Freescale MPC56xx / ST SPC56 Family In-Circuit Emulation

Contents

Contents ... 1
1 Introduction ... 2
2 Emulation Options ... 3
2.1 Hardware Options ... 3
2.2 Vcc/Clock ... 3
2.3 Initialization Sequence ... 4
3 CPU Setup ... 7
3.1 General Options ... 7
3.2 Debugging ... 8
3.3 Reset ... 10
3.4 Nexus ... 12
3.5 MPC5xxx Options ... 14
4 Real-Time Memory Access ... 16
5 Access Breakpoints ... 16
6 Trace, Profiler and Execution Coverage ... 18

1 Introduction

This document covers MPC560xB, MPC560xP, MPC564xB and MPC564xC debugging.

MPC560xB and MPC560xP devices are based on e200z0 core, which complies with the Power Architecture embedded category and only implements the VLE (variable-length encoding) APU, providing improved code density, significantly reducing memory requirements.

MPC564xB and MPC564xC devices are based on e200z4 core.

The Nexus Development Interface (NDI) block provides real-time development support capabilities for the microcontroller in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for microcontrollers without requiring external address and data pins for internal visibility.

The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface.

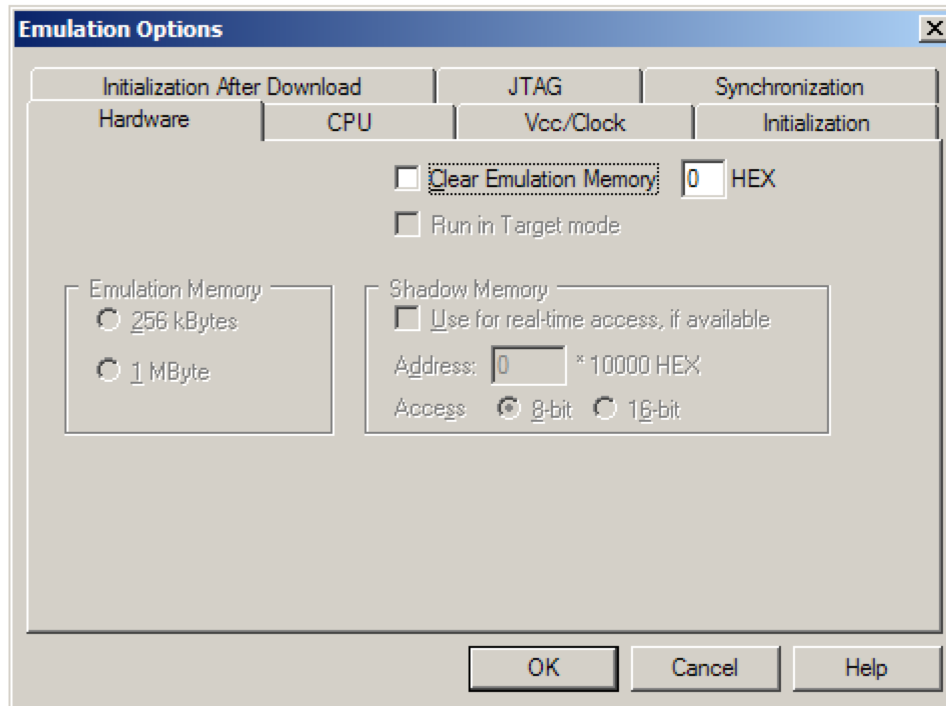
The NDI block interfaces to the e200 core, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace, watchpoint messaging, ownership trace, watchpoint triggering, processor overrun control, run-time access to the MCU's internal memory map, and access to the e200 internal registers during halt, via the JTAG port.

Features

- Four or eight hardware execution breakpoints
- Unlimited software breakpoints including in the internal CPU flash
- Access breakpoints
- Real-time memory access
- Nexus Trace
- Profiler
- Execution Coverage

2 Emulation Options

2.1 Hardware Options



In-Circuit Emulator Options dialog, Hardware page

Clear Emulation Memory

This option allows you to force clearing (with the specified value) of the emulation memory before debug download. If this option is used, it is recommended to clear the emulation memory with 0xFF value since it imitates an empty flash memory in the final target application.

2.2 Vcc/Clock

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

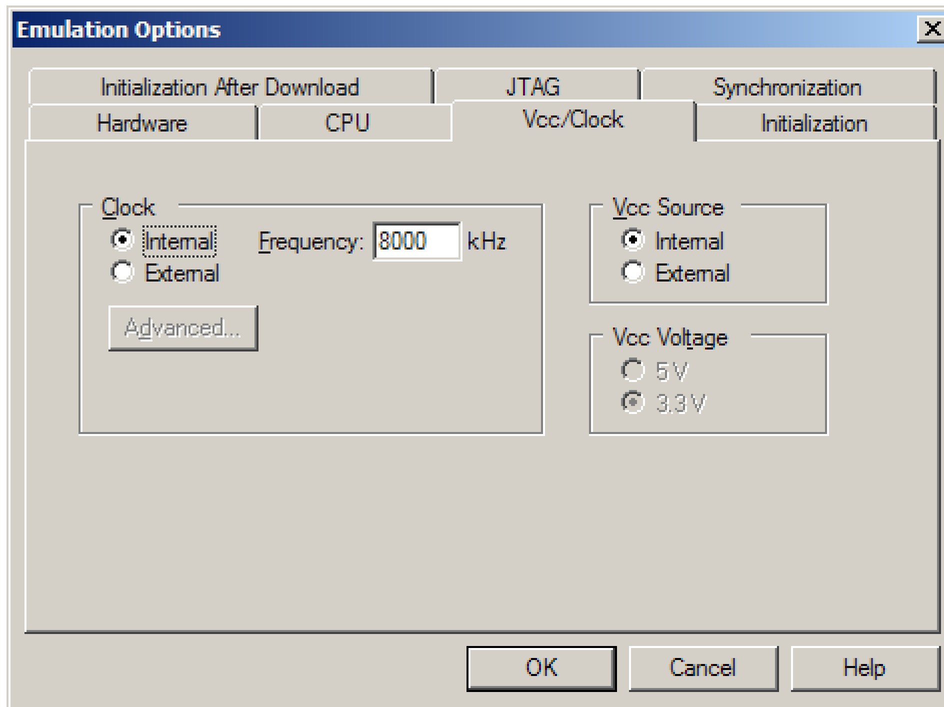
Note: When either of these settings is set to External, the corresponding line is routed directly to the CPU from the target system.

Clock

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

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

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



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

Vcc Sourcey

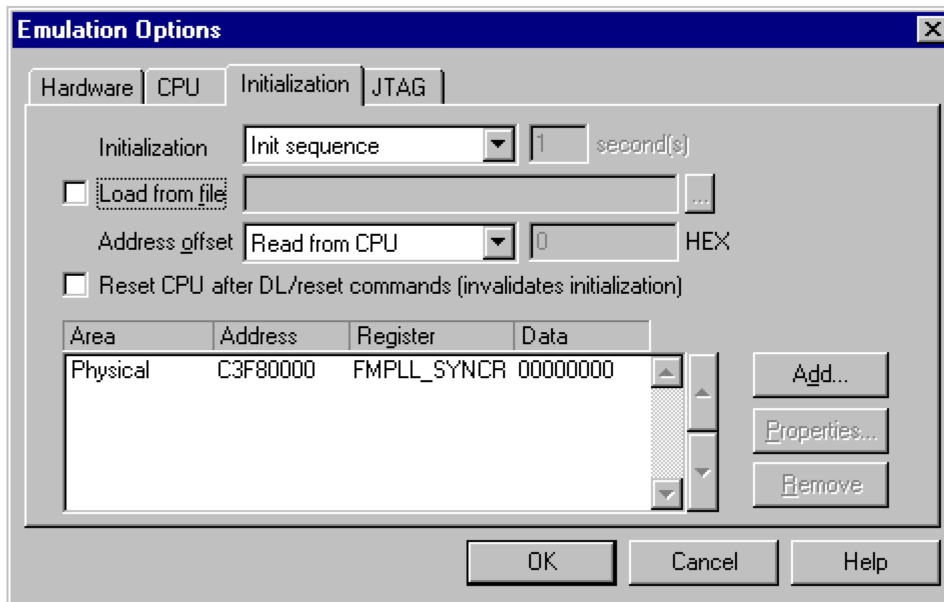
The POD can be powered either from the target or from the emulator ('Hardware/Emulation Options/Vcc/Clock' tab. From the emulator, it's always powered with 5V while from the target it can be powered either with 3.3V or 5V (depending on the target power supply)

2.3 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 written in 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. Few most often used cases:

- writing to SPR register
  A "SPR":(270) L 0x10000000
- writing to SFR
  S EBI_BR0 L 0x20000003
  S P2M W 0x0003
  S P3O_MC B 0x4F
- writing to a core register
  R PC L 0xF0000000
- writing to a memory location
  A (0x00000400) L 0x00000400
- pause 100ms
  P 100

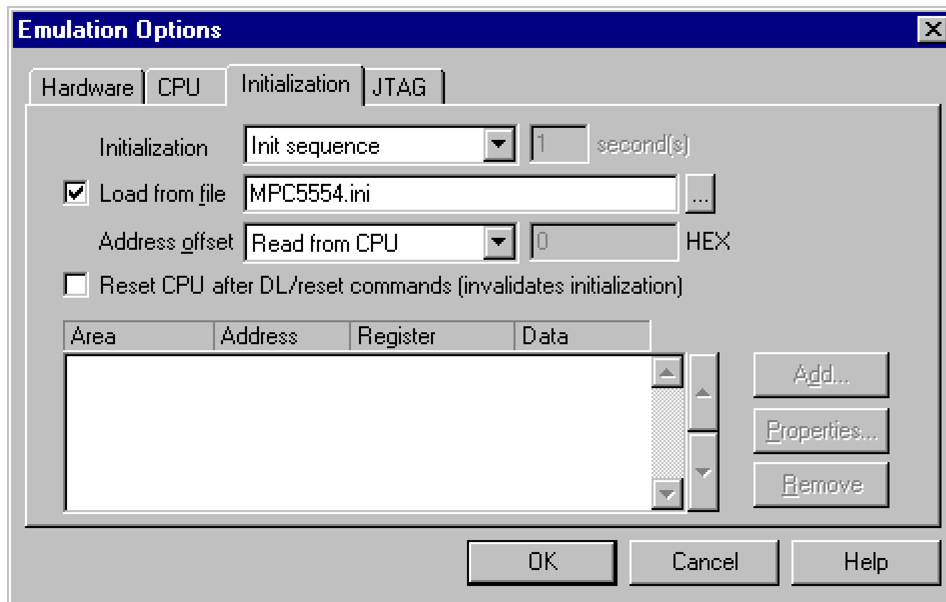
Excerpt from the example MPC5554.ini file

```
S FMPLL_SYNCR   L 0x00000000            // comment
```

Note: MPC5000 Memory Management Unit (MMU) is implementation with a 24-entry fully associative translation lookaside buffer (TLB). The TLB is accessed indirectly through several MMU assist (MAS) registers. The debugger can access MAS registers through SPR access and then write them to the TLB with a tlbwe (TLB write entry) instruction.

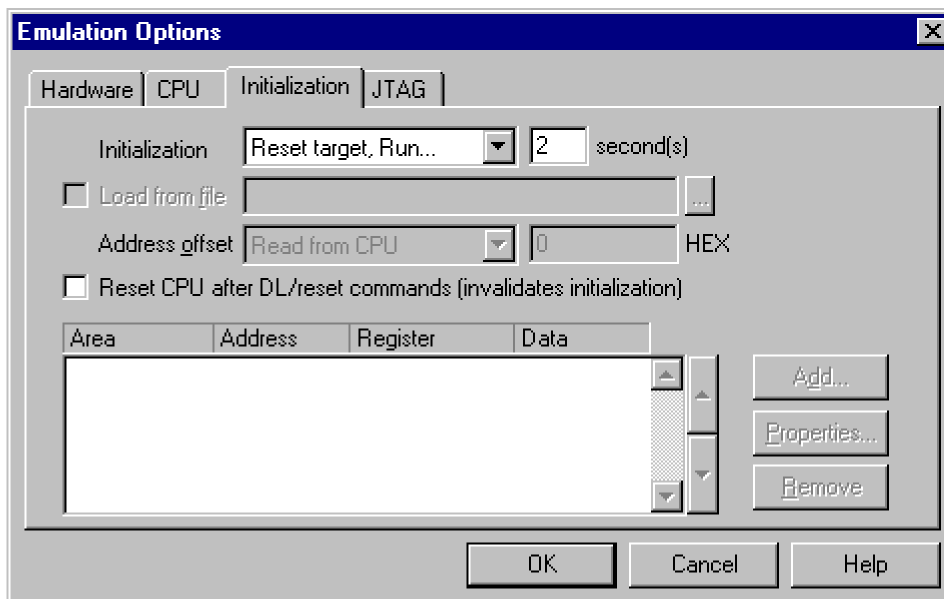
```
A "SPR":(270) L 0x10000000     // MAS0
A "SPR":(271) L 0xC0000500     // MAS1
A "SPR":(272) L 0xFFF0000A     // MAS2
A "SPR":(273) L 0xFFF0003F     // MAS3
I  7C0007A4                    // tlbwe
```

A new access method in the .ini file was introduced for MPC5500 family. Last line "I 7C0007A4" actually results in CPU executing instruction OpCode 7C0007A4.



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.

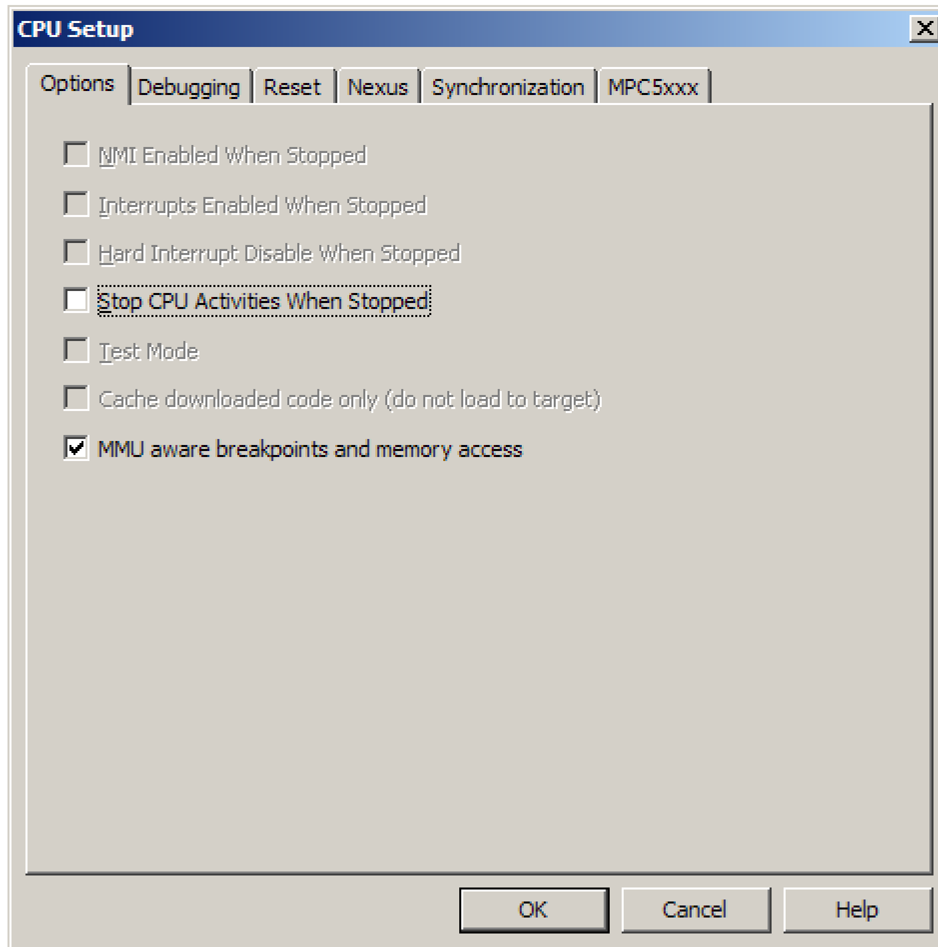
3. There is also a third method, which can be used too but it's not highly recommended for the start up. The user can initialize the CPU by executing part of the code in the target ROM for X seconds by using 'Reset and run for X sec' option.



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

Stop CPU Activities When Stopped

When this option is checked, peripheral functions like timers are stopped as soon as the application is stopped. However, this works fine only when hardware execution breakpoints are used for debugging. Per default, software execution breakpoints use the BGND instruction and for this instruction this option does not stop the timers. There is an alternative to use the TRAP instruction for software execution breakpoints (CPU Setup/MPC55xx tab) since it stops the timers too but the user must have in mind that the TRAP instruction can be used by the application too. In this case, a user TRAP instruction would stop the timers too. The BGND instruction is reserved exclusively for debugging.

In general, it is recommended that the option is checked in order to have more predictable behaviour of the application using the peripheral functions.

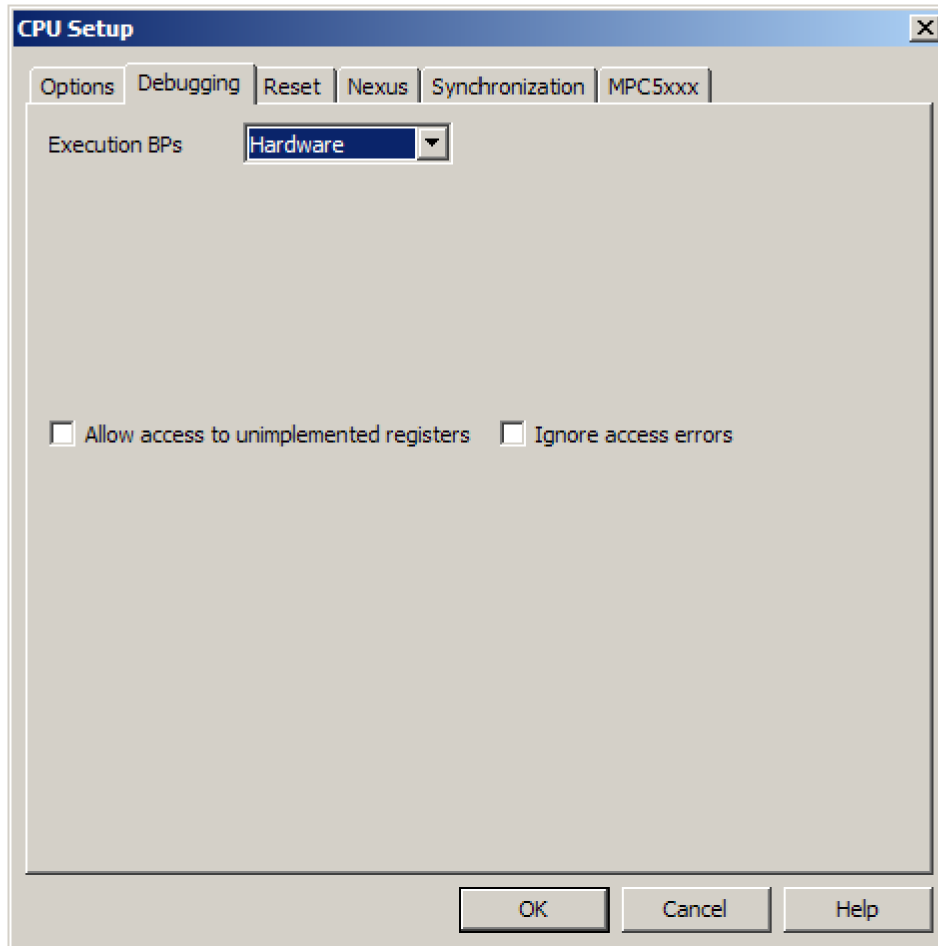
MMU aware breakpoints and memory access

If checked, breakpoints are set with physical addresses, memory accesses are performed using virtual/physical mapping.

If the option is cleared, no distinction is made between physical and virtual addresses.

Note: This option is not available for all microcontrollers.

3.2 Debugging



Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to four or eight, depending on the core. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writeable memory (ROM) or self-modifying code. If hardware breakpoints are 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.

The same on-chip debug resources are shared among hardware execution breakpoints, access breakpoints and trace trigger. Consequentially, debug resources used by one debug functionality are not available for the other two debug functionalities. In practice this would mean that no trace trigger can be set for instance on instruction address, when four execution breakpoints are set already.

Software Breakpoints

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation. The debugger also features unlimited software breakpoints in the MPC5xxx internal flash, which operate slowly comparing to hardware breakpoints due to the relatively large flash sectors.

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.

Allow access to unimplemented registers

When this option is checked, the debugger will allow access to the core registers (SPRs, PMRs, DCRs), which are not directly supported yet by winIDEA SFRs window.

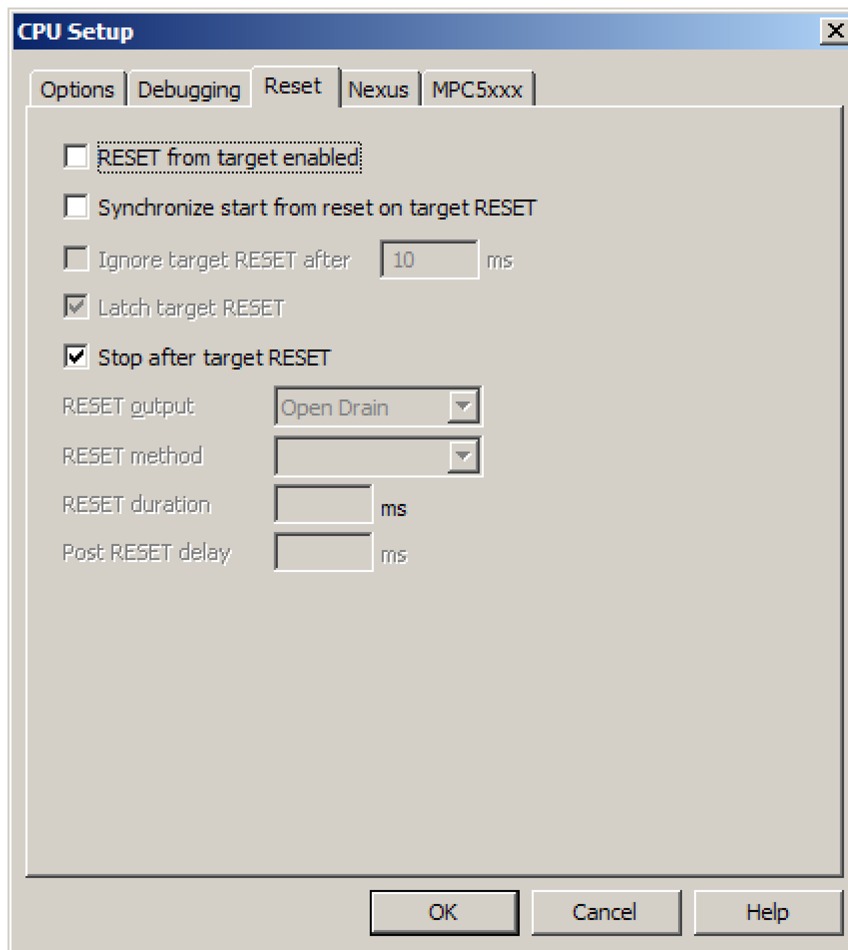
Possible use case would be if the customer finds a core register, which is not listed yet in the SFRs window. By checking this option and addressing this missing core register in the watch window, user gets immediate access to this register before winIDEA fix is provided.

Note that if you try to access unimplemented core registers, the CPU may hang. Therefore, use this option with caution.

Ignore Access errors

When checked, the debugger identifies memory access errors for individual memory location(s). When the option is unchecked, the debugger would declare access error for remaining memory locations once one access error is detected within a memory read block, which is used in the disassembly window or memory window.

3.3 Reset



RESET from Target Enabled

Beside the debugger, the target can have additional external reset sources, like power-on reset, watchdog circuitry or even reset push-button. In general, it's recommended to disable all external reset sources in the target, which may disturb the debugger in a way that debug communication is lost and complete system needs to be reinitialized.

It's recommended that all reset sources are designed as an open drain type. 'Reset from Target Enabled' option in the 'CPU Setup/Options' tab must be normally checked to assure safer debugging. Then the debugger can detect any reset source and service it properly.

Since target reset lines are designed as an open drain type, the debugger can detect all resets, even if they have been initiated by hardware other than the emulator itself. In certain applications, though, the requirement to disable this type of checking is required.

To disable reset sources from the target to be detected by the debugger, uncheck the 'RESET From Target Enabled' option. In this case, only the emulator will be able to generate a reset and the debugger will ignore all reset sources from the target.

Note: Wrong setting of this option can significantly change the operation of the target!

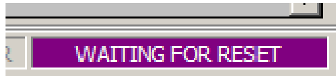
Synchronize start from reset on target RESET

This option is suppose to be used only with the targets featuring periodic target reset signal.

If option ‘Synchronize start from reset on target RESET’ is enabled then application run is synchronized with the periodical target reset. Synchronization is started either after the debug download or after the CPU reset. After the download (reset) winIDEA displays status MUST RUN.

In this mode target reset is disabled but all debug functions are available. Memory can be read, breakpoints set, trace started, instruction & source step performed, etc.

When the CPU is put into running (F5) status is changed to WAITING FOR RESET.



In this mode target reset is still disabled and the emulator is waiting for rising edge on the Target reset line.

When rising edge is detected, emulator puts the CPU into running and enables target reset.

Note: If target reset period is too fast, it’s possible that this status is not displayed at all because rising edge is detected before winIDEA displays that status.

Ignore target RESET after … ms

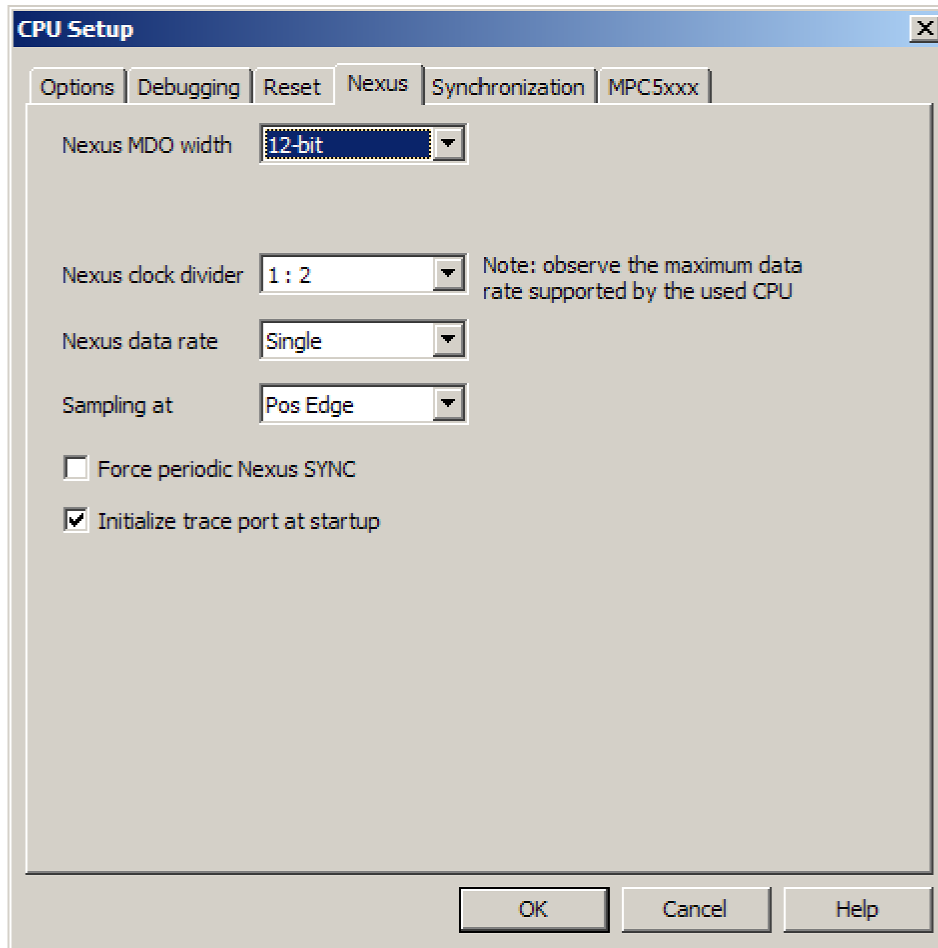
This option becomes enabled when RESET from Target Enabled’ and ‘Synchronize start from reset on target RESET’ options are both checked.

With this option, it’s possible to mask (ignore) target reset after some time. When the specified time expires, emulator ignores target reset. Using this option you can stop the CPU (by breakpoint for example) after the emulator disables target reset.

Stop after target RESET

CPU can be optionally stopped after the CPU reset is detected and handled. If the option is unchecked, the application is resumed upon reset release.

3.4 Nexus



Nexus MDO width

Nexus signals can be located on the alternate CPU pins, which can be configured for different alternate functions.

Microcontrollers supported by MPC564xB and MPC564xL Active PODs, allow configuring Nexus MDO port either as 4 or 12-bit port. A 12-bit MDO implementation ensures optimum Nexus operation. A 4-bit MDO implementation requires less CPU signals than the 12-bit MDO but the Nexus throughput is decreased, which is a crucial factor for correct trace operation. Note that the trace displays errors when the CPU doesn't manage to send out complete Nexus messages to the external development system. It's highly probably that 4-bit MDO port will result in trace errors while 12-bit MDO port will function flawlessly.

Microcontrollers supported by MPC560xB and MPC560xP Active PODs can have either 2-bit or 4-bit Nexus MDO port. The debugger displays available Nexus MDO width based on the selected CPU.

In general the user should opt for wider Nexus port since it provides maximum bandwidth of the Nexus interface. When the target implements only the lower Nexus MDO count of the two possible, the system will be more prone to Nexus overflows.

Nexus clock divider

This selection directly affects Nexus clock.

1:1 selection yields maximum Nexus clock. However, typically when the CPU clock goes over 100MHz, 1:2 selection must be used in order to reduce the Nexus clock frequency. Note that Nexus signals are typically

located on CPU I/O pins as an alternate operation. Output drivers of the I/O ports are typically designed for frequencies below 100MHz, which means they are not capable of driving Nexus signals at e.g. 128MHz. For this reason, Nexus clock is controlled over the Nexus clock divider in order not to exceed the maximum frequency of the physical Nexus ports.
For example, let's take a look MPC5643L device. As long as the CPU runs at 64MHz, 1:1 Nexus clock divider will work. When the same CPU runs at 128MHz, 1:2 Nexus clock divider must be used. However, this also halves the Nexus port bandwidth that is the amount of information which can be broadcasted over the Nexus port. Nexus overflows will occur when the nexus port bandwidth is exceeded.

Therefore, per default the divider should be 1:1 and it should be set to1:2 only when otherwise Nexus clock would exceed the maximum frequency of the Nexus port. Refer to the CPU Data Sheet for maximum port frequency of a particular device.

Nexus data rate

On some fast CPUs, it is possible to configure CPU to broadcast Nexus messages at double data rate, which yields increased Nexus port bandwidth. This means the Nexus trace will less likely overflow, which happens when the Nexus traffic is higher than the Nexus port bandwidth.

Sampling at

Per default, Nexus signals are sampled at positive edge of the Nexus clock. However, some targets may have delayed Nexus data signals comparing to the Nexus clock to such extent that it might be necessary to sample data on negative edge of the Nexus clock in order to capture valid Nexus information.

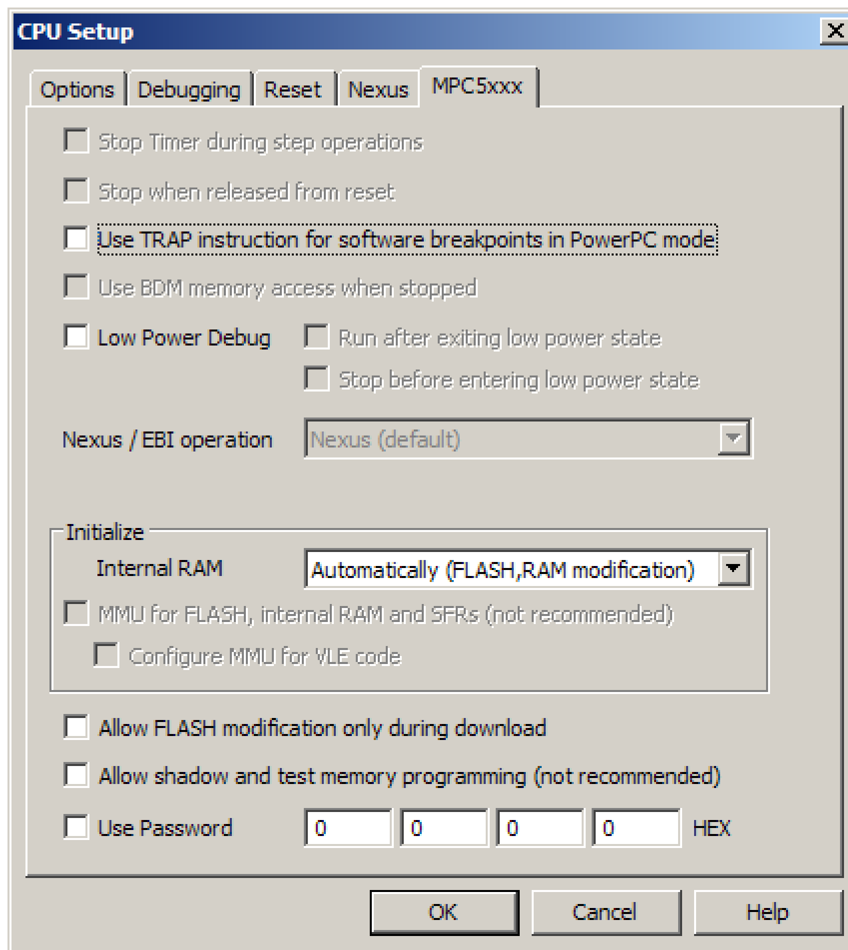
Force periodic Nexus SYNC

When there are problems with the trace or profiler (for instance no content is displayed), it is recommended to check this option for the test. If the application runs in some loop, which generates no SYNC nexus messages it may happen that the trace decoder cannot reconstruct any absolute CPU address from the recorded Nexus messages. When this option is checked, the debugger periodically inserts extra Nexus SYNC messages, which allow the trace and the profiler to reconstruct absolute CPU address for every such message.

Initialize trace port at startup

On some CPUs, certain CPU registers must be configured before Nexus trace can be used.

3.5 MPC5xxx Options



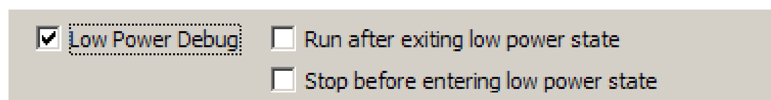
Use TRAP instruction for software breakpoints in PowerPC mode

Check the option when the limitation of the default software breakpoint instruction (BGND) is unacceptable for the target application being debugged. See more details on limitation at the 'Stop CPU Activities When Stopped' option description (CPU Setup/Options tab).

Low Power Debug

Note: This option is not available for microcontrollers (e.g. MPC5554), which don't feature according debug support.

Entry to low power mode and exit from low power mode is synchronized by the emulator when this option is checked.



When 'Run after exiting low power state' option is checked, the CPU is put into running after exiting the low power mode. Otherwise, default the CPU is stopped.

When 'Stop before entering low power state' option is checked, the debugger stops the CPU when the CPU receives a request to enter the low power mode. Under this condition the CPU will enter low power mode as soon as the user resumes the program (e.g. via Run debug command).

A “Low power mode exit” respectively ‘Enter to low power mode’ message pops up on low power mode entry/exit event when the ‘Display message box’ option is checked in the ‘Debug/Hardware Breakpoints/Action’ tab. Per default the option is unchecked.

Initialize

- *Internal RAM*

  Flash programming is implemented by loading small monitor into the internal CPU SRAM and executing its code hidden from the user. Since the target CPU SRAM features ECC control, it must be initialized before it can be used by the debugger.

  Per default, workspace is configured to initialize internal RAM always. This ensures that internal SRAM and flash are writable at any time during the debug session.

  When ‘Never’ is selected, the debugger does not initialize the CPU SRAM. This allows analyzing the CPU state after the CPU reset without RAM ECC initialization being performed by the debugger. Note that in this case SRAM and flash are not writable during the debug download or after the debug reset.

  A third selection ‘Automatically (FLASH, RAM modification)’ is also available. In this case, the initialization is performed at the moment when the user tries to modify SRAM or program flash (e.g. debug download, write in memory window, etc.).

  Note: Before the application can write to the RAM it must perform RAM ECC initialization in the startup code.

- *MMU for FLASH, internal RAM and SFRs (not recommended)*

  Under some circumstances the user may use ‘Go To’ address after the debug download or debug CPU reset. In such case, BAM code, which is otherwise executed by the CPU and configures MMU among other things, is skipped and the CPU program counter preset. Consequentially, program counter points to the address range for which the MMU is not configured and the debugger pops up an error “MMU TLB Entry for this address not found. Ensure correct MMU configuration«. To overcome this problem, check the Initialize 'MMU for FLASH, internal RAM and SFRs’ option in order for the debugger to configure the MMU after the CPU reset instead of the BAM code.

  Note that when ‘Go To’ address points to VLE code section, ‘Configure MMU for VLE code’ option should be checked too. Note that some devices support PowerPC instruction set only, some VLE instruction set only and some both instruction sets. Refer to the reference manual of your particular device for supported instruction set(s).

  Note: It is not recommended to use ‘Go To’ address after the debug download or the debug CPU reset unless the user is really aware of the consequences skipping the BAM and application startup code. It can easily happen that application being debugged with this option checked, will not run in standalone.

Allow flash modification only during download

When this option is checked, internal flash can be modified only through the debug download. When unchecked, it can be modified via memory window too.

Allow shadow and test memory programming (not recommended)

When this option is checked, shadow and test memory block will be programmed via debug download. Make sure you don’t overwrite by accident censorship or any other vital register which can lock the CPU forever.

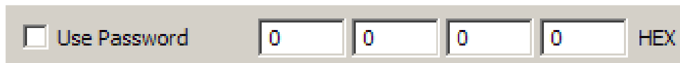
During the debug download and a memory write through the memory window complete flash block is read first then erased and after that modified data is written to flash (read/modify/write). There is no check where data is written to. Password for example (also invalid!) can be set in this way. Debugging is impossible if invalid password is written to the shadow block!

Test flash is write-once flash so it can't be erased.

'Hardware/Flash/Mass erase' command doesn't erase the shadow block. During the mass erase, shadow block is read first, then erased and after that a non user area of shadow block is written back. For instance, on MPC560x non user area is 0x200000-0x200007 and 0x203DD0-0x203FFF. 'Fill…' option from the local menu in the memory window can be used to fill a memory array with a certain value.

Use password

This option is available only for newer Bolero devices and allows protecting the CPU from unauthorized access. Check this option and enter four 16-bit values combining valid 64-bit password.



4 Real-Time Memory Access

On-chip debug module supports real-time memory access. 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.

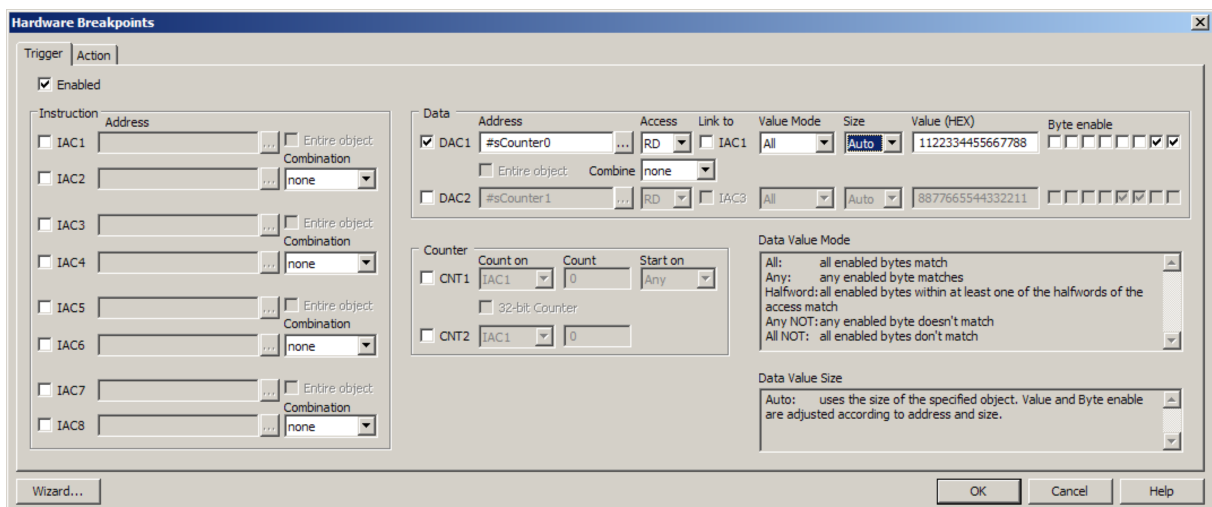
Please refer to the Software User's Guide for more information on Real-Time watches.

Note: Due to CPU issues, real-time access is not used while the Nexus trace port is active. In practice this means, whenever the trace, profiler or execution coverage is active, real-time access is not used.

In general, it's not recommended to use real-time access for Special Function Registers (SFRs) window. On most architectures real-time access still means stealing some microcontroller cycles. As long as the number of real-time access requests stays low, this is negligible and doesn't affect the application. However, if SFRs window or memory window is updated via real-time access, the application may start behaving differently due to stealing too many cycles. This is valid for this particular architecture too.

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 alternatively open but with SFRs collapsed). This allows observing a special function register in real-time with minimum intrusion on the application.

5 Access Breakpoints



MPC56xx Access Breakpoints

The same on-chip debug resources are shared among hardware execution breakpoints, access breakpoints and trace trigger. Consequentially, debug resources used by one debug functionality are not available for the other two debug functionalities. In practice this would mean that no trace trigger can be set for instance on instruction address, when four execution breakpoints are set already.

The debug interface includes four or eight Instruction Address Comparators (IAC1-4), which can be set to four/eight addresses or two/four ranges; and two Data Address Comparators (DAC1-2), which can be set to two addresses or one range. The ranges can be specified with the Inside, Outside or Mask combinations or the entire object can be used. Read or write accesses can be differentiated.

A data value can be set for each of the data address comparators (DAC1-2)

Refer to the Development Capabilities and Interface section of the respective CPU Reference Manual for more details on access breakpoints debug resources.

Wizard…

Since configuring access breakpoints may require deeper knowledge of the available on-chip debug resources, an easy to use Wizard (button in the left bottom corner) is available, which allows setting up few basic access breakpoints scenarios in few steps. Sometimes it’s also a good starting point for setting up more complex access breakpoints by first configuring basic access breakpoint using Wizard and then adjusting existing or configuring additional fields and options.

6 Trace, Profiler and Execution Coverage

Trace, Profiler and Execution Coverage are supported by the development system.

These functionalities are explained in the 'Freescale MPC5xxx & ST SPC56 Nexus Class 2+' and 'Freescale MPC5xxx & ST SPC56 Nexus Class 3+' technical notes documents. Use the according document depending on the Nexus Class type featured by the ActiveGT POD being used.

MPC560xB and MPc560xP ActiveGT PODs feature Nexus Class 2+ and MPC564xB and MPC564xC ActiveGT PODs feature Nexus Class 3+.

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.