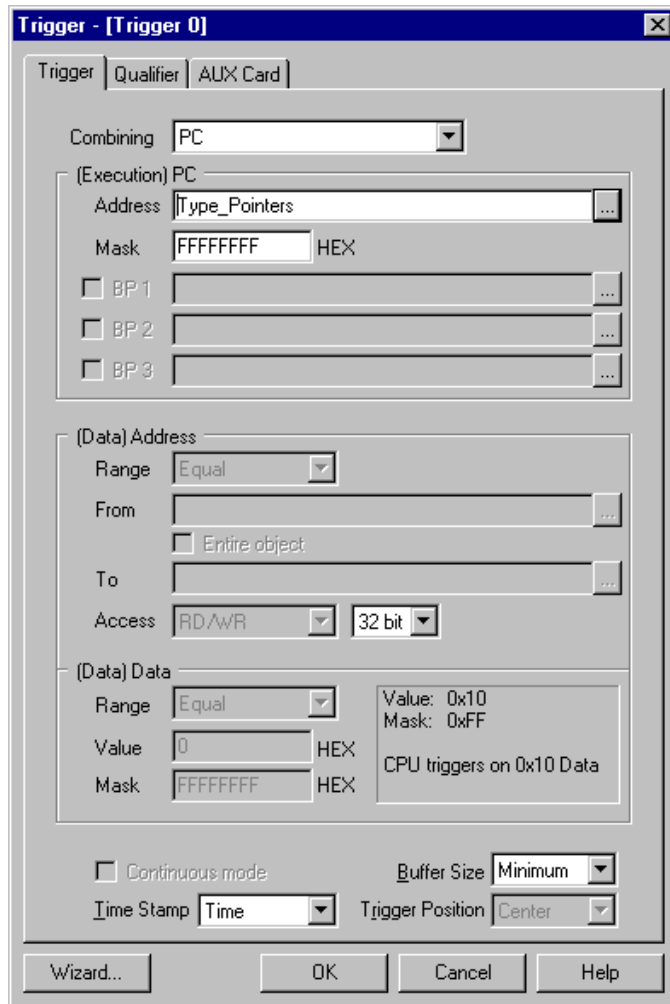
This is a very often used scenario. Trace triggers on a specific function or source line and the user can inspect the program execution before and after the trigger event.

Open *Trace configuration* dialog, select 'Use trigger/qualifier' operation type and define new trigger, named 'Trigger 0'.



Open *Trigger* configuration dialog by pressing the 'Configure... ' button and define a trigger event.

Select 'PC' condition in the 'Combining' field and select `Type_Pointers` function by pressing button on the right which invokes 'Symbol Browser'.

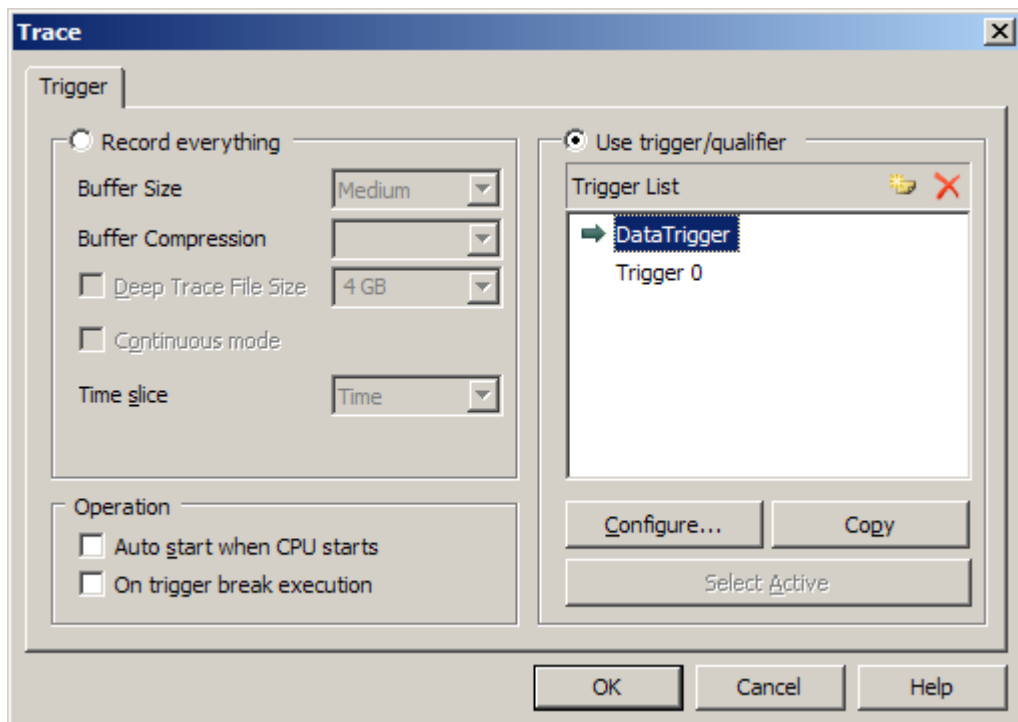


Now, execute *CPU reset*, start trace and run the program. After *Type_Pointers* is executed for the first time, trace window displays program flow before and after the entry to the function.

Number	Address	Data	Content	Time
-5	000011E6	5E0000	4E5E UNLK A6 Instruction	-2.611 us
-4	000011E8	750000	Type_Arrays_EXIT_ 4E75 RTS Instruction	-1.981 us
-3	0000104C	B90000	Type_Pointers(); 4EB9000011EA JSR (000011EA Instruction	-1.171 us
-2	00001050	EA0000	Instruction	-1.171 us
-1	000011EA	56FFF8	{ Type_Pointers 4E56FFF8 LINK.W A6,#-0008 Instruction	-801 ns
0	000011EE	3C0041	c='A'; 103C0041 MOVE.B #41,D0 Instruction	0 ns
1	000011F2	40FFFF	1D40FFFF MOVE.B D0,(-0001,A6) Instruction	0 ns
2	000011F6	EEFFFF	pC=&c; 41EEFFFF LEA.L (-0001,A6),A0 Instruction	750 ns
3	000011FA	48FFF8	2D48FFF8 MOVE.L A0,(-0008,A6) Instruction	820 ns

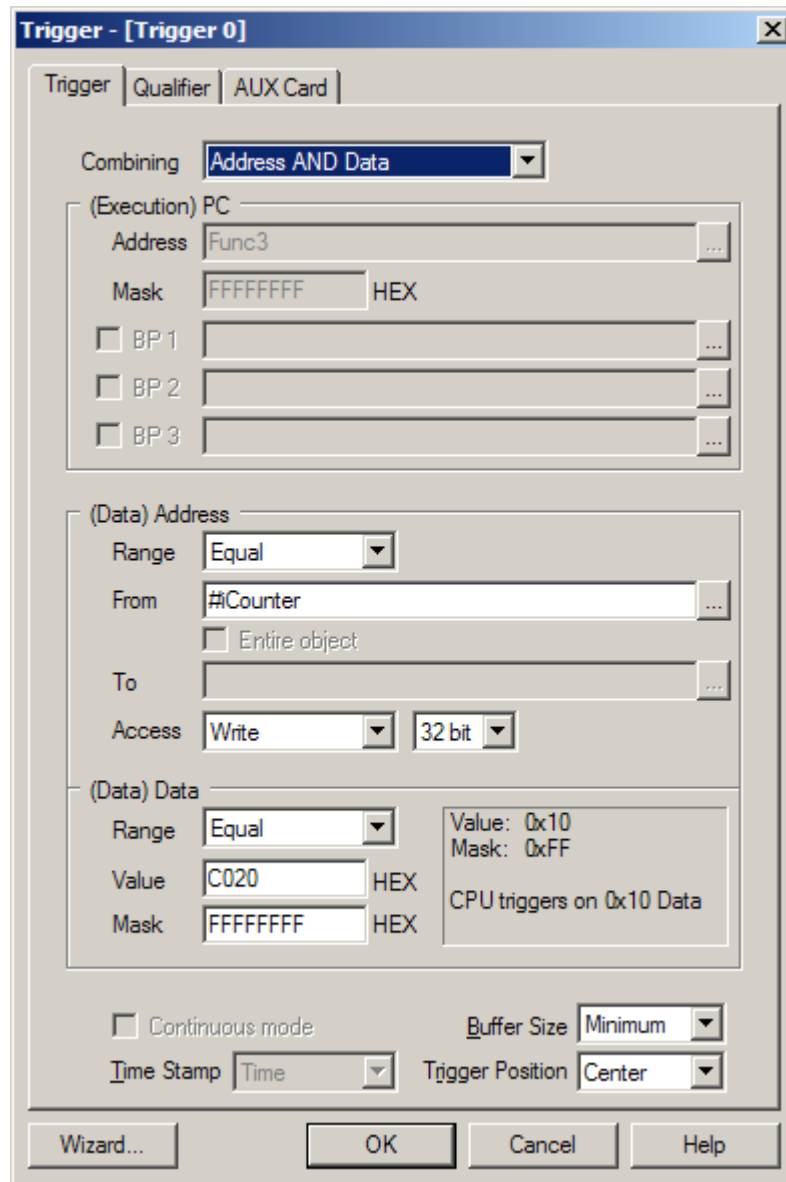
Example 2: The user gets incorrect writes to a particular address/variable. Example 2 demonstrates how to locate the code writing faulty value (0xC020) to a global variable iCounter.

Open *Trace configuration* dialog, select ‘Use trigger/qualifier’ operation type and define new trigger, named ‘Data Trigger’.

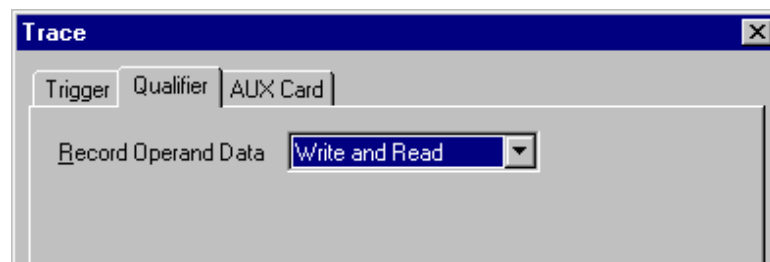


Open *Trigger configuration* dialog by pressing the ‘Configure...’ button and define a trigger event.

Select 'Address AND Data' condition in the 'Combining' field and select `iCounter` variable by pressing button on the right which invokes 'Symbol Browser'. Next, set 'Write' for the access type and finally enter data value `0xC020`.



Let's define a qualifier now. In our case, the trace should additionally capture data read and write for operands. By default Qualifier is set to 'None', which results in trace recording instructions only.



Execute *CPU reset*, start the trace and run the program. When faulty write occurs, a trace trigger event occurs simultaneously. The trace buffer fills up and the program flow is reconstructed and displayed. Since pre-trigger history is visible, the user can easily locate the code writing `0xC020` to the `iInterruptCounter` variable and fix the problem. There is always some misalignment between the actual trigger event and the one depicted in

the trace window (frame 0) due to the nature of the on-chip trace. In this particular case, instruction which generated the trigger event, holds marker -26.0.

Number	Address	Data	Content	Time
-27.0	00000760	1D40FFFF	1D40FFFF MOVE.B D0, (-0001,A6) Instruction	-600 ns
-27.1	00000764	7001D1B9	++iCounter; 7001 MOVEQ.L #01,D0 Instruction	-600 ns
-26.0	00000766	D1B90001	D1B900010024 ADD.L D0, (00010024).L Instruction	-580 ns
-26.1	0000076A	00244E5E	Instruction	-580 ns
-26.2	00000000	0000C01F	Data	-580 ns
-26.3	00000000	0000C020	Data	-580 ns
-13.0	0000076C	4E5E4E75	4E5E UNLK A6 Instruction	-260 ns
-11.0	0000076E	4E754E56	4E75 RTS Instruction	-220 ns
-8.0	000004CE	4EB90000	Type_Enum(); 4EB900000770 JSR (00000770).L Instruction	-160 ns

9 Profiler

In general from the functional point of view, profiler can be used to profile functions and/or data.

Note: Coldfire V1 doesn't feature profiler since the trace doesn't provide time stamp information

- **Functions Profiler**

Functions profiler helps identifying performance bottlenecks. The user can find which functions are most time consuming or time critical and need to be optimized.

Its functionality is based on the trace, recording entries and exits from profiled functions. A profiler area can be any section of code with a single entry point and one or more exit points. Existing functions profiler concept does not support exiting two or more functions through the same exit point. Exit point can belong to one function only. In such cases, the application needs to be modified to comply with this rule or alternatively data profiler with code arming can be used in order to obtain functions profiler results.

The nature of the functions profiler requires quality high-level debug information containing addresses of function entry and exit points, or such areas must be setup manually. Profiler recordings are statistically processed and for each function the following information is calculated:

- total execution time
- minimum, maximum and average execution time
- number of executions/calls
- minimum, maximum and average period between calls

- **Data Profiler**

While functions profiler is based on analyzing code execution, data profiler performs time statistics on the profiled data objects, which are typically global variables. Typical use cases are task profiler and functions profiler based on code instrumentation.

When an operating system is used in the application, task profiler can be used to analyze task switching. When the task profiler is used in conjunction with the functions profiler, functions' execution can be analyzed for each task individually.

Data profiler can be used on WDDATA instruction only. The operand data trace does not provide the required information for the data profiler.

The ColdFire development system features a so called off-line profiler. Off-line profiler is entirely based on the trace record. It first uses trace to record a complete program flow and then off-line, function entry and exit points are extracted by means of software, the statistic is run over the collected information and finally the results are displayed.

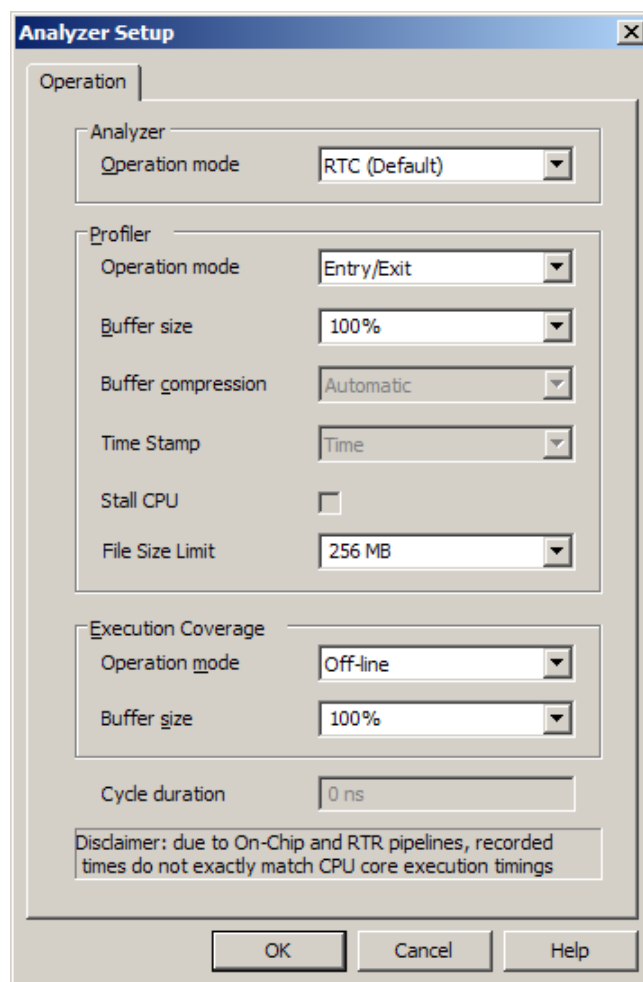
Refer to a separate document titled Profiler User's Guide for more details on profiler and its use.

Trace, Profiler and Execution Coverage functionalities cannot be used at the same time since they are all based on the trace. Single functionality can be used at the time only.

Be careful when including source lines in the offline profiler. A source line can often consists of a block of sequential instructions, which have all the same time stamp information due to the trace based on branch-trace concept. For instance, first instruction of the source line (entry) and last instruction (exit) will have the same time in such case and the profiler would display zero time spent in the source line although this is not the case in reality.

Typical Use

To use profiler, select working profiler buffer size in the 'Hardware/Analyzer Setup' dialog. Any value between 1% and 100% can be entered.



Next, select 'Profiler' window from the View menu and configure profiler settings (see next figure). Select 'Functions' option in the 'Profile' field when profiling functions.

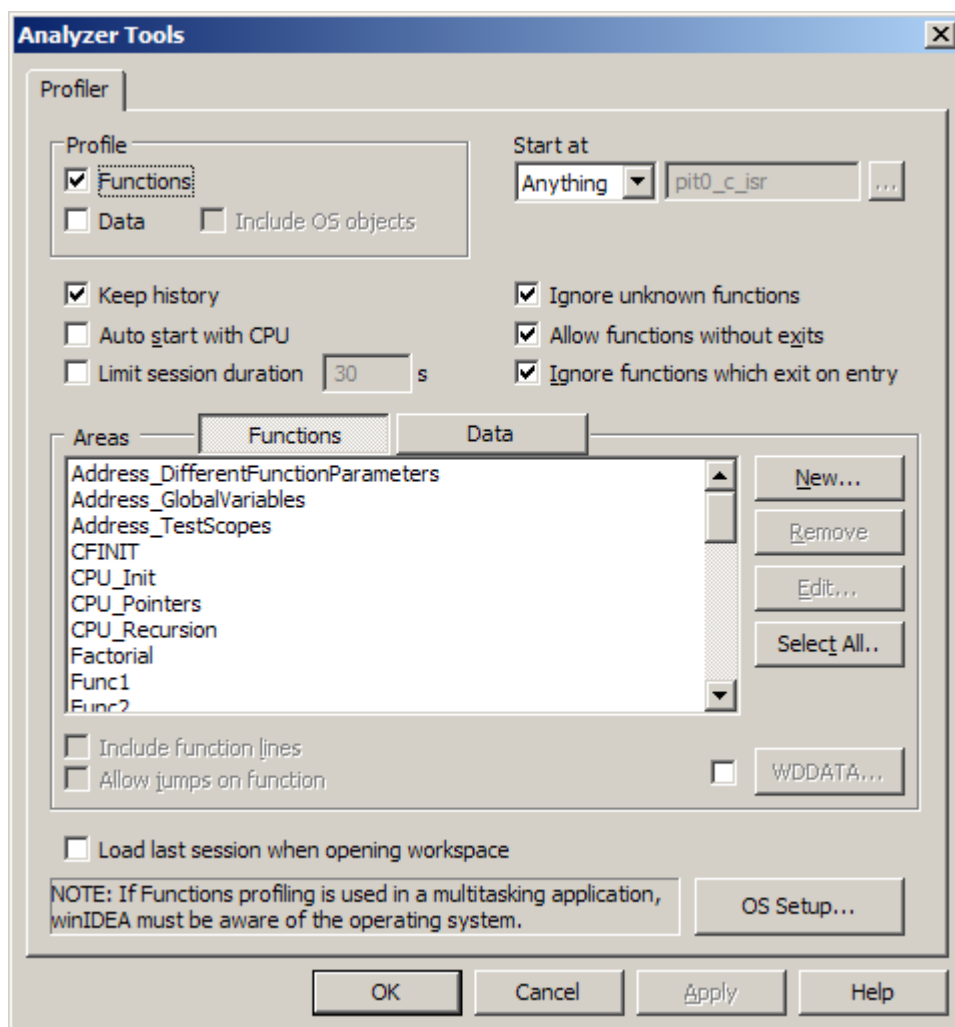
In order to profile debug dedicated WDDATA instructions, 'Data' should be checked in the Profile field and check box checked left from the WDDATA button. For instance, WDDATA Profiler can be used as a Task Profiler, if the operating system writes a unique task ID to the trace port via WDDATA instruction at every task switch.

When using functions profiler in application with operating system, the task switches ABSOLUTELY & UNCONDITIONALLY MUST be profiled too!

Make sure that 'Keep history' option is checked if History view is going to be used during the results analysis. If the option is unchecked, all recorded profiler data are discarded after the statistic information is calculated and history view shows no results.

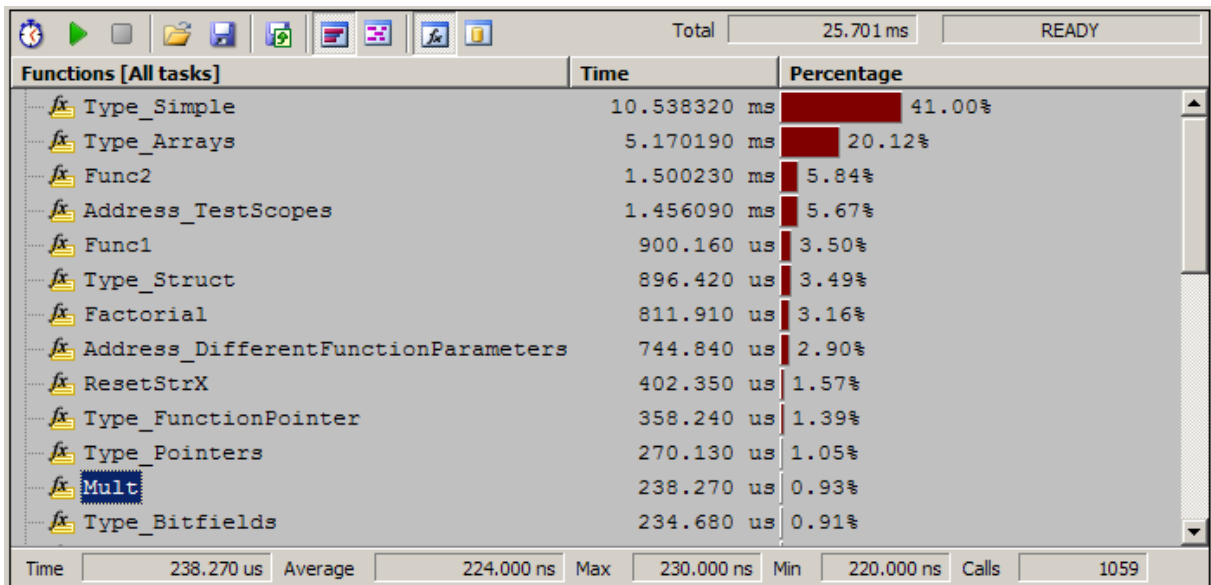
Finally, profiled functions are selected by pressing 'New...' button. It's recommended that 'All Functions' option is selected for the beginning.

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

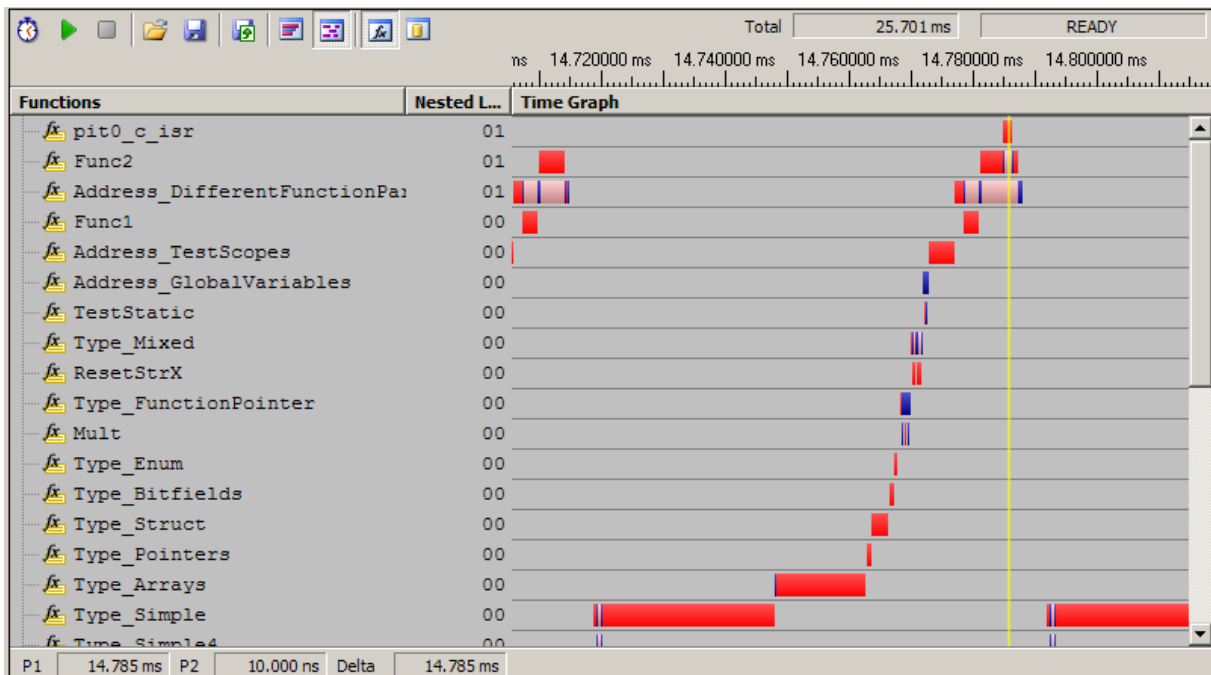


Profiler configuration settings

Profiler is configured. Reset the application, start Profiler and run the application. The Profiler will stop recording on a user demand or after the profiler buffer becomes full. While the buffer is uploaded, the recorded information is analyzed and profiler results displayed.



Statistics view



History view

10 Execution Coverage

Note: It doesn't make sense to use execution coverage on Coldfire V1, where only up to few thousands of program instructions can be recorded in the best case (depends on the application code). It can still be used, if execution coverage is required to be performed on a particular function or code section only.

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 the so called "dead code", the code that was never executed. Such code represents undesired overhead when assigning code memory resources.

The development system features a so called off-line execution coverage.

Off-line execution coverage is entirely based on the trace record. It first uses trace to record the executed code (capture time is limited by the trace depth) and then offline executed instructions and source lines are extracted by means of software and finally the results displayed.

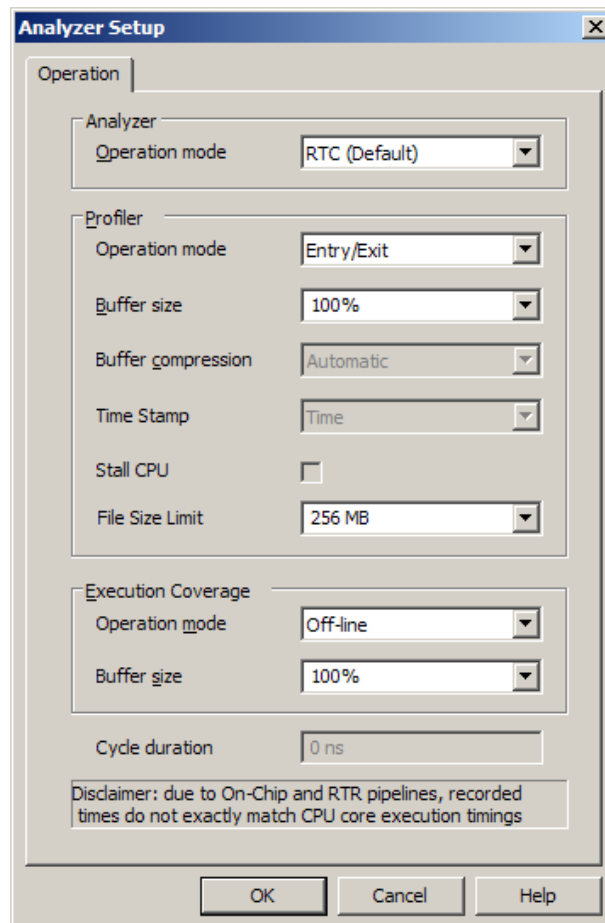
Off-line execution coverage tests the code for two metrics: statement coverage and decision coverage

Refer to a separate Execution Coverage User's Guide for more details on execution coverage configuration and use.

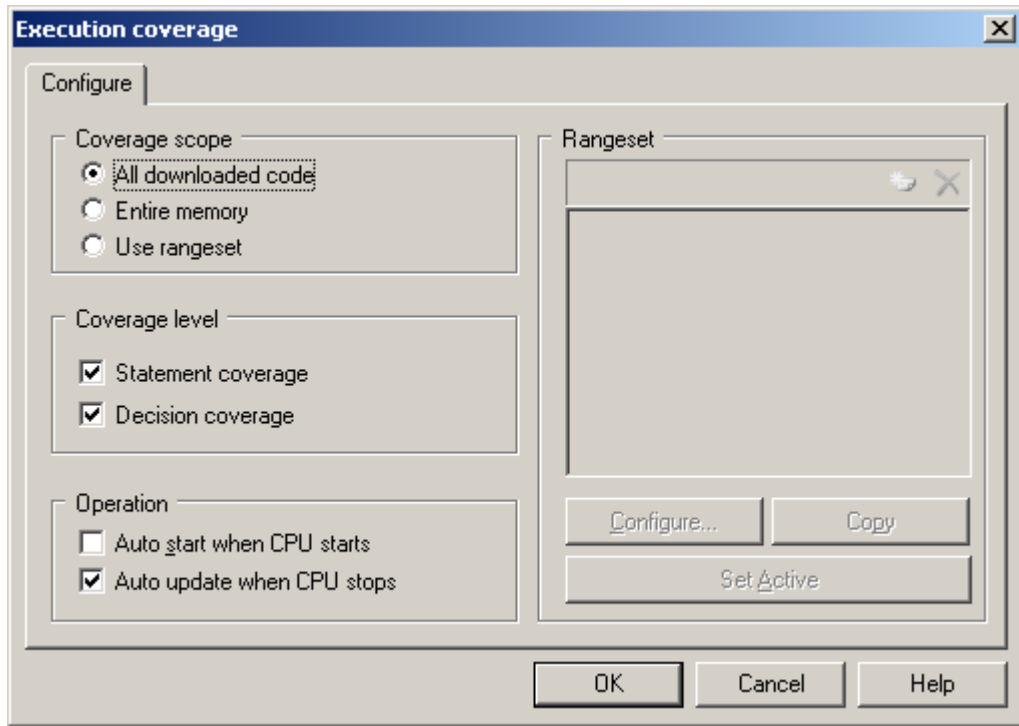
Trace, Profiler and Execution Coverage functionalities cannot be used at the same time since they are all based on the trace. Single functionality can be used at the time only

Typical Use

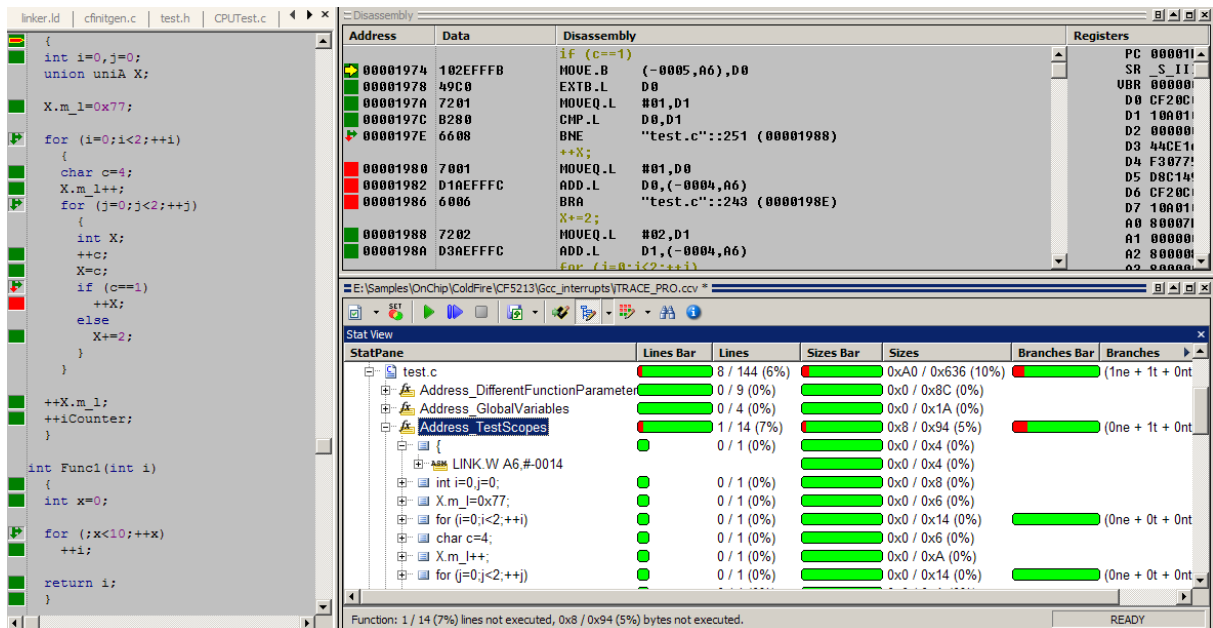
To use execution coverage, select working execution coverage buffer size in the 'Hardware/Analyzer Setup' dialog. Any value between 1% and 100% can be entered.



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.



Execution Coverage is configured. Reset the application, start Execution Coverage and then run the application. The debugger uploads the results when the trace buffer becomes full or when requested by the user.



Execution Coverage results

11 Emulation Notes

- Most of ColdFire CF52xx CPUs feature the Revision A enhanced hardware debug support while there are new CF52xx CPUs, which feature the Revision B/B+ enhanced hardware debug support. ColdFire CF54xx feature the Revision D enhanced hardware debug support. Since there is different debugging behaviour between them, the user should check specific Reference Manual (Debug Module chapter) in order to determine the Revision of enhanced hardware debug support that is in his specific CPU.

In the Revision A implementation of the debug module, certain hardware structures are shared between BDM and breakpoint functionality.

Register	BDM Function	Breakpoint Function
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for all BDM write commands	Data for data breakpoint

Rev. A Shared BDM/Breakpoint Hardware

Thus, loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, a BDM command to access memory overwrites an address breakpoint in ABHR. A BDM write command overwrites the data breakpoint in DBR.

All above facts are important during debug support implementation but at the end they impact the user when debugging his application. When CPU has Rev. A debug module, the user must not use access breakpoints nor trace trigger together with any memory referencing debug functionalities. Thereby, make sure that access breakpoints and trace trigger are not active (disabled) when using real-time watches, setting software breakpoints, modifying and reading memory, etc. In order to use access breakpoints or trace trigger, disable real-time access and use one available hardware execution breakpoint.

- ColdFire data trace is noticeably different from other on-chip Nexus based trace implementations supporting data trace. ColdFire trace was one of the first on-chip trace solutions and was implemented quite some time before Nexus standard was defined, which is widely adopted by silicon vendors in last few years.

ColdFire debug module provides a bit, which controls operand data capture for PSTDDATA (trace port). The user can opt for no operand data capture, capture all write data, capture all read data and capture all read/write data.

Let's first take a look how a data record looks in a Nexus based on-chip trace available for instance on Freescale MPC5500 family. The trace window shows concrete memory read and memory write access to the memory. For instance, frame 213 shows 32-bit memory write of value 0x00000004 to address 0x40002014.

Number	Address	Data	Content	Time
207	40005FD0	12345678	Write	94.327 us
208	40005FD0	11000000	Write	98.664 us
209	40005FD0	00220000	Write	100.327 us
210	40005FD0	00003300	Write	100.664 us
211	40005FD0	00000044	Write	101.664 us
212	40002014	00000003	iCounter iCounter+0 iCounter+0 iCounter+0 Read	103.318 us
213	40002014	00000004	iCounter iCounter+0 iCounter+0 iCounter+0 Write	104.318 us
214	0000079C	3C001234	uA.m_l=0x12345678L; 3C001234 addis r0,12340000 Instruction	104.655 us

MPC5500 Nexus trace record

Now, let's see the ColdFire trace record for a similar code, which reads 0x3 from 32-bit iCounter variable, increments it and writes back the new value (0x4).

Number	Address	Data	Content	Time
-29.0	00000718	7001D1B9	++iCounter; 7001 MOVEQ.L #01,D0 Instruction	-730 ns
-23.0	0000071A	D1B90001	D1B900010024 ADD.L D0,(00010024).L Instruction	-580 ns
-23.1	0000071E	00244E5E	Instruction	-580 ns
-23.2	00000000	00000003	Data	-580 ns
-23.3	00000000	00000004	Data	-580 ns
-10.0	00000720	4E5E4E75	4E5E UNLK A6 Instruction	-220 ns
-8.0	00000722	4E754E56	4E75 RTS Instruction	-180 ns

ColdFire CF5235 trace record

First of all, ColdFire trace displays operand data and not concrete memory read or write access. In this particular case, two data cycles, depicted as frame -23.2 and -23.3, belong to ADD.L D0, (0x0010024).L assembler instruction and not to the particular memory access. Chapter Debug Module in the CPU Reference Manual precisely describes what kind of operand data can each assembler instruction generate. In our example, first operand data (frame -23.2) represent source for the instruction and second operand data destination. Trace record does not directly show the address of the variable, which the application reads first and then writes to.

- Starting the trace while the CPU is already running

In case of ColdFire CF52xx CPUs, the data trace must not be used in order for the debugger to be able to synchronize program trace with the current CPU program counter.

ColdFire CF54xx CPUs have slightly modified trace implementation in the debug module comparing to CF52xx due to higher CPU frequencies. Due to these modifications, the debugger cannot synchronize program trace with current program counter even when the data trace is disabled. Hence, the trace needs to

be started before program run while the CPU is in stop. Only in this case the debugger can always synchronize properly program trace with actual program execution.

- Trace Trigger

ColdFire CF54xx trace implementation does not allow reliable hardware detection of the trigger event. The CPU transmits extra coding on the trace port for the trigger event but the same coding can be generated as part of a trace message indicating a change in program flow. Consequentially, when using trace trigger, the trigger marker more often than not points to an incorrect location in the trace record. Thereby, it is recommended to use the trace in 'Record everything' mode instead in 'Trigger' mode.

- Interrupts

It is not allowed to perform source step over the code, which modifies the interrupt level mask in the core SR register. Such step would yield wrong SR register value since interrupts are disabled (interrupt level raised to maximum) before every source step. Interrupts are disabled by the debugger to have a more predictable debugging of applications using interrupts. For instance, if there is a periodic interrupt and interrupts would not be disabled during the source step, the user would keep entering the interrupt routine while stepping.

- Watchdog

It is recommended to disable any internal or external watchdog to prevent conflicts with the debugger. Per default, the debugger cannot detect any internal watchdog reset and will lose the control over the CPU in case of internal CPU reset. Refer to chapter 3.3 for more details on 'Internal CPU reset detection' option.

12 Getting Started

- 1) Connect the system
- 2) Make sure that the target debug connector pinout matches with the one requested by a debug tool. If it doesn't, make the necessary adaptation to comply with the standard connector otherwise the target or the debug tool may be damaged.
- 3) Power up the emulator and then power up the target.
- 4) Execute debug reset
- 5) The CPU should stop on the reset location.
- 6) Enable internal CPU RAM by manually initializing belonging CPU register (RAMBAR).
- 7) Open memory window at internal CPU RAM location and check whether you are able to modify its content.
- 8) If you passed all 7 steps successfully, the debugger is operational. Now you may add the download file and load the code to the RAM. Read the '[Internal FLASH Programming](#)' chapter before downloading the code in the CPUs with the internal flash memory.
- 9) To program the external flash or download the code to the external RAM, which is not accessible after reset, make sure you use the initialization sequence to enable the access. First, the debugger executes reset, then the initialization sequence and finally the download or flash programming is carried out.

13 Troubleshooting

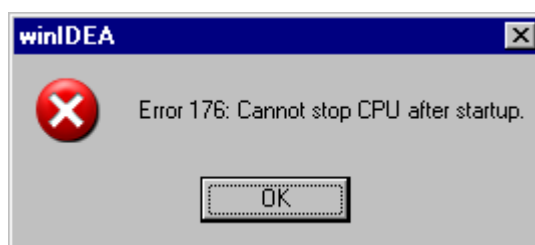
- I'm programming the code in the external flash. The flash is programmed correctly but after the download completes, I see bad content in the memory window. I get correct content again if I perform a debug reset afterwards. What's wrong?

Check the 'Cache downloaded code only (do not load to the target)' option on the 'Hardware/Emulation Options/CPU Setup/Options' tab. With current configuration, the debugger performs additional memory writes into the flash area (as part of the debug download) after the flash is being programming already. This seems to conflict with the CPU and yields bad starting point. With the above option checked, memory writes belonging to the debug download, do not propagate to the target flash. However, be careful not to use this option if you are trying to download something into the RAM. Refer to [chapter 3.3](#) for more details on this option.

- In case of problems with flash programming, double check that the CFMCLKD value matches with the target CPU clock during the debug download when the flash is being programmed. Refer to the [chapter 6](#) covering the flash programming and to CPU reference manual for more details on the CFMCLKD register.
- In case of any problems with the program flow in the trace or with profiler operation or with coverage operation, double check the settings in the 'Hardware/Analyze Setup/ColdFire' tab where trace branch target address is configured.
- When I trace my program I get nothing displayed in the trace window. What's wrong?

First of all make sure that the program image is provided to the debugger via the download file. Otherwise the trace cannot reconstruct the program. This could be an issue if the code would be run for instance from the preprogrammed external flash. Next, the "problem" may also come from the program. The debugger reads physical trace memory in blocks and can read block from any location in the trace buffer depending on the position of the scroll bar in the trace window. Sometimes it can happen that such block has no indirect branch message recorded, which reports the debugger the program counter from which point on a program reconstruction can start in the trace. This can easily happen with a long program loop or an idle loop waiting for an interrupt. Such code may not report any program counter information on the trace port but generate sequential instruction information only. Try to shorten the loop or remove such the code for the test and see if that's the problem.

- When Error 176 pops up, try to increase the post reset delay.



- Make sure that the power supply is applied to the target BDM connector when 'Vref' is selected for 'Debug I/O levels' in the 'Hardware/Emulator Options/Hardware' tab, otherwise emulation fails or may behave unpredictably.
- When initial debug connection fails, make sure that there is no capacitor on the reset line. If there is an external reset circuitry, it is recommended to temporarily disable/disconnect it while troubleshooting the initial debug connection.
- Try different setting for debug I/O levels ('Hardware/Emulator Options/Hardware' tab) in case of startup problems when debugging with ColdFire iCARD.

- When the debugger addresses reserved IPSBAR memory space, it will yield an unterminated bus cycle that causes the core to hang. Only a hard reset will allow the core to recover. This can easily happen when for instance Variable window is open during the debug session and an uninitialized pointer points to the reserved IPSBAR space. For example, on CF5213, IPSBAR window starts at 0x4000 0000 after reset and covers 1GB memory space. Special function registers are allocated at the begging of the window. Accessing area with no mapped registers (e.g. address 0xFFFF4000) will hang the core. To avoid this, it is recommended to block read accesses into this area by using the 'Debug/Debug Options/Memory Regions' dialog. For this particular case, blocking accesses into the 0x4500 000 – 0xFFFF FFFF memory range, would solve the problem. Make sure you don't block accesses into the special function registers accidentally. Note also that the IPSBAR space can be moved by the application through the IPSBAR register.
- External flash programming CF5407, CF5470, CF5471, CF5472, CF5473, CF5474, CF5475: Flash programming monitor must not be allocated in the internal SRAM. On these CPUs, internal SRAM is either only connected to the instruction bus or data bus, depending on the setting in the RAMBAR register. This means that the monitor cannot be executed from the SRAM and at the same moment being able to write into the SRAM (monitor data buffer).
- When performing any kind of checksum, remove all software breakpoints since they may impact the checksum result.
- When the trace content doesn't seem to match with the executed code:
 - Check if correct Sampling and debug threshold levels are set in the 'Hardware/Emulation Options/Hardware' tab.
 - Make sure that the 'Entire 4 bytes' option is set for the Branch target address in the 'Hardware/Analyzer Setup/ColdFire' tab.
 - Make sure that the image of the executed code is downloaded to the debugger since the trace reconstruction is based on it.
- In case of ColdFire V1 devices, it is possible to use data access address & data value for on-chip trace start/stop condition. However, data value does not work for on-chip trace while it works fine when configured for access breakpoint.
- MCF5282 errata V2.0 (check the latest errata if the issue is fixed): The V2 core used in MCF5282 adds support for separate user and supervisor stack pointers. The hardware implements an active stack pointer and an "other_stack_pointer". Whenever the operating mode of the processor changes (supervisor->user, user->supervisor), the processor hardware "exchanges" the active SP and the other SP. This exchange operation does not work when the processor mode is changed by a write to the SR from the BDM port. The hardware in the processor core required to process the BDM load_SR operation and enable the stack pointer exchange is missing. The exchange works properly when the SR is changed through software.
Workaround: Use software for any operations that require exchanging the stack pointers.
- Q: I download the file, which is in Elf/dwarf format. I don't get any verify errors. However, the code doesn't run. It seems like not all code was really loaded. What could be the problem?
A: In your particular case, 'Load Code from' in the winIDEA Elf/Dwarf Options dialog (click Properties after specifying the download file) is set to Program Header / Virtual.

As usual, the choice between virtual and physical addresses is compiler and linker configuration dependant. If 'not all code is loaded', the following procedure is advisable:

1. generate map file
2. in winIDEA, Elf properties, select 'Dump Elf header'
3. Compare map file to the PROGRAM HEADERS entries for VIRTADDR and PHYADDR and see which is suitable

In this particular case the PROGRAM HEADERS part looks like this:

```

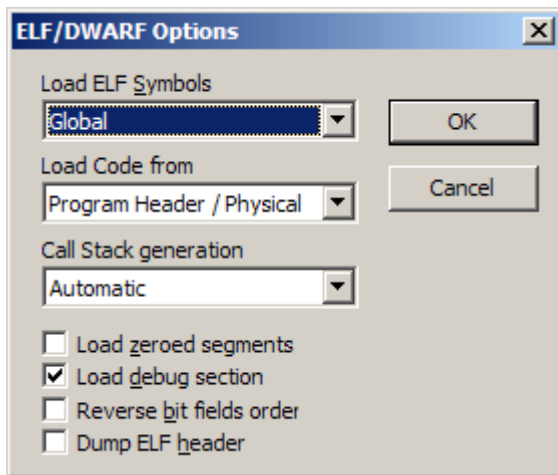
=====
PROGRAM HEADERS
=====

```

NUM	TYPE	OFFSET	SIZE	VIRTADDR	PHYADDR	MEMSIZE	FLAGS	ALIGNMNT
0	LOAD	174	300	0	0	300	5	4
1	LOAD	474	18	400	400	18	5	4
2	LOAD	174	0	500	500	0	5	4
3	LOAD	48C	724	500	500	724	5	4
4	LOAD	174	0	20000000	20000000	0	7	4
5	LOAD	174	0	20000400	20000400	0	7	4
6	LOAD	BB0	4	20000400	C24	4	7	4
7	LOAD	BB4	0	20000404	20000404	11C	7	4
8	LOAD	BB4	0	20000520	20000520	0	7	4
9	LOAD	BB4	18	20000520	C28	18	7	4

As you (should) know, a C/C++ application must initialize global data before entering main. The initialized data segment is copied from ROM to RAM, uninitialized (.bss) is simply zeroed.

On your specific CPU, ROM/FLASH resides on address 0, while RAM resides on address 20000000h. The program headers layout shows a few entries where PHYADDR is different to VIRTADDR. It is obvious that in this configuration the PHYADDR denotes the FLASH load location and VIRTADDR the link location (the address of RAM where variables will be accessed). Since apparently the application is PROMable (i.e. startup code will copy .initdata to .data), we must ensure that the FLASH is loaded with initialized data image and the correct choice is to select "Program Header / Physical"



Note that on average in 70% of cases Program Header / Virtual is the right choice, so winIDEA uses this setting per default.

Disclaimer: iSYSTEM assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information herein.

© iSYSTEM. All rights reserved.