
Technical Notes

Freescale MAC7100/MAC7200 Family On-Chip Emulation

Contents

Contents.....	1
1 Introduction	2
2 Emulation Options.....	3
2.1 Hardware Options.....	3
2.2 Initialization Sequence	4
2.3 JTAG Scan Speed.....	6
3 CPU Setup.....	7
3.1 General Options.....	7
3.2 Debugging Options.....	8
3.3 Advanced Options	9
3.4 PLL and FLASH Options	10
4 Hot Attach	11
5 Enhancing Debug Performance.....	12
6 Internal FLASH Programming	13
7 Internal FLASH Software Breakpoints	14
8 FLASH Security	14
9 Real-Time Memory Access	15
10 Access Breakpoints	16
11 On-Chip Trace	17
11.1 MAC7100 Trigger and Qualifier Configuration.....	18
11.2 MAC7200 Trigger and Qualifier Configuration.....	20
11.3 Examples	23
12 Getting Started.....	28
13 Troubleshooting.....	28

1 Introduction

The MAC7100/MAC7200 microcontrollers are members of a pin-compatible family of 32-bit Flash-based MCUs developed specifically for embedded automotive applications. This pin-compatible family concept enables users to select between different memory and peripheral options for scalable designs. All MAC7100 and MAC7200 members are built around 32-bit ARM7TDMI-S core including embedded Flash. The inclusion of PLL circuit allows power consumption and performance to be adjusted to suit operational requirements.

The debugger communicates with the on-chip debug module (EmbeddedICE) through the standard JTAG interface. The debug module provides all basic debug functions: Read and Write Memory, Read and Write Registers, Hardware Breakpoints, Run and Stop. Single step is not supported and is implemented by the debugger on a higher level. Real-time access is additional debug feature comparing to standard ARM debug features.

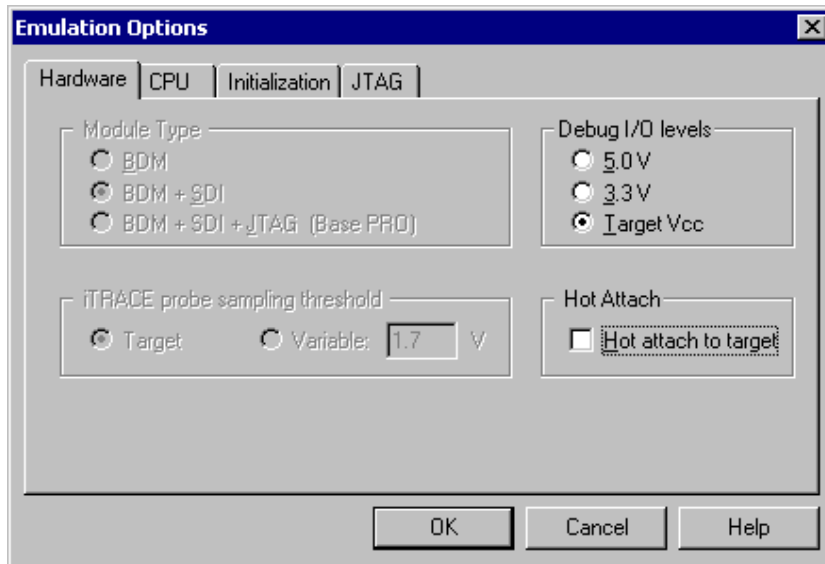
Debug Features

The MAC7100/MAC7200 debugging features:

- 2 hardware breakpoints
- Unlimited software breakpoints, including in the internal FLASH
- Access breakpoints
- Fast flash programming
- THUMB support
- Hot Attach
- Real-time Access
- On-Chip Trace

2 Emulation Options

2.1 Hardware Options



Emulation options, Hardware pane

Debug I/O levels

The development system can be configured in a way that debug (BDM/JTAG) signals are driven at 3.3V, 5V or target voltage levels. When 'Target Vcc' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin (target debug connector) is used as a reference voltage for driving debug (BDM/JTAG) signals. Make sure that the target reference voltage pin is connected when 'Target Vcc' Debug I/O level is selected

Hot Attach

The JTAG module supports the Hot Attach function. This is a function, which enables the emulator to be connected to a working target device and have all debug functions available.

The procedure for Hot Attach:

1. The target application should be running.
2. Hot Attach should be selected in the software.
3. A download should be performed, but without the JTAG cable connected. The emulator will be initialized and the ATTACH status will be shown.
4. Connect the JTAG cable.
5. Select the Attach option in the Debug menu. When this option is selected, the emulator tries to communicate through JTAG. If it is successful, it shows the STOP or RUNNING status. At this point, all debug functions are available.
6. When the debugging is finished, the CPU should be set to running and Detach selected from the Debug menu. The status shown is ATTACH. Now the JTAG cable can be safely removed.

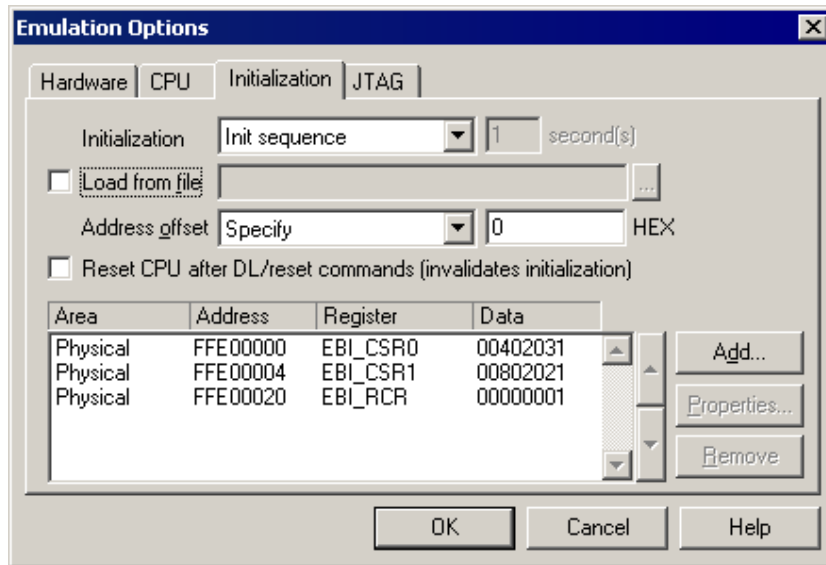
If Hot attach is used, please refer to Chapter 4 for more details.

2.2 Initialization Sequence

Before the flash programming or download can take place, the user must ensure that the memory is accessible. This is very important since there are many applications using memory resources (e.g. external RAM, external flash), which are not accessible after the CPU reset. In that case, the debugger must execute after the CPU reset a so called initialization sequence, which configures necessary CPU chip selects and then the download or flash programming can actually take place. The user must set up the initialization sequence based on his application.

The initialization sequence can be set up in two ways:

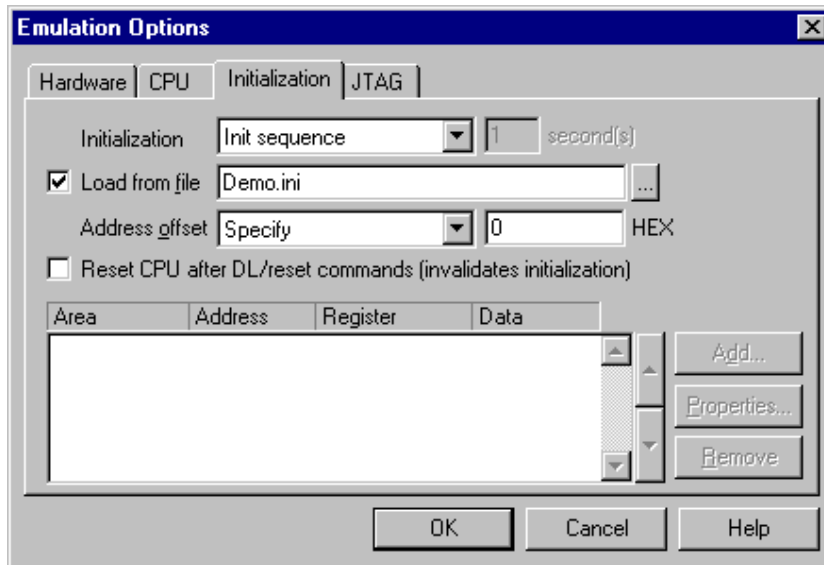
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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

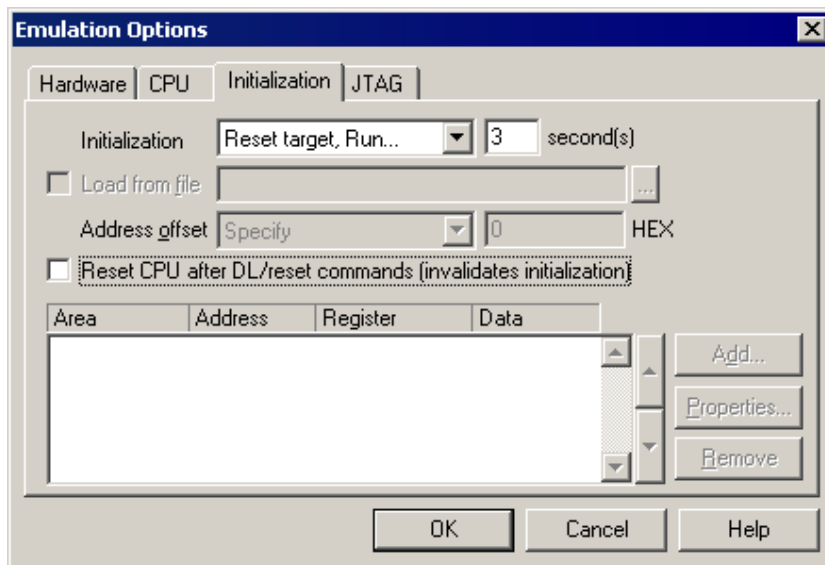
Excerpt from Demo.ini file:

```
S EBI_CSR0 L 0x00402031 // CS0 - ext. flash, 4 wait states
S EBI_CSR1 L 0x00802021 // CS1 - ext. SRAM
S EBI_RCR L 0x00000001 // remap internal RAM
```

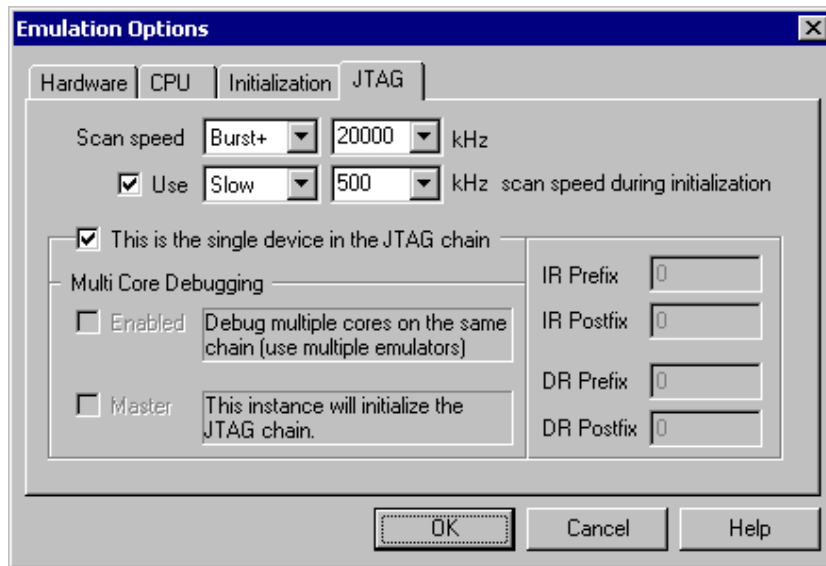


The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

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.



2.3 JTAG Scan Speed



JTAG Scan Speed definition

Scan speed

The JTAG chain scanning speed can be set to:

- Slow - long delays are introduced in the JTAG scanning to support the slowest devices. JTAG clock frequency varying from 1 kHz to 2000 kHz can be set.
- Fast – the JTAG chain is scanned with no delays.
- Burst – provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz.
- Burst+ - provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz

Slow and Fast JTAG scanning is implemented by means of software toggling the necessary JTAG signals. Burst mode is a mixture of software and hardware based scanning and should normally work except when the JTAG scan frequency is an issue that is when the JTAG scan frequency used by the hardware accelerator is too high for the CPU. In general, selecting an appropriate scan frequency usually depends on scan speed limitations of the CPU. In Burst+ mode, complete scan is controlled by the hardware accelerator, which poses some preconditions, which are not met with all CPUs. Consequentially, Burst+ mode doesn't work for all CPUs.

In general, Fast mode should be used as a default setting. If the debugger works stable with this setting, try Burst or Burst+ mode to increase the download speed. If Fast mode already fails, try Slow mode at different scan frequencies until you find a working setting.

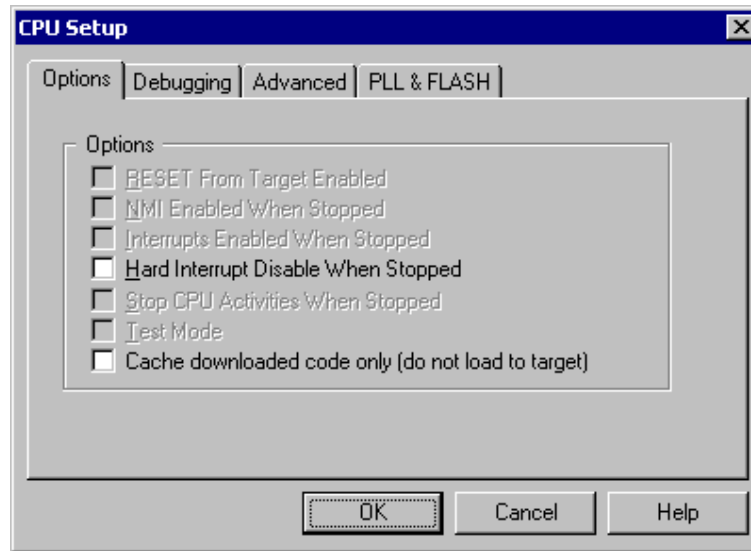
Note: Burst and Burst+ modes are implemented for PowerPC, M-CORE and ARM CPUs, including XScale.

Use – Scan Speed during Initialization

On some systems, slower scan speed must be used during initialization, during which the CPU clock is raised (PPL engaged) and then higher scan speeds can be used in operation. In such case, this option and the appropriate scan speed must be selected.

3 CPU Setup

3.1 General Options



MAC7100/MAC7200 Family Debugging Options

Hard Interrupt Disable When Stopped

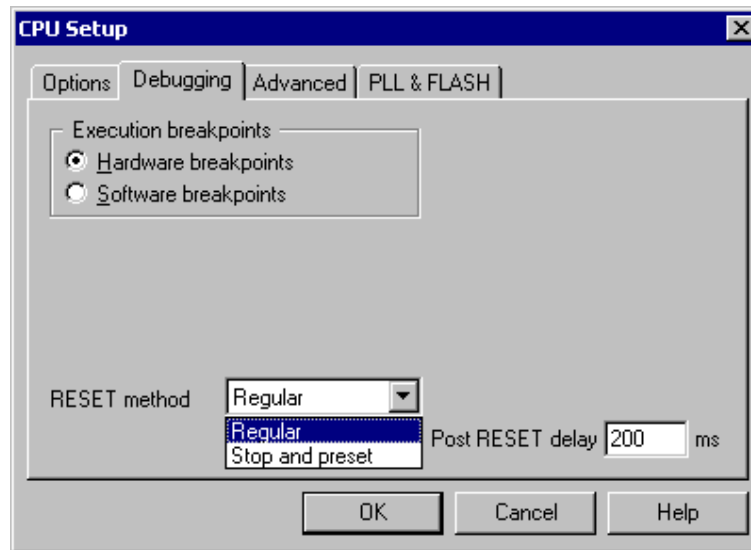
When this option is checked interrupts will be enabled immediately after program execution resumes. Otherwise, the CPU must execute a couple of instructions before returning to the program to determine whether interrupts were enabled when the CPU was stopped. These extra instruction executions can prevent task preemption when an interrupt is already pending.

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 ETM and Nexus 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 yield 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



MAC7100/MAC7200 Family Debugging Options

Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to two. 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

The ARM7TDMI core of the MAC7100/MAC7200 provides two hardware breakpoints, which often prove insufficient. The debugger can use unlimited software breakpoints to work around this limitation.

Note: ARM has no dedicated breakpoint instruction. Instead an invalid op-code is used and one hardware breakpoint is configured to trigger when this instruction is fetched. Thus only one hardware breakpoint remains available for hardware execution breakpoints, access breakpoints and trace trigger. If breakpoints are expected to be set only in areas, where software breakpoints cannot be used, it is advised to turn software breakpoints off, since this will enable the usage of the hardware breakpoint that is normally reserved.

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.

Besides setting software breakpoints in the RAM type memory, the debugger can set software breakpoints in the MAC7100/MAC7200 internal flash too. Refer to chapter 7 for more details on flash software breakpoints use.

Reset method

The option controls the resetting of the CPU after each download or reset command. Normally, the **'Regular'** RESET method is used. On some targets, the program cannot be stopped immediately after reset (targets where CPU reset and JTAG reset lines are not separated). In this case, the **'Stop and preset'** option is selected and the CPU (the program counter) is only preset to the default state.

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.

3.3 Advanced Options



MAC7100/MAC7200 Family Advanced options

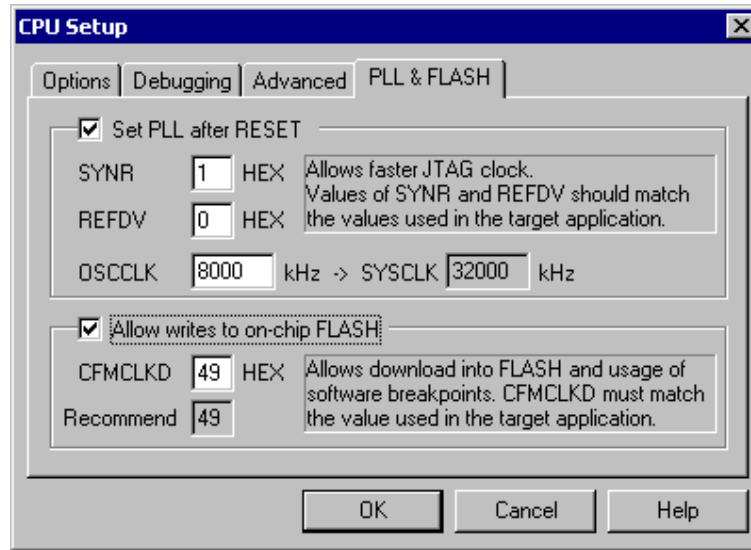
Override startup register values

This option overrides the default Program Counter reset value with the value set.

Use Handshaking

When this option is checked, execution of every command is handshaked. This is required by CPUs for which the JTAG clock is too high. This option should be turned off for MAC7100/MAC7200.

3.4 PLL and FLASH Options



PLL & FLASH configuration

Set PLL after RESET

This option allows configuration of PLL, which allows faster JTAG clock. The values must match the values used in the target application.

Allow writes to On-Chip FLASH

With this option checked, the writes to the On-Chip FLASH and the usage of software breakpoints in the internal FLASH are enabled. The value of CFMCLKD must match the value used in the target application.

For more information on setting PLL and Flash options, please see the chapters Enhancing Debug Performance and Internal FLASH Programming.

4 Hot Attach

The Hot Attach function can be enabled for troubleshooting purposes. The full hot attach is not supported, which means that the target must be first connected to the emulator and then turned on. The target must be functional, which means that it must contain a FLASH with a working code.

First, make sure the Hot Attach to Target function in the Hardware/Emulation Options/Hardware dialog is turned on (checked).

Typical usage, with the emulator turned on:

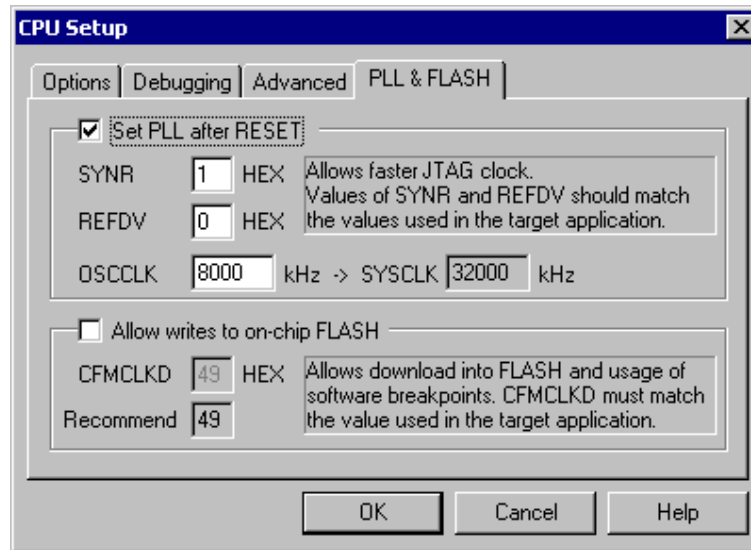
- turn the init sequence off
- erase the download file
- turn on the target
- invoke Reset
- now the emulator must be able to read the status – typically Running or Reset will be shown in the status. If the target is running, it can now be stopped with the Stop command. If the stop has succeeded, the debugging is operational (including breakpoints, stepping, etc.).

With this function, the debugger does not set breakpoints to Reset or in any other way manipulates with the internal debug logic. This function is limited to polling the status of the internal CPU logic (Embedded ICE). This is especially useful for initializing and troubleshooting, on the other hand for normal debugging this option should be turned off.

5 Enhancing Debug Performance

MAC7100/MAC7200 devices boot with a low system clock which, prevents usage of high-speed JTAG clocks, thus yielding poor performance of the debugger. Since the application will probably boost the system clock anyway, the debugger can do so at boot time to allow faster program download.

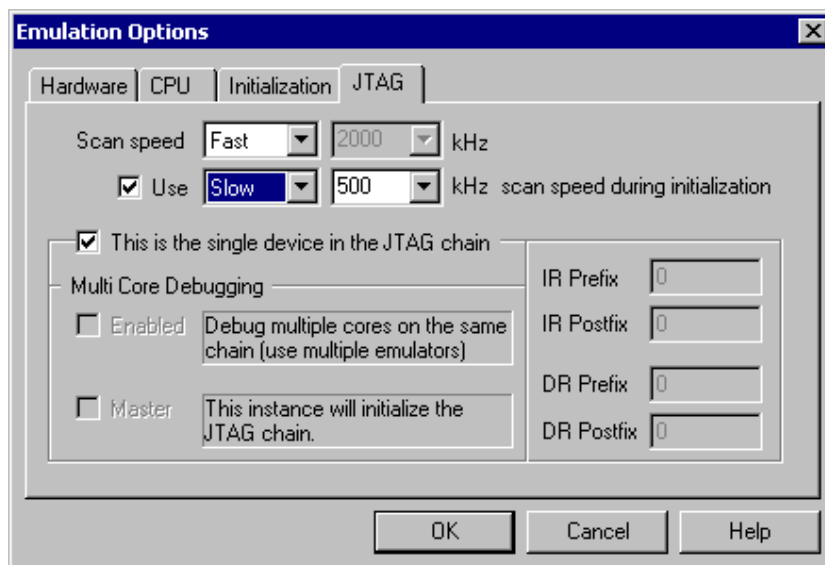
It is recommended to configure the PLL in the same manner that the target application will use. The *Hardware/Emulation Options/CPU/Setup/PLL & FLASH* dialog provides options to configure the PLL.



PLL & FLASH configuration

Use OSCCLK setting to specify the oscillator clock and then specify PLL settings (SYNR, REFDV).

When the debugger is configured to boost the system clock, the JTAG scan speed settings should be adjusted too to achieve the maximum debug performance (e.g. download speed). The *Hardware/Emulation Options/JTAG* dialog provides options to configure the JTAG scan speed.



MAC7100/MAC7200 JTAG speed setup

If the PLL has been configured to boost the system clock, 'Fast' setting can be used. Make sure the 'Use Slow scan speed during initialization' is used and a low frequency is selected. This setting forces slower clock until the initialization of the PLL is complete. If your oscillator is fast enough, you can try rising frequency of Slow scan speed – that will speed up the initialization somewhat.

With the above settings winIDEA will perform the following actions after reset:

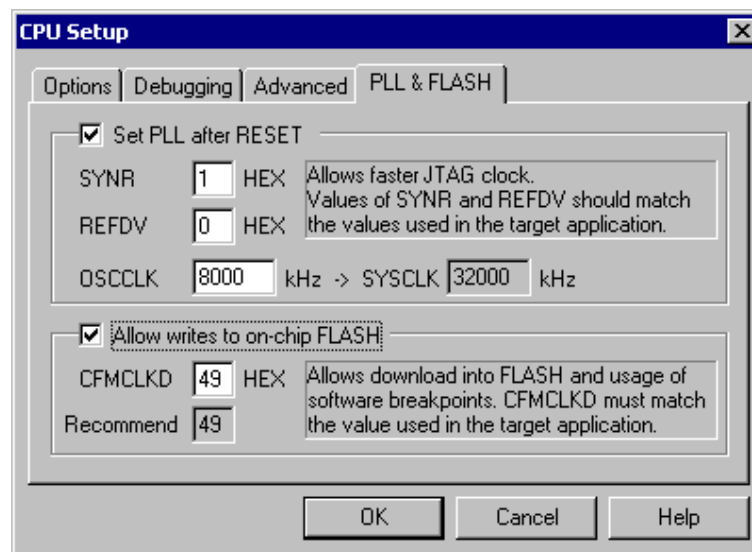
- Configure SYNCR and REFDV registers using Slow JTAG scan speed
- Enable the PLL
- Start using Fast JTAG scan speed

6 Internal FLASH Programming

MAC7100/MAC7200 devices have an on-chip FLASH module called Common FLASH module or CFM. Accessing the CFM poses certain requirements on the debugger as well as on the target application. Most notably the CFM Clock divider register (CFMCLKD) can only be written once, it must however be properly configured before any CFM operation is performed.

The user must specify the value of the CFMCLKD to allow the debugger to access FLASH functions without rendering it unusable for the target application.

The *Hardware/Emulation Options/CPU/Setup/PLL & FLASH* dialog provides options to configure the CFMCLKD.



When PLL is used to boost the debug performances, a recommended value for the CFMCLKD will be calculated - user should verify that this value is also used in his application.

No additional initialization is required when downloading the code to the internal FLASH only. The *Hardware/Emulation Options/Initialization* dialog can be blank that is 'None' selected.

The code can now be downloaded to the internal CPU FLASH through standard debug download. There is no need to invoke and configure standard 'FLASH Programming Setup' dialog. Additionally, unlimited software breakpoints can be used in the FLASH and the FLASH modified through the memory window.

7 Internal FLASH Software Breakpoints

Note: To use software breakpoints in the MAC7100/MAC7200 internal FLASH the *Hardware/Emulation Options/CPU/Setup/PLL & FLASH* options must be properly configured to allow write access to FLASH.

A software breakpoint is configured by programming a breakpoint instruction in the FLASH at the specified address. Note that:

- The debugger will remove all software breakpoints when the debug session ends. However, if target is force disconnected, software breakpoints will remain in FLASH memory and the target will not run properly standalone.
- When a software breakpoint is hit, the software must perform the following actions to resume execution:
 - Clear the breakpoint
 - Perform a single instruction step
 - Set the breakpoint again

For every such action the FLASH sector is erased and programmed twice. If the FLASH has a low reprogram rating, software breakpoint usage should be minimized. In worst case, the FLASH may become worn out due to intense and long lasting debugging using FLASH software breakpoints.

8 FLASH Security

The CFM provides several protection measures. It is strongly advised to study the *Common Flash Module* section in the *MAC7100/MAC7200 Users Manual*, particularly the *CFM Flash Configuration Field*.

IMPORTANT: if the CFM Flash Configuration Field (0x400 – 0x41B) is (over)written with user program, it is highly likely that the device will be secured AFTER the next RESET. To avoid this, make sure that your application contains deterministic values placed at those addresses. The bellow listing gives an example.

```
*** FlashSecurity.s ***
.section .FLASH, "a"
.code 32

    .long 0xffffffff /* Backdoor */
    .long 0xffffffff /* Backdoor */
    .long 0xffffffff /* Program FLASH protection Bytes */
    .long 0xffffffff /* Program FLASH SUPV Access Bytes */
    .long 0xffffffff /* Program FLASH DATA Access Bytes */
    .long 0x00000002 /* with this value the FLASH is not secured */
    .long 0xffffffff /* Data FLASH protection, SUPV access, DATA access */
.end
*** Linker command file ***
__INT_START__ = 0x00000000;
__FLASH_START__ = 0x00000400; /* the base of CFM config block */
__TEXT_START__ = 0x00001000;
__DATA_START__ = 0x40000000;
__STACK_END__ = 0x40008000;

SECTIONS
{
    /* Read-only sections, merged into text segment */
    .intvec __INT_START__ :
    {
        *(.intvec)
    }
}
```

```

}
.FLASH __FLASH_START__ :
{
  (.FLASH)      /* place values from FlashSecurity.s here */
}
.text __TEXT_START__ :
{
  *(.text)
}
. . .

```

If the CFM is accidentally secured, use the following procedure to unsecure the device:

- Switch off the power to the target and emulator.
- Force the MAC7100/MAC7200 to boot in single chip (set **MODA** and **MODB** pins to **0** – marked **ON** on the EVB DIP switch – see *Freescale MAC7100/MAC7200 EVB Configuration*)
- Turn on the emulator and the target
- Select Hardware/FLASH/Unsecure

The device should now be unsecured.

With some specific values a different kind of protection can occur (the debugger will fail to initialize without error message) – use these steps:

- Switch off the power to the target and emulator.
- Force the MAC7100/MAC7200 to boot in expanded mode (set **MODA** to **0** and **MODB** pin to **1**)
- Turn on the emulator and the target
- Make sure FLASH/Setup/Scope/Entire chip is selected
- Select FLASH/Program/Erase.

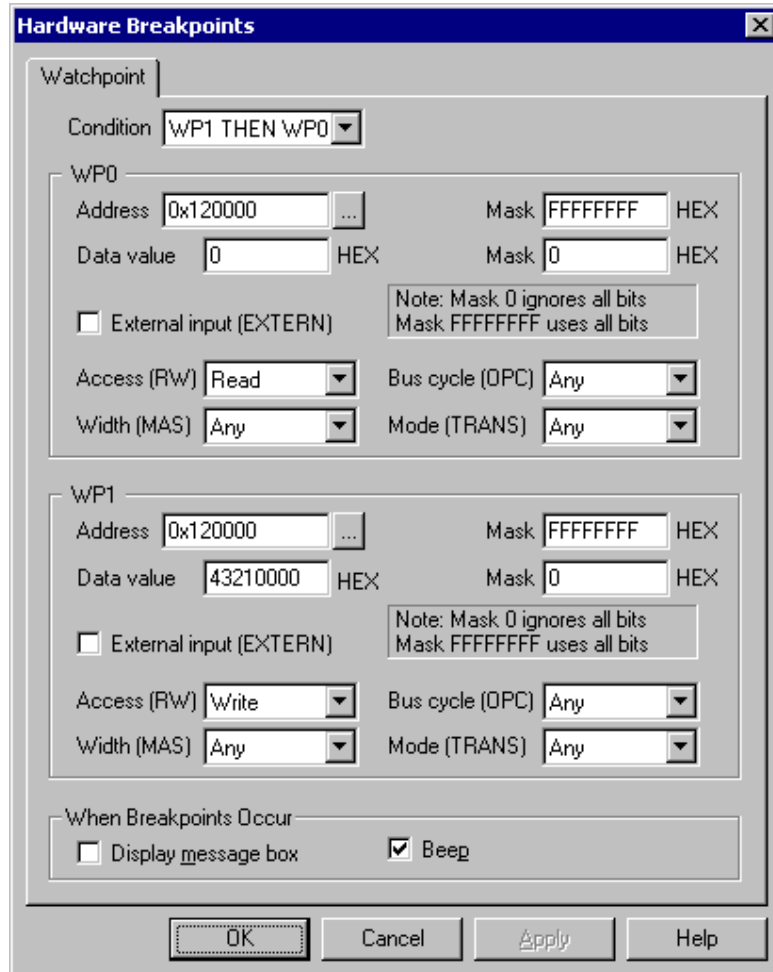
9 Real-Time Memory Access

MAC7100/MAC7200 provides real-time memory access capabilities. Watch window's *Rt. Watch* panes can be configured to inspect memory without stalling the CPU. Optionally, memory and SFR windows can be configured to use real-time access as well.

Please refer to the Software User's Guide for more information on Real-Time watches.

10 Access Breakpoints

For access breakpoints, on-chip breakpoint resources are being used. These on-chip resources are being shared between On-Chip Trace and Access Breakpoints. Bear in mind that the number of on-chip resources is limited and therefore an on-chip resource used for one purpose can not be used for the other.



ARM Hardware Breakpoints menu

Condition

Both internal ARM watchpoints can be used. Select the watchpoint combination that triggers the breakpoint.

Note: Refer to the ARM CPU manual for explanation of the RANGE mode.

WP0 and WP1

Specify the address and data bus states to monitor for the watchpoints.

Address

The address of the access breakpoint is entered here.

The mask can be also set. The mask 0 ignores all bits of the address and the mask FFFFFFFF uses all bits of the address.

Data Value

The data, which triggers the breakpoint, is entered here.

The mask can be also set. The mask 0 ignores all bits of the value; the mask FFFFFFFF uses all bits of the value.

Access, Bus Cycle, Access Width, Mode, External Input

The masks to be monitored.

When Breakpoints Occur

A beep can be issued and/or a message displayed indicating that an access breakpoint has occurred.

11 On-Chip Trace

MAC7100/MAC7200 features on-chip trace compliant with the Nexus standard. Nexus is a message based asynchronous protocol. In contrast to an In-Circuit Emulation where all buses are visible to the emulator and allow implementation of various analyzer features, a Nexus based tool is limited by the information provided through the Nexus port.

For tracking the sequence of program instructions, the Nexus port broadcasts only information related to instructions that cause a change to the normal sequential execution of instructions. With knowledge of the source code, which is programmed in the CPU flash, the debugger can reconstruct the path of execution through many instructions from the recorded change-of-flow information.

The MAC7100/MAC7200 provides Nexus 2+ level on-chip trace featuring program, data (only MAC7200) and OTM trace.

On-Chip Trace features (iTRACE PRO):

- Compliant with Nexus standard
- External trace buffer
- Instruction, Data (MAC7200 only) and OTM Trace
- Profiler
- Time Stamps
- AUX inputs

11.1 MAC7100 Trigger and Qualifier Configuration

For trigger and qualifier configuration, on-chip resources are being used. These on-chip resources are being shared between On-Chip Trace and Access Breakpoints. Bear in mind that the number of on-chip resources is limited and therefore an on-chip resource used for one purpose can not be used for the other.

The screenshot shows the 'Trace' dialog box with the 'AUX Card' tab selected. The 'Trigger' section is active. Under 'Condition', 'WP0' is selected. The 'Continuous mode' checkbox is unchecked. The 'WP0' section includes: Address 'iCounter', Mask 'FFFFFFF' (HEX), Data value '77' (HEX), Mask 'FF' (HEX), 'External input (EXTERN)' unchecked, Access (RW) 'Write', Bus cycle (DPC) 'Data access', Width (MAS) 'Any', and Mode (TRANS) 'Any'. A note states: 'Note: Mask 0 ignores all bits. Mask FFFFFFFF uses all bits'. The 'WP1' section includes: Address, Mask 'FFFFFFF' (HEX), Data value '0' (HEX), Mask '0' (HEX), 'External input (EXTERN)' unchecked, Access (RW) 'Any', Bus cycle (DPC) 'Any', Width (MAS) 'Any', and Mode (TRANS) 'Any'. The 'Record Messages' section has 'Program' and 'DTM on Address' checked, with the address field containing '#iCounter'. 'Time Stamp' is set to '250 ns', 'Buffer Size' is 'Minimum', and 'Trigger Position' is 'Center'. 'OK', 'Cancel', and 'Help' buttons are at the bottom.

MAC7100 Trigger configuration

Condition

Both internal ARM watchpoints can be used. Select the watchpoint combination that triggers the breakpoint.

Note: Refer to the ARM CPU manual for explanation of the RANGE mode.

WP0 and WP1

Specify the address and data bus states to monitor for the watchpoints.

Address

The address of the trigger condition is entered here.

The mask can be also set. The mask 0 ignores all bits of the address; the mask FFFFFFFF uses all bits of the address.

Data Value

The data, which triggers the event, is entered here.

The mask can be also set. The mask 0 ignores all bits of the value; the mask FFFFFFFF uses all bits of the value.

Access, Bus Cycle, Access Width, Mode, External Input

The masks to be monitored.

Record Messages

The ***Record Messages*** group configures the Nexus engine whether to generate **Program Trace** (these are required to reconstruct the program flow) and **Ownership Trace (OTM)** messages. The iTRACE PRO will capture all recorded messages.

Note: An OTM message is issued when the target application writes to a specific memory location. The data written is included in the OTM message. On MAC7100 this location is configurable and can thus be used as a limited data trace. In the bellow configuration all write accesses to **iCounter** will be recorded.

When trigger occurs and trace buffer fills, the recording is analyzed and displayed in the trace window.

Time Stamp

Determines the time stamp clock.

Note: Trace recording is synchronized with CPU cycles no matter what the time stamp setting is. The time stamp is a recording of an independent timer on the trace board.

Buffer Size

Specifies whether minimum or maximum trace buffer size is used. Usually the minimum size suffices. Maximum size uses the full trace buffer, which however takes a longer time to upload.

Note: Nexus is a message based protocol. When a certain message is received it is 'expanded' into one or more frames – typically only one message is received for every chunk of sequential code executed. Since timestamp is recorded on per-message basis, all frames derived from a certain message will have the same time.

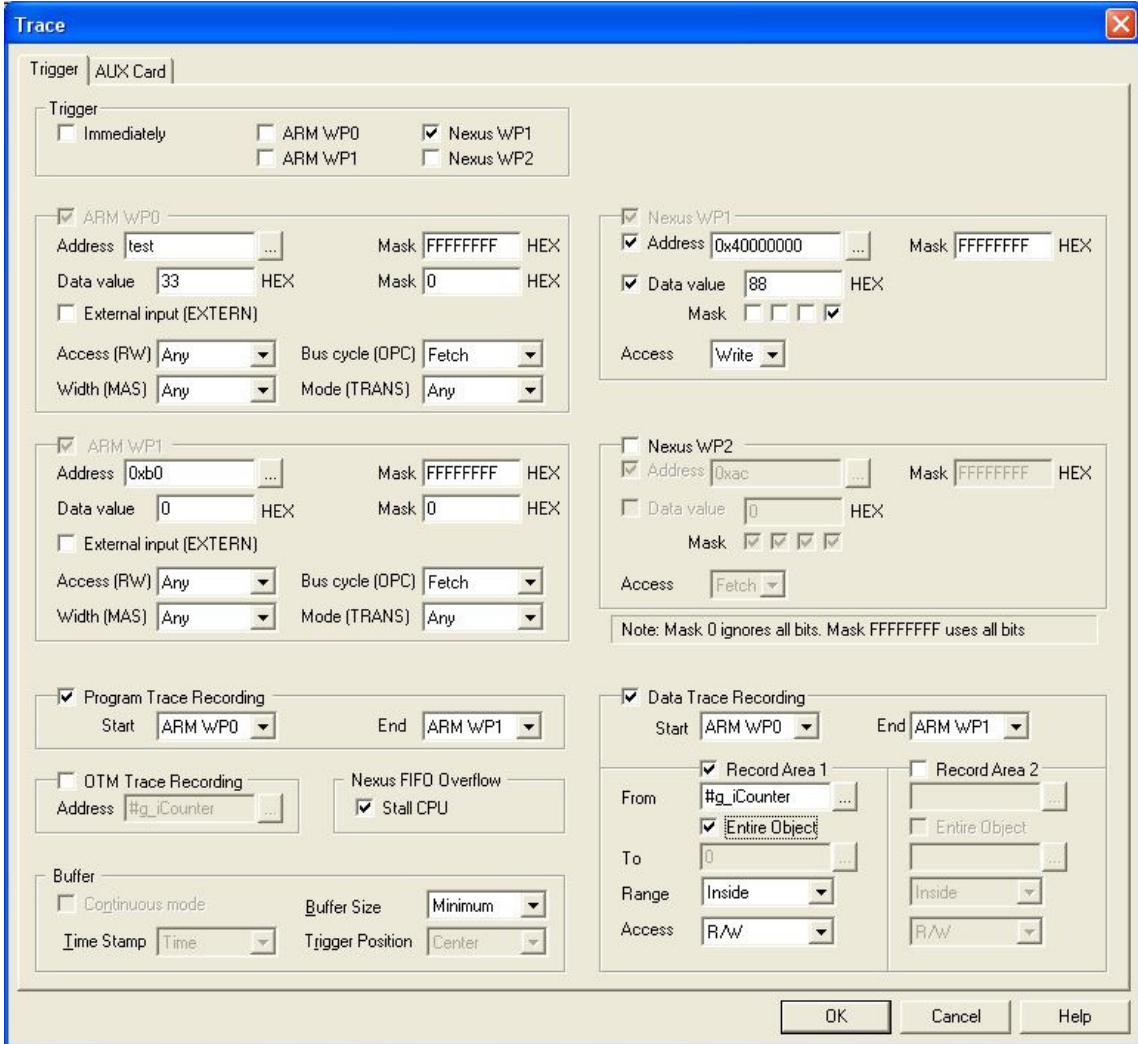
Trigger Position

Trigger position in the buffer. If the buffer size is set to minimum, this option is disabled, as well as when trigger condition is set to Anything.

- Begin - yields maximum post trigger size (7/8 after the trigger)
- Center - sets equal pre and post trigger sizes (1/2 after the trigger)
- End - yields maximum pre trigger size (1/8 after the trigger)

11.2 MAC7200 Trigger and Qualifier Configuration

For trigger and qualifier configuration, on-chip resources are being used. Some of the on-chip resources are being shared between On-Chip Trace and Access Breakpoints. Bear in mind that the number of on-chip resources is limited and therefore an on-chip resource used for one purpose can not be used for the other.



MAC7200 Trigger configuration

Trigger

All internal ARM watchpoints can be used. Select the watchpoint combination that triggers the breakpoint. If 'Immediately' is selected, the system triggers immediately and all other watchpoints are ignored.

ARM WP0 and WP1 and NEXUS WP0 and WP1

The ARM WP0 and WP1 are standard ARM7TDMI watchpoints. Note that they are shared with debugger - they can be used also for the hardware breakpoints. It is advised to use them after Nexus WP1 and Nexus WP2 watchpoints in order to keep the ARM WPs free for hardware execution and access breakpoints.

The NEXUS WP1 and WP2 are watchpoints used only for trace - as a trigger source, start or stop of the data or program trace.

Specify the address and data bus states to monitor for the watchpoints.

Address

If checked, the program or data address will be compared using Mask bits. Mask 0 ignores all bits, Mask FFFFFFFF uses all bits.

Data Value

If checked, the data value will be compared with corresponding byte Mask set.

Access, Bus Cycle, Access Width, Mode, External Input

Specifies the fetch for Program, Read, Write and R/W for data accesses.

Recording Messages

The ***Record Messages*** groups configure the Nexus engine whether to generate **Program Trace** (these are required to reconstruct the program flow), **Data Trace** and **Ownership Trace (OTM)** messages.

Note: An OTM message is issued when the target application writes to a specific memory location. The data written is included in the OTM message. On MAC7200 this location is configurable and can thus be used as a limited data trace. In the bellow configuration all write accesses to **iCounter** will be recorded.

When trigger occurs and trace buffer fills, the recording is analyzed and displayed in the trace window.

Program Trace Recording

If set, the OnChip program trace is switched on. The OnChip program trace can start immediately or after selected watchpoint, and the OnChip program trace can never end or after a selected watchpoint.

Data Trace Recording

If set, the OnChip data trace is switched on. The OnChip data trace can start immediately or after selected watchpoint, and the OnChip data trace can never end or after a selected watchpoint.

In the ***Data Trace Recording Record Area 1*** and ***2*** the filters for defining only specific data areas to be recorded are specified. The ***'From'*** field the Start address for data filtering is specified, if ***'EntireObject'*** is selected and if the "From" contains a data symbol with known size, then this check box set will automatically set the "To" address. The ***'To'*** field specifies the rnd address for data filtering.

The Data Trace Recording Range Can be inside or outside of the "From - To" area; the Data Trace Recording Access can be only Read, only Write of any of them.

OTM Trace Recording

If set, the OTM trace recording is switched on. The Address specifies the address of the CPU memory mapped register/variable which is broadcasted by OTM on every write to this address.

Nexus FIFO Overflow Stall CPU

With this option set, the Nexus FIFO overflows can be avoid by stalling the CPU. Note that this can have an impact to CPU performance.

Buffer Continuous mode

The Continuous mode can be set only if the trigger is set to Immediately. The trace will stop after the CPU stops or upon explicit trace stop command in the trace window. It can be used to trace the history before the CPU is stopped.

Time Stamp

Determines the time stamp clock.

Note: Trace recording is synchronized with CPU cycles no matter what the time stamp setting is. The time stamp is a recording of an independent timer on the trace board.

Buffer Size

Specifies whether minimum, medium or maximum trace buffer size is used. Usually the minimum size suffices. Maximum size uses the full trace buffer, which however takes a longer time to upload.

Note: Nexus is a message based protocol. When a certain message is received it is 'expanded' into one or more frames – typically only one message is received for every chunk of sequential code executed. Since timestamp is recorded on per-message basis, all frames derived from a certain message will have the same time.

Trigger Position

Trigger position in the buffer. If the buffer size is set to minimum, this option is disabled, as well as when trigger condition is set to Anything.

- Begin - yields maximum post trigger size (7/8 after the trigger)
- Center - sets equal pre and post trigger sizes (1/2 after the trigger)
- End - yields maximum pre trigger size (1/8 after the trigger)

11.3 Examples

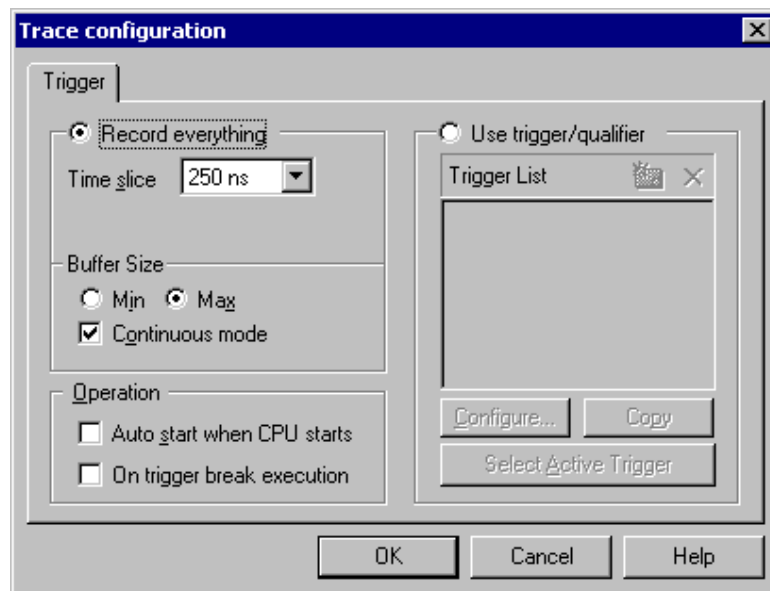
Example 1

Click the **Hardware Configuration** button in the Trace window toolbar.

Record everything is the default setting – once activated, the trace will immediately start recording all activity on the Nexus port.

Nexus will record the CPU execution (instructions) until the CPU is stopped by either the user or the program itself in case of any problems in the application. From the history, any problem originating from the code or the target can be filtered out.

Select the 'Record everything' mode in the 'Trigger List' dialog. Set buffer size to maximum to achieve the best results and check the 'Continuous mode' option.



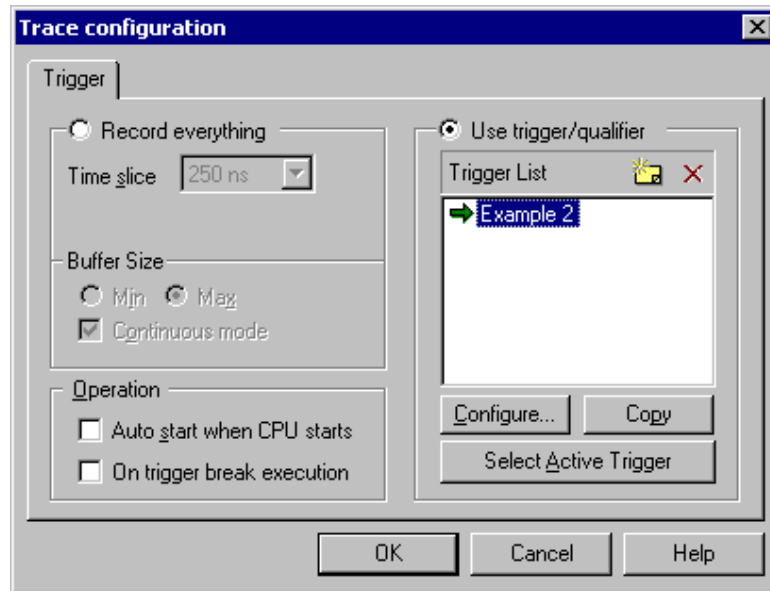
Before the program is set to run or while it is running already, activate trace recording via 'Trace begin' tool bar or shortcut key. The trace stops recording when the program execution is stopped. After the trace stops recording, the collected information is analyzed and displayed.

Example 2

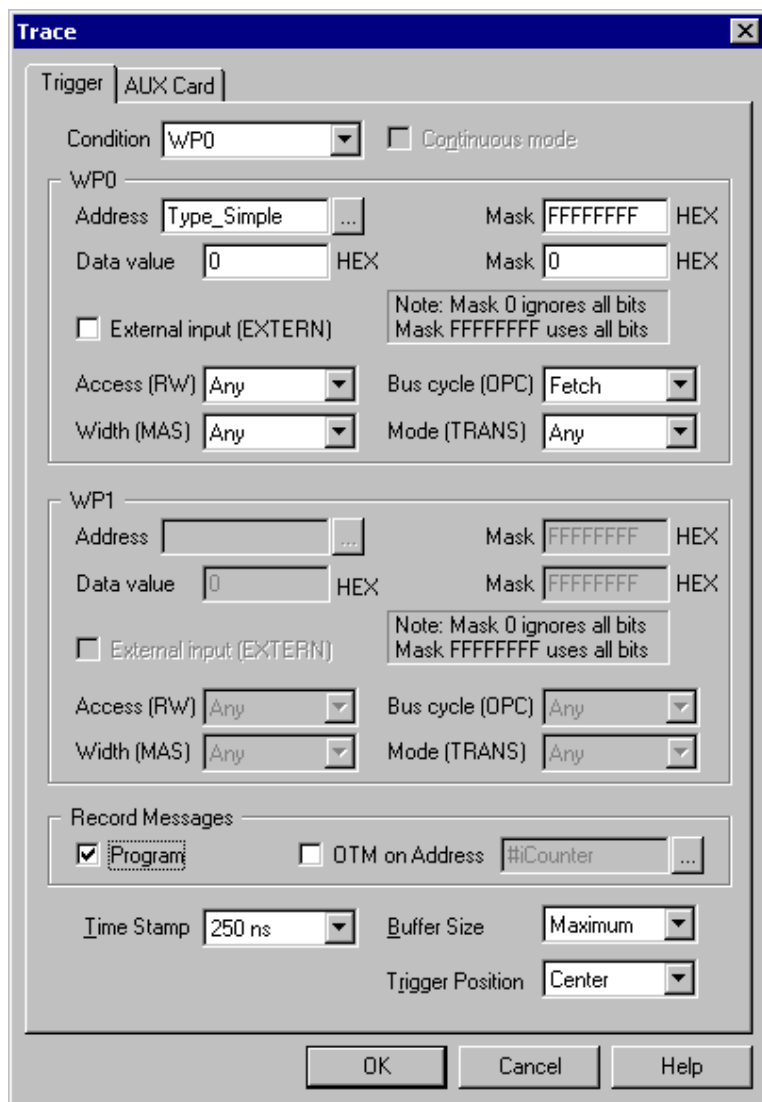
Nexus trace will trigger on a function `Type_Simple` execution and record instructions.

*Use **trigger/qualifier** allows finer configuration of the trace. Create a new trigger, then click the **Configure...** button.*

Select the 'Use trigger/qualifier' mode in the 'Trigger List' dialog and configure a new trigger called 'Example 2'.



Configure the trigger by invoking the 'Trace' dialog using the 'Configure' button.



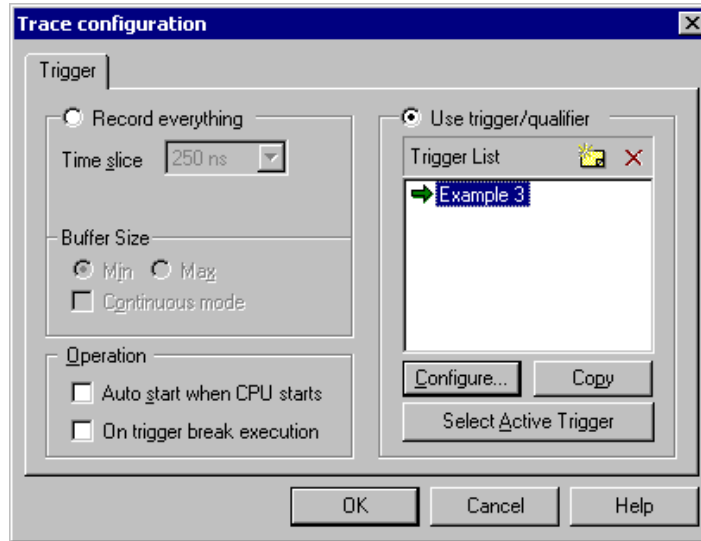
Trace Trigger Configuration

Define the condition WP0 and define the trigger to point to the `Type_Simple` address using the symbol browser invoked with the ‘...’ button. Define the Bus Cycle (OPC) to ‘Fetch’ and Access to ‘Any’.

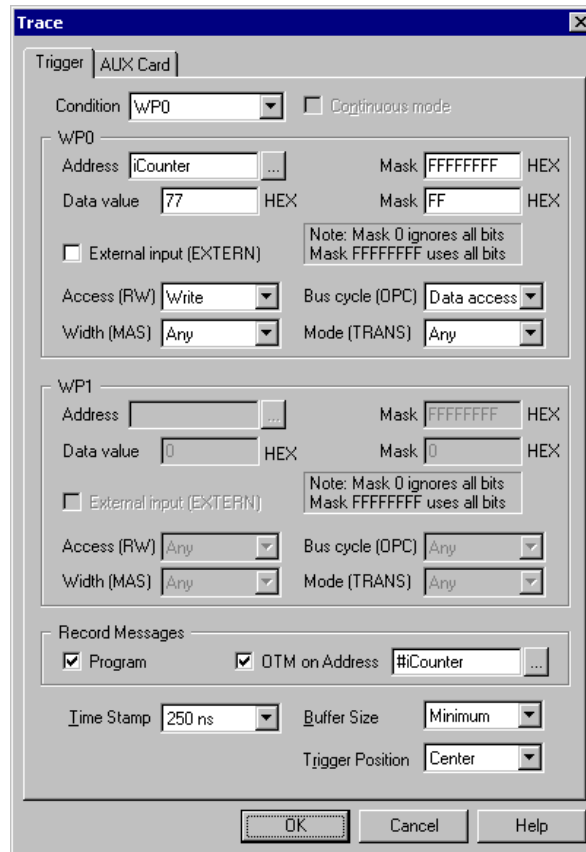
After the trace is being activated, Nexus starts recording after the `Type_Simple` is executed. After the iTRACE buffer is fulfilled, the results are displayed.

Example 3

Trace will trigger when `0x77` is written into the least significant byte of the address where variable `iCounter` is located and the instruction flow will be recorded.



Trace configuration



Trigger configuration

The **Record Messages** group configures the Nexus engine to generate both **Program Trace** (these are required to reconstruct the program flow) and **Ownership Trace (OTM)** messages. The iTRACE PRO will capture all recorded messages.

Note: An OTM message is issued when the target application writes to a specific memory location. The data written is included in the OTM message. On MAC7100 this location is configurable and can thus be used as a limited data trace. In the bellow configuration all write accesses to **iCounter** will be recorded.

Next,

- Close the trigger dialog
- Click the Begin button in the trace window toolbar
- Run the CPU

When trigger occurs and trace buffer fills, the recording is analyzed and displayed in the trace window.

Use the **Options/State View** from window's local menu to configure the display.

Note: Nexus is a message based protocol. When a certain message is received it is 'expanded' into one or more frames – typically only one message is received for every chunk of sequential code executed. Since timestamp is recorded on per-message basis, all frames derived from a certain message will have the same time.

Number	Data	Content	Time
28	0030A0E3	for (I = 0; ; I++)	2.75 us
	00001038 E3A03000	mov r3,#00	
29	10300BE5	0000103C E50B3010 str r3,[r11,0	2.75 us
30	34209FE5	iCounter++;	2.75 us
	00001040 E59F2034	ldr r2,[pc,00	
31	30309FE5	00001044 E59F3030 ldr r3,[pc,00	2.75 us
32	00000008	OTM	4.25 us
33	00000009	OTM	5.75 us
34	003093E5	00001048 E5933000 ldr r3,[r3]	8.00 us
35	013083E2	0000104C E2833001 add r3,r3,#01	8.00 us
36	003082E5	00001050 E5823000 str r3,[r2]	8.00 us
37	20309FE5	if (0x77 == (0xFF & iCounter))	8.00 us
	00001054 E59F3020	ldr r3,[pc,00	
38	003093E5	00001058 E5933000 ldr r3,[r3]	8.00 us
39	FF3003E2	0000105C E20330FF and r3,r3,#FF	8.00 us
40	770053E3	00001060 E3530077 cmps r3,#77	8.00 us

Trace window showing intermix of disassembly, source lines and OTM messages.

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) Make sure that slow JTAG scan speed is selected.
- 5) Execute debug reset
- 6) The CPU should stop on the reset location.
- 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 internal FLASH.
- 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

Make sure that the power supply is applied to the target BDM connector when 'Target VCC' is selected for Debug I/O levels in the Hardware/Emulator Options/Hardware tab, otherwise emulation fails or may behave unpredictably.

When performing any kind of checksum, remove all software breakpoints since they may impact the checksum result.

Try 'Slow' JTAG Scan speed if the debugger cannot connect to the CPU.

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.