
Technical Notes

Freescal MPC5xxx & ST SPC56 Family On-Chip Emulation

Contents

Contents.....	1
1 Introduction	2
2 Emulation Options.....	4
2.1 Hardware Options	4
2.2 Initialization Sequence	5
2.3 Initialization After Download Sequence	8
2.4 JTAG Scan Speed	8
3 CPU Setup	9
3.1 General Options	9
3.2 Debugging Options	11
3.3 Reset	13
3.4 Nexus	14
3.5 Synchronization	16
3.6 MPC5xxx Options	17
4 Flash	21
4.1 Programming	21
4.2 ECC Corruption	21
5 VLE	22
6 Hot Attach	24
7 eTPU Debugging.....	25
8 Debugging 2 nd Core	26
9 Real-Time Memory Access	27
10 MMU Support	27
11 Access Breakpoints	29
12 Trace, Profiler and Execution Coverage.....	30
13 Getting Started.....	30
14 Troubleshooting.....	31

1 Introduction

This document covers on-chip debugging for Freescale MPC5xxx and ST SPC56 microcontrollers while Nexus trace use is explained in separate documents. Nexus trace is explained separately for microcontrollers with Nexus Class 2+ and Nexus Class 3+ interface. Below table provides overview and some features of supported microcontrollers (valid July 2011). Contact iSYSTEM for the latest list of supported devices.

Freescale	ST	Nickname	Core	Max. clock (MHz)	Nexus Level	Nexus MDO size	Nexus double data rate
MPC5673K MPC5674K MPC5675K		Komodo	Z7+Z7	180	3+	12/16	Y
MPC5668E MPC5668G		Fado	Z6+Z0	116	3+(Z6), 2+(Z0) BGA256 emulation device	4/12	N
MPC5643L	SPC65EL44 SPC65EL50 SPC65EL60	Leopard	Z4 + Z4	120	3+	4/12	Y
MPC5644A MPC5646A	SPC564A80	Andorra	Z4	132	3+	4/12	Y
MPC5673F MPC5674F MPC5671L MPC5673L		Mamba	Z7	264 264 150 180	3+	12/16	N
MPC5602S MPC5604S MPC5606S	SPC560S50 SPC560S60	Spectrum	Z0	64	2+ MPC5606S in QFP176 & BGA208 emulation device	2/4	Y
MPC5632M MPC5633M MPC5634M	SPC563M54 SPC563M60 SPC563M64	Monaco	Z335	80	2+	4/12	Y
MPC5645S		Rainbow	Z4	125	3+	4/12	Y
MPC5566 MPC5567 MPC5568			Z6	132	3+	4/12	N
MPC5601B/D MPC5602B/C/D MPC5603B/C MPC5604B/C MPC5605B MPC5606B MPC5607B	SPC560B40 SPC560B44 SPC560B50	Bolero	Z0	64	2+ MPC5604B & MPC5607B BGA208 emulation device	2/4	Y
MPC5644B MPC5644C MPC5645B MPC5645C MPC5646B MPC5646C	SPC564Bxx SPC56ECxx	3M Bolero	Z4, some Z4+Z0	120	3+ MPC5646C in BGA256 only	8/12	Y
MPC5602P MPC5603P MPC5604P	SPC560P50 SPC560P40 SPC560P44 SPC560P 55	Pictus	Z0	64	2+	2/4	N
MPC5516			Z1+Z0	80	2+	4/8	N
MPC553x			Z3	80	3+	4/12	N
MPC555x			Z6	132	3+	4/12	N

Note: Nexus 2+ and 3+ feature Program and Ownership trace while only Nexus 3+ level feature Data trace.

Microcontrollers can feature the full 32-bit Power ISA instruction set as well as the ability to implement variable length encoding (VLE) instructions. For instance, Freescale MPC551x devices are designed to run the variable length encoding (VLE) instructions only, which delivers a high level of code density, significantly reducing memory requirements.

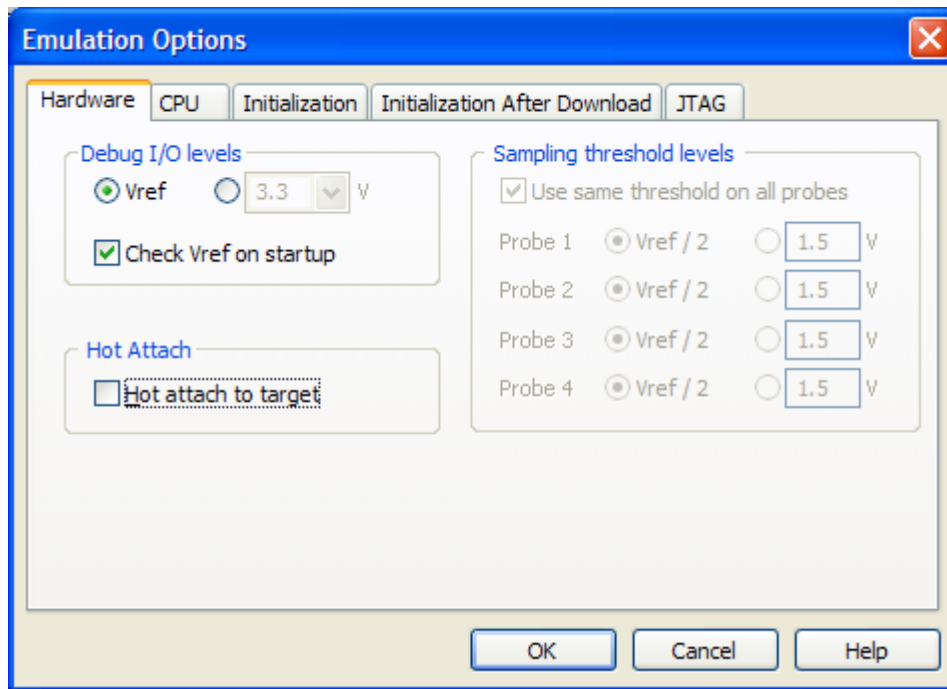
Basic debugging is based on on-chip debug control logic accessible through the JTAG debug interface while trace, profiler and execution coverage are implemented by using IEEE-ISTO 5001-2003 Nexus Class 2+ or 3+ interface.

Features

- Four or eight hardware execution breakpoints – depending on the core type
- Unlimited software breakpoints including in the internal CPU flash
- Access breakpoints
- Real-time memory access
- Flash programming
- Hot Attach
- MMU support
- eTPU debugging
- On-Chip Nexus Trace (e200, eTPU1, eTPU2, eDMA, FlexRay)
- Nexus RTR Trace (e200z1 core – full 32-bit Power ISA instruction set)
- Profiler
- Execution Coverage

2 Emulation Options

2.1 Hardware Options



Debug I/O levels

The development system can be configured in a way that the debug JTAG 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 JTAG signals. The user must ensure that the target power supply is connected to the Vref pin on the target JTAG 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.

Check Vref on startup

This option is available for iC5000 development system only. When checked, the system will check the presence of voltage on the Vref pin on the target debug connector. If no voltage or too low voltage is detected, a warning message is pop up.

Sampling threshold levels (iTRACE PRO/GT only)

Voltage levels of the debug input and output signals are adjusted depending on the setting.

Hot Attach

The debugger supports 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. See 'Hot Attach' chapter for more details on Hot Attach use.

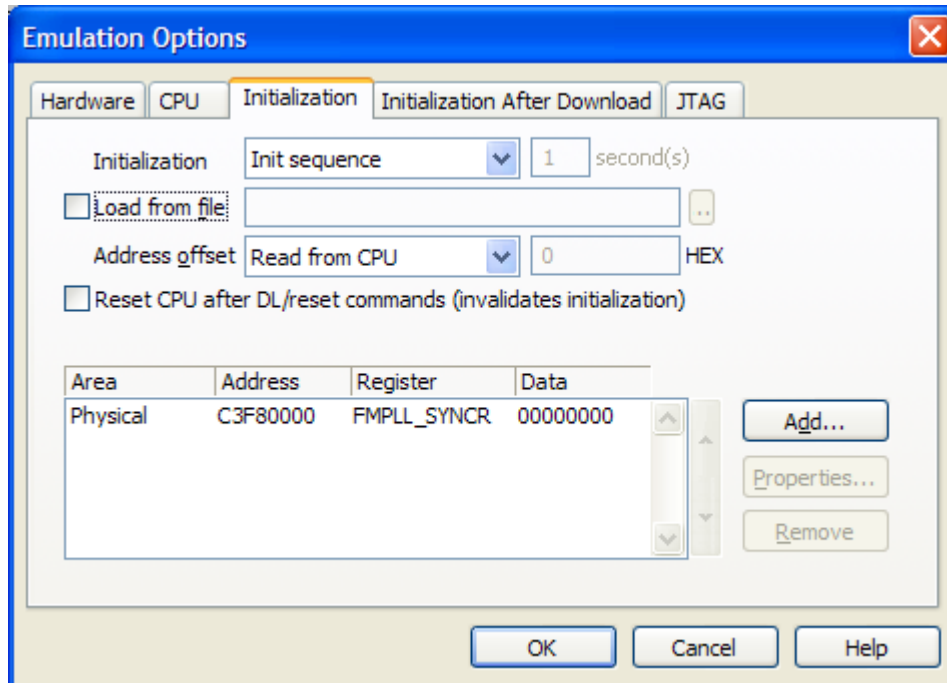
Note: Hot Attach function cannot be used for any flash programming or code download!

2.2 Initialization Sequence

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

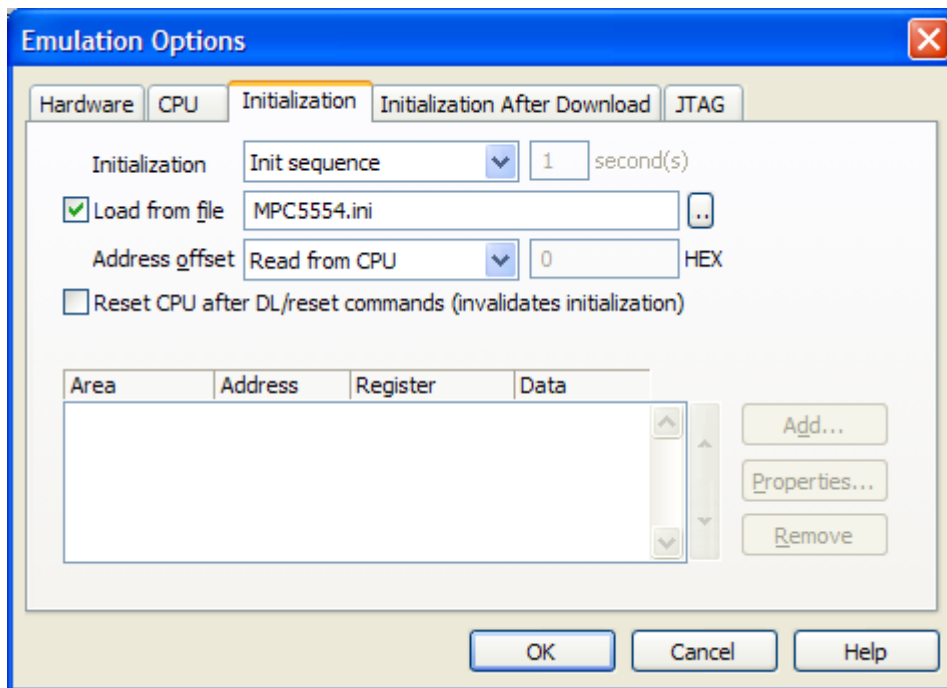
Excerpt from the example MPC5554.ini file
S FMPLL_SYNCR L 0x00000000 // comment

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 0xFFFF000A // MAS2
A "SPR":(273) L 0xFFFF003F // 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.

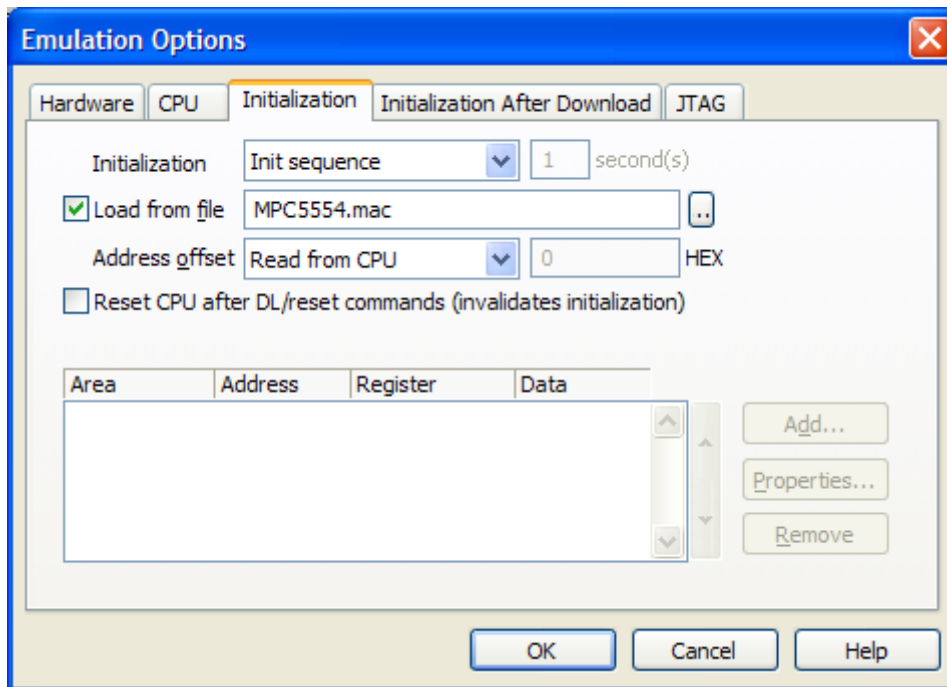


The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

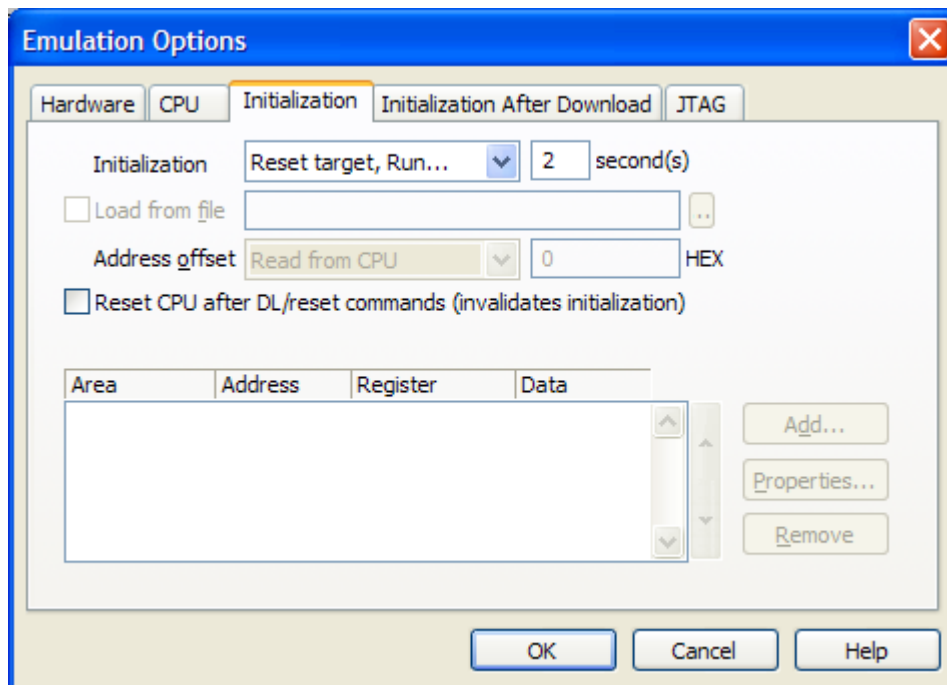
winIDEA also supports initialization files generated from Code Warrior IDE, which generates CPU initialization files depending on CPU and project settings. winIDEA recognizes the format by file extension: *.mac files. The file is specified in the 'Hardware/Emulation Options/Initialization' dialog (same as winIDEA *.ini files).

Note 1: to facilitate debugging any issues, winIDEA creates a converted winIDEA initialization (.ini) file next to the .mac file. The file is named **\$mac_file_name.ini**

Note 2: Currently only PowerPC architecture is supported.



- 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.

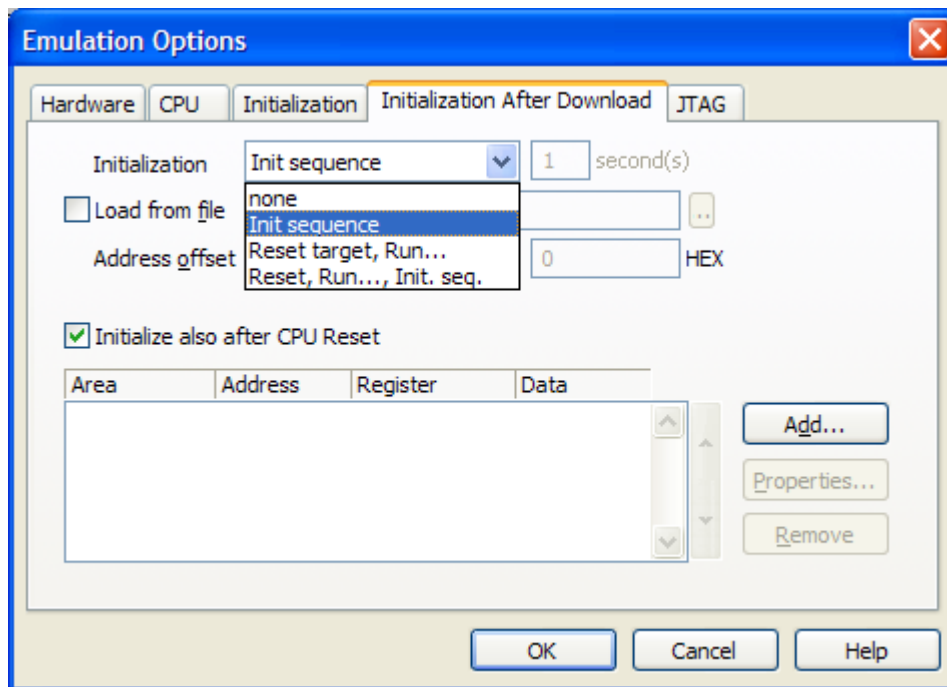


Reset CPU after DL/reset commands (invalidates initialization)

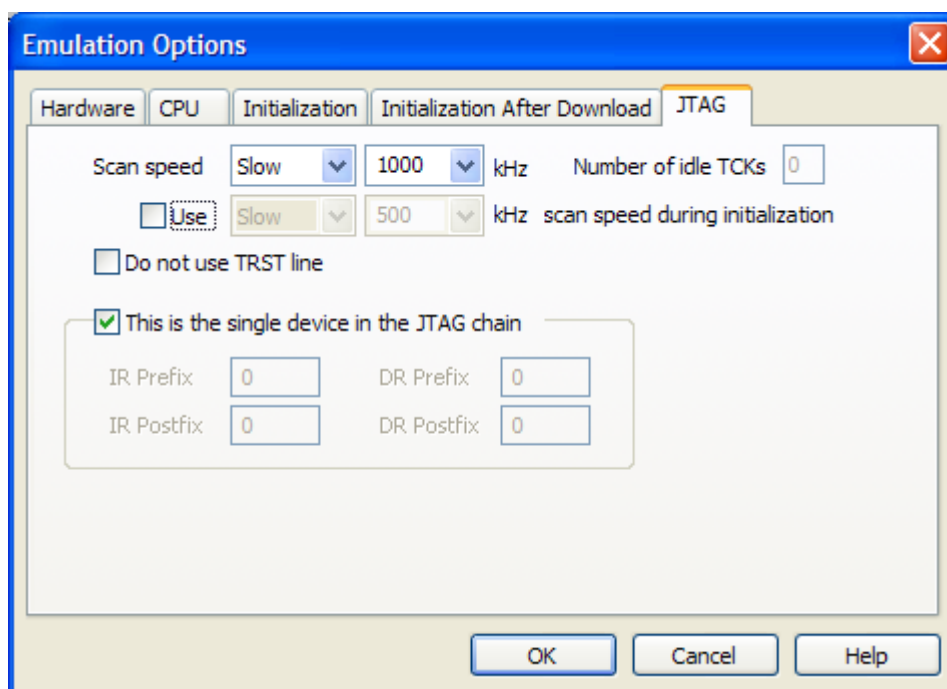
When the option is checked, the configured initialization, which is probably required for debug download (flash programming), is invalidated after the debug download and debug reset.

2.3 Initialization After Download Sequence

There are rare cases when a different initialization of the CPU is required for debug download and after the debug download. When Initialization After Download is used, typically the ‘Reset CPU after DL/reset commands (invalidates initialization)’ option described in the previous chapter is checked too.



2.4 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.
- Other scan speed types (not supported for MPC5xxx and SPC56 family) can be seen and are automatically forced to Slow.

Slow and Fast JTAG scanning is implemented by means of software toggling the necessary JTAG signals.

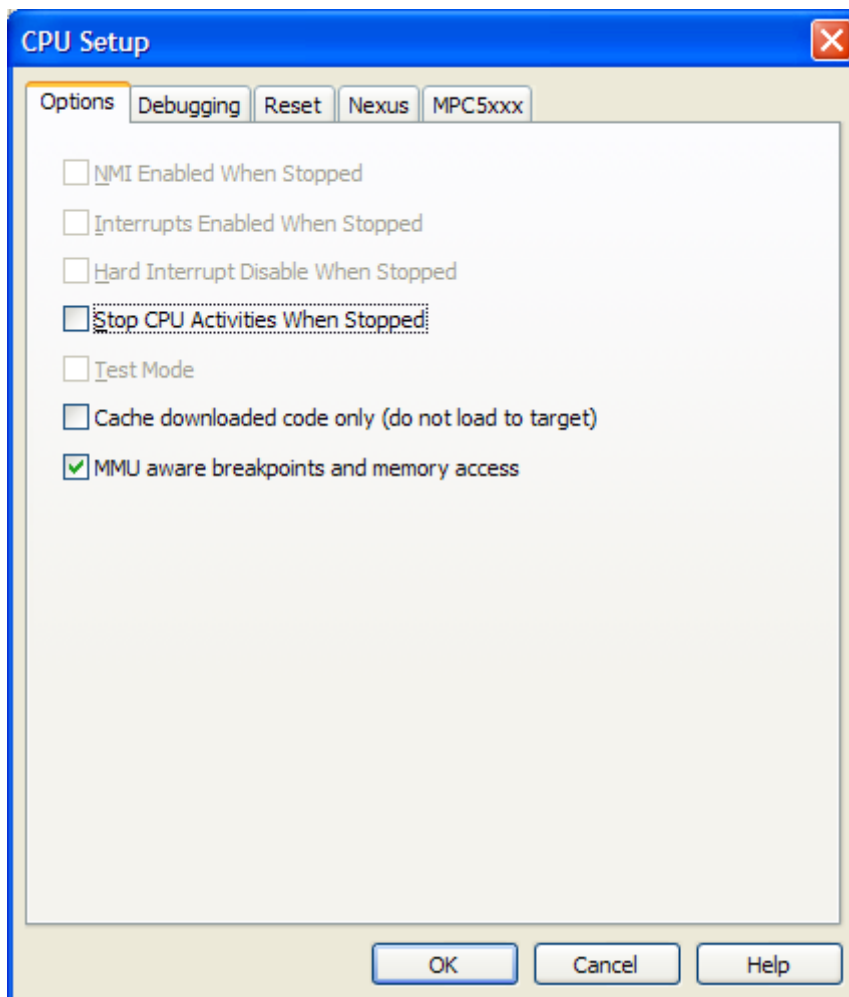
In general, Fast mode should be used as a default setting. If Fast mode fails or the debugging is unstable, try Slow mode at different scan frequencies until you find a working setting.

Use – Scan Speed during Initialization

On some systems, slower scan speed must be used during initialization, during which the CPU clock is raised (PLL 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



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.

Note: This option is not available for all microcontrollers.

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.

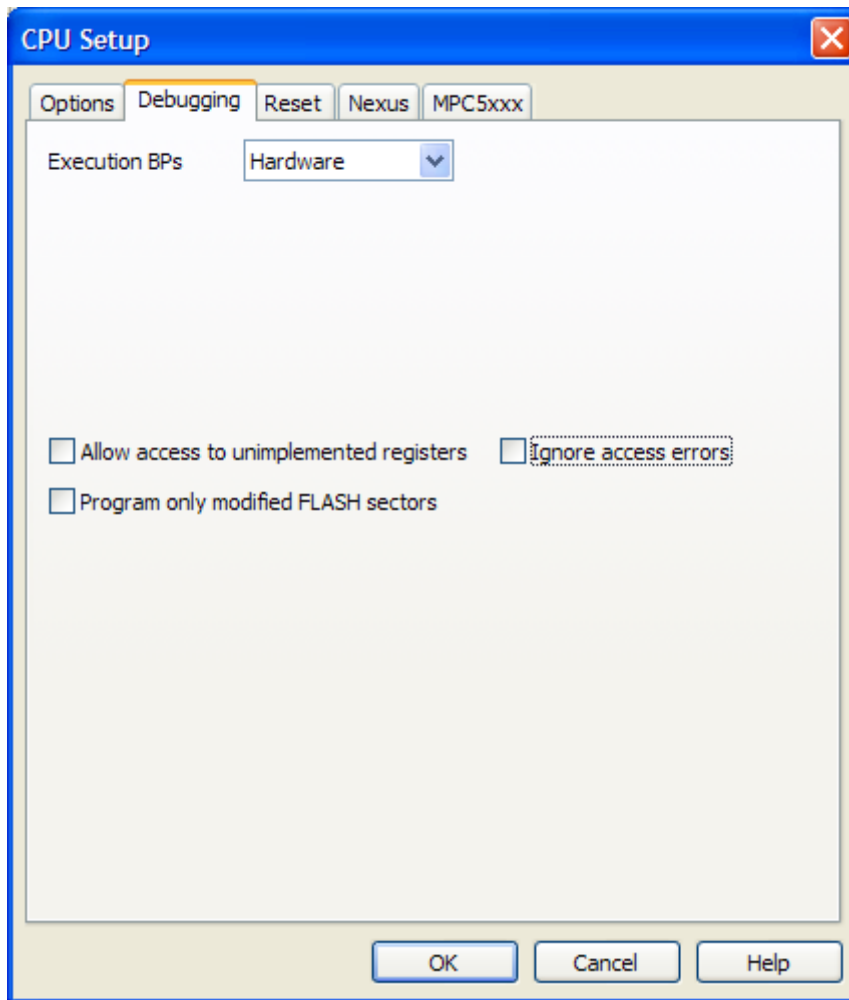
MMU aware breakpoints and memory access

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 available for MPC55xx devices only.

3.2 Debugging Options



PowerPC 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 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-writable 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

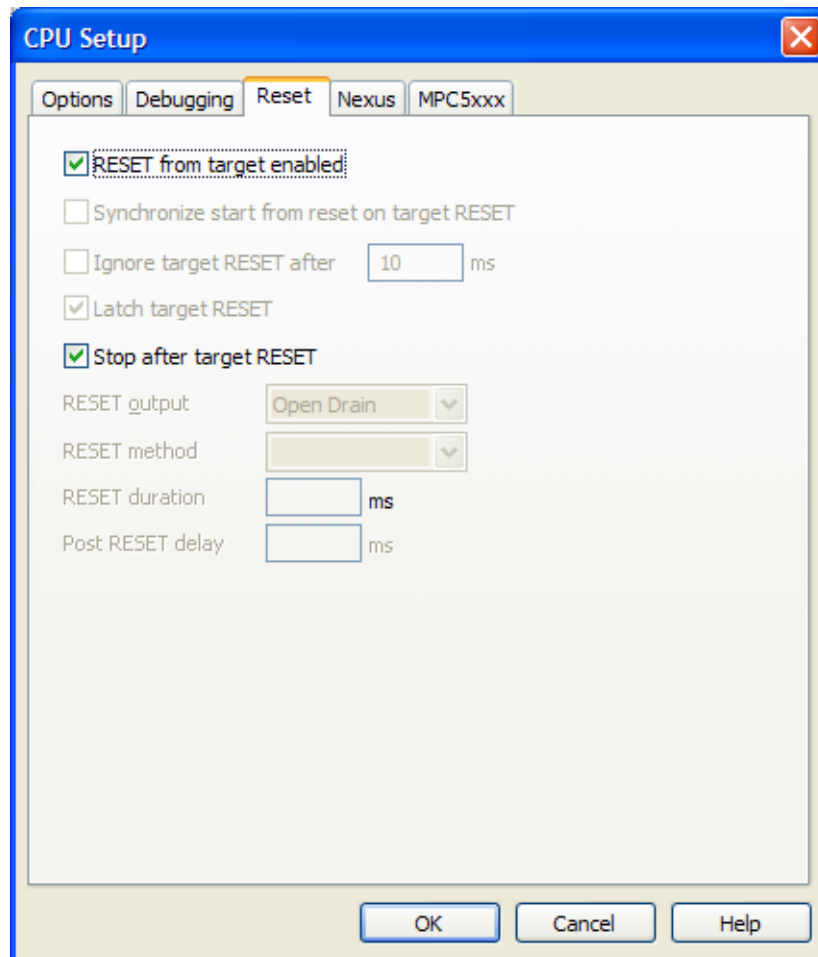
When checked, the debugger identifies memory access errors for individual memory location(s). When the option is unchecked, the debugger would declare access error for remaining memory locations once one access error is detected within a memory read block, which is used in the disassembly window or memory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

Program only modified FLASH sectors

Optionally, when a download is performed into an already programmed device, only the FLASH sectors which have been modified are programmed. This speeds up download times when only small changes are performed.

Note: This option is available in winIDEA build 9.12.4 or newer.

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.

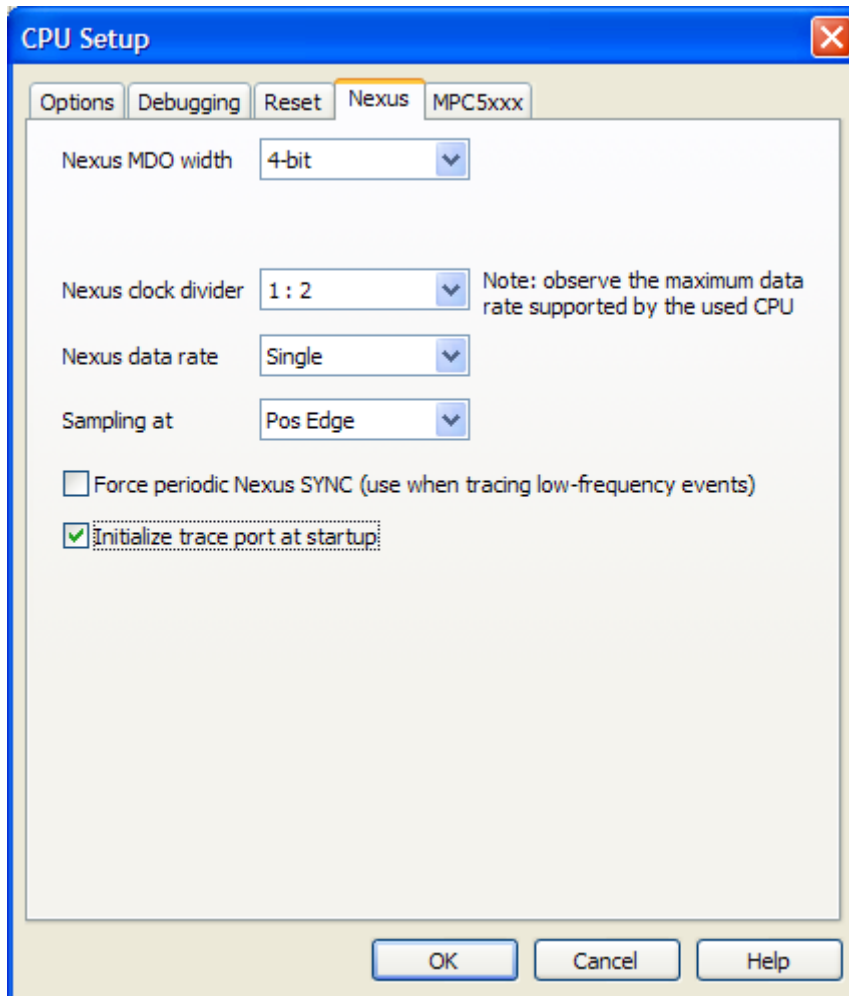
Note: Wrong setting of this option can significantly change the operation of the target!

Stop after 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

Note: This tab is available only when debugging is performed via iC5000 or iTRACE PRO/GT system. Only configuration options which are relevant to the selected CPU and platform are displayed.



Nexus MDO width

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

The MPC5500 devices 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.

MPC551x device can have either 4-bit or 8-bit Nexus MDO port and MPC56xx devices can have either 2-bit or 4-bit Nexus MDO port.

The debugger displays available Nexus MDO width based on selected CPU. See also overview table on page 2 for available MDO width for different microcontrollers.

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 to 1:2 only when otherwise Nexus clock would exceed the maximum frequency of the Nexus port.

Note: Refer to the CPU's *Reference Manual/AC Specifications/PAD AC Specifications* 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 (at every clock edge), which yields double 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

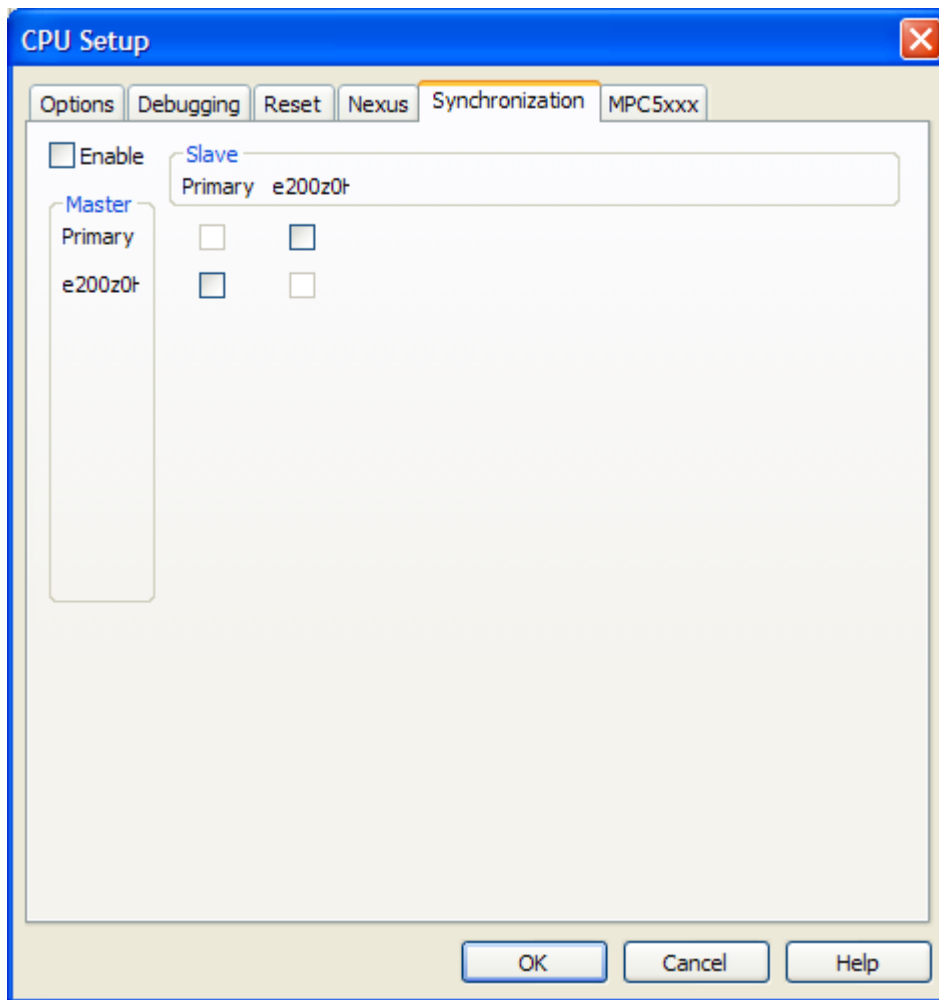
When there are problems with the trace or profiler (for instance no content is displayed), it is recommended to check this option for the test. If the application runs in some loop, which generates no SYNC nexus messages it may happen that the trace decoder cannot reconstruct any absolute CPU address 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

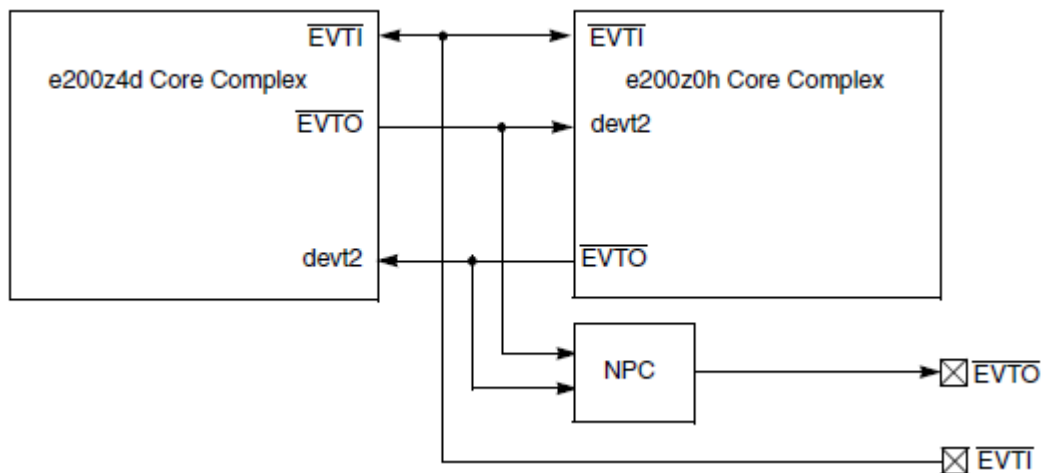
On some CPUs, certain CPU registers must be configured before Nexus trace can be used.

3.5 Synchronization

Note: This tab is available only for multi-core devices (e.g. MPC5646C, MPC5668G)

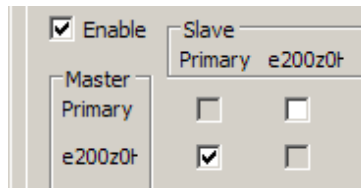


Below block scheme shows how EVTO (Nexus event out pin) and EVTI (Nexus event in pin) are connected between the two cores on MPC5646C.

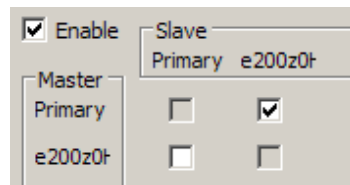


Each core can be configured in a way that EVTO is generated when the associated application is stopped (e.g. breakpoint hit) and that the core is stopped at devt2 request.

With below configuration, secondary core is a master and will stop primary core whenever it stops.

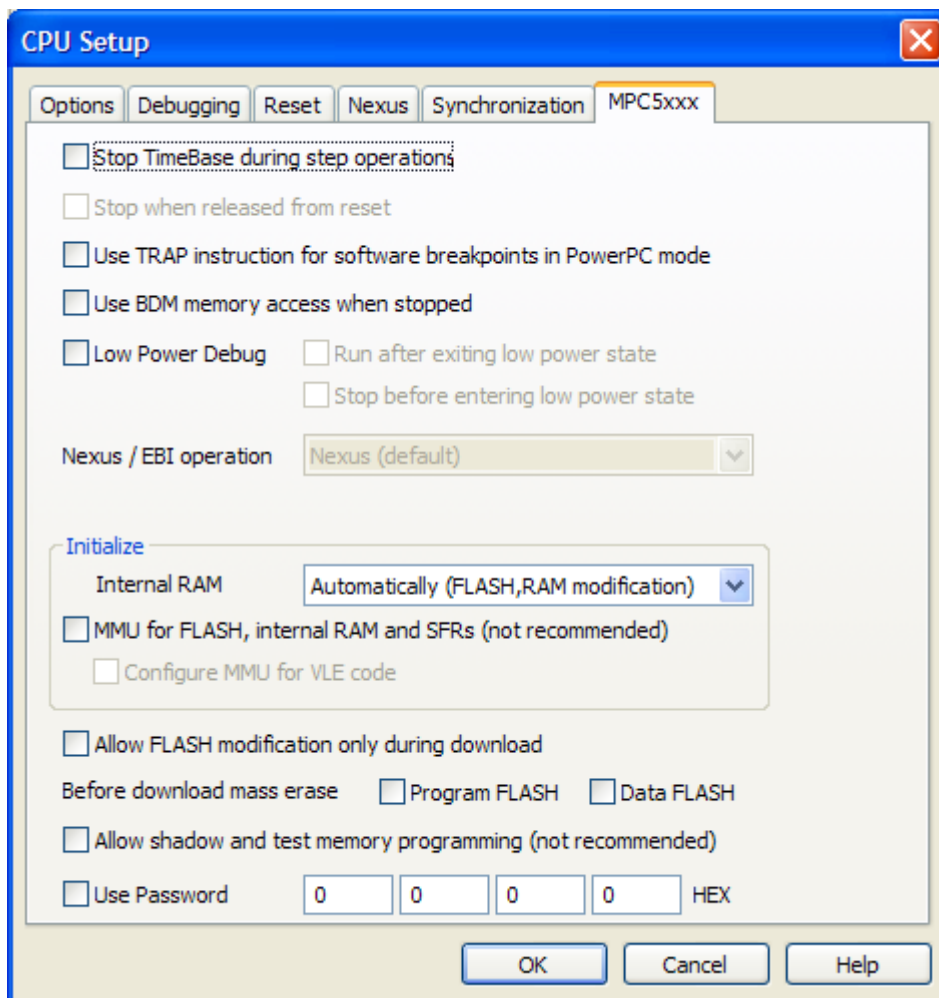


With next configuration, primary core is a master and will stop secondary core whenever it stops.



If both ticks are checked, it means both cores will stop whenever any of the cores is stopped.

3.6 MPC5xxx Options



Stop TimeBase during step operation

When checked, the debugger disables the TimeBase during the debug step operation. This yields more predictable behaviour of the timers and interrupts depending on the timers during debugging.

This option directly controls the TBEN bit in the HID0 register (core register). On some microcontrollers, this bit directly controls all the peripheral clocks already while on some MCUs individual peripheral clock control is additionally controlled via the associated FRZ bit (for example, the FRZ bit in the PITMCR register controls the timers of the Periodic Interrupt Timer (PIT) module when the MPC5643L is in debug mode).

This option is typically used in combination with the ‘Stop CPU Activities When Stopped’ in the ‘CPU Setup/Options’ tab.

Note: This option is not available for all microcontrollers

Stop when released from reset

This option is available when debugging e200z0 core of the MPC5516 device. After power on, the e200z0 core is in reset state and no debug operation can be performed. It’s recommended to check the option when there is a need to debug the e200z0 startup code.

Note: This option is not available for all microcontrollers

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).

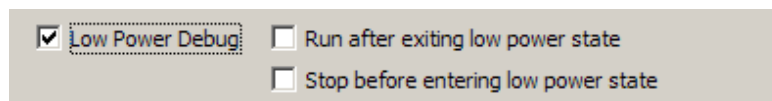
Use BDM memory access when stopped (MPC553x/555x/556x only)

When checked, the BDM interface will be used to access memory rather than Nexus interface. BDM interface is slower, but it allows caches to remain untouched to view memory. With Nexus access, caches are flushed when CPU is stopped.

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.

Nexus / EBI operation (MPC551x only)

MPC551x devices share the same pins between (debug) Nexus port and EBI bus. The user must select how the pins are used in his application. If Nexus operation is selected, external bus can not be used. If EBI operation is selected, Nexus trace is available except that trigger is not available.

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.

Before download mass erase

Optionally, user can force mass erase of Program and/or Data flash instead of the default sector erase being used by the debug download.

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.

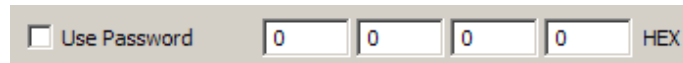
During the debug download and a memory write through the memory window complete flash block is read first then erased and after that modified data is written to flash (read/modify/write). There is no check where data is written to. Password for example (also invalid!) can be set in this way. Debugging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



The image shows a software interface element for setting a password. It consists of a checkbox labeled "Use Password" which is currently unchecked. To the right of the checkbox are four adjacent input boxes, each containing the hexadecimal digit "0". To the right of these input boxes is the label "HEX".

4 Flash

4.1 Programming

Internal program and data flash are programmed through the standard debug download. The debugger identifies which code from the download file fits in the internal flash and programs it during the debug download. Only sectors, which are programmed, are erased.

To speed up the flash programming through the debug download, check the '[Program only modified Flash sectors](#)' option in the Hardware/Emulation Options/CPU Setup/Debugging' dialog.

Mass erase is performed manually via the 'Hardware/FLASH -> Mass Erase' command. It erases both, program and data flash.

A standard FLASH setup dialog accessible from the FLASH menu is used only for programming external flash devices.

Internal flash memory can be modified:

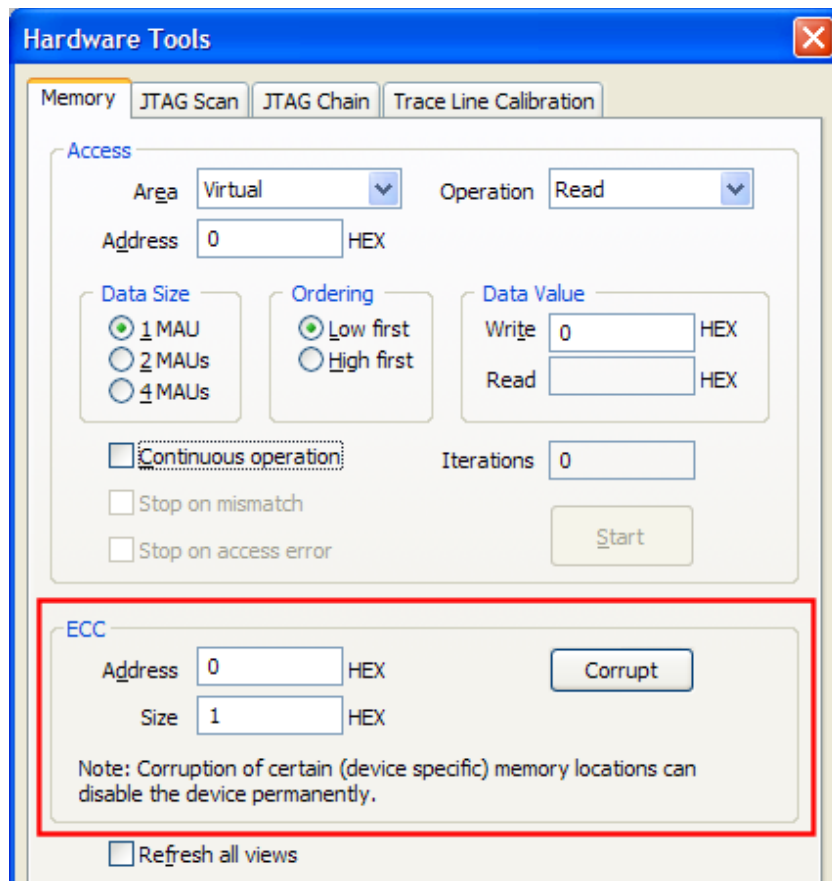
- through the debug download
- via the initialization sequence which is executed before the debug download
- anytime on manual request through the target debug download (without resetting the microcontroller once the debug session is active)
- through the memory window
- through the 'Hardware/Tools/Memory' tab
- from other scripting (e.g. Python) and programming (e.g. C) languages accessing winIDEA through the `isystem.connect` interface

Note: The debugger resolves all ECC (Error Correction Code) errors prior to erasing and programming the flash.

4.2 ECC Corruption

If ECC of the flash or SRAM is to be tested, ECC error can be induced via the 'Hardware/Tools/Memory' dialog.

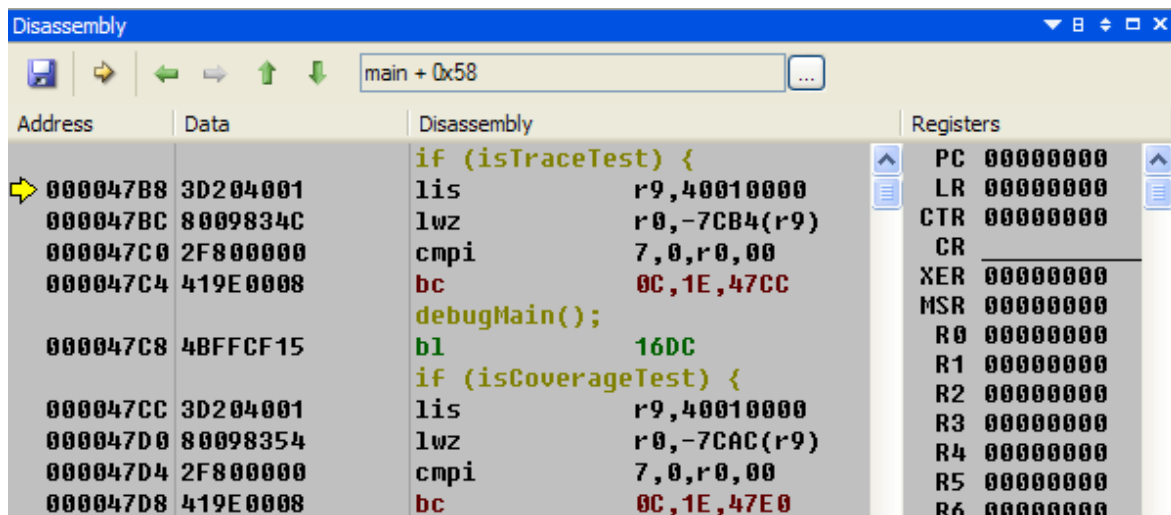
ECC corruption is realized by overwriting FLASH locations. If this is done indiscriminately, the device can be rendered permanently unusable.



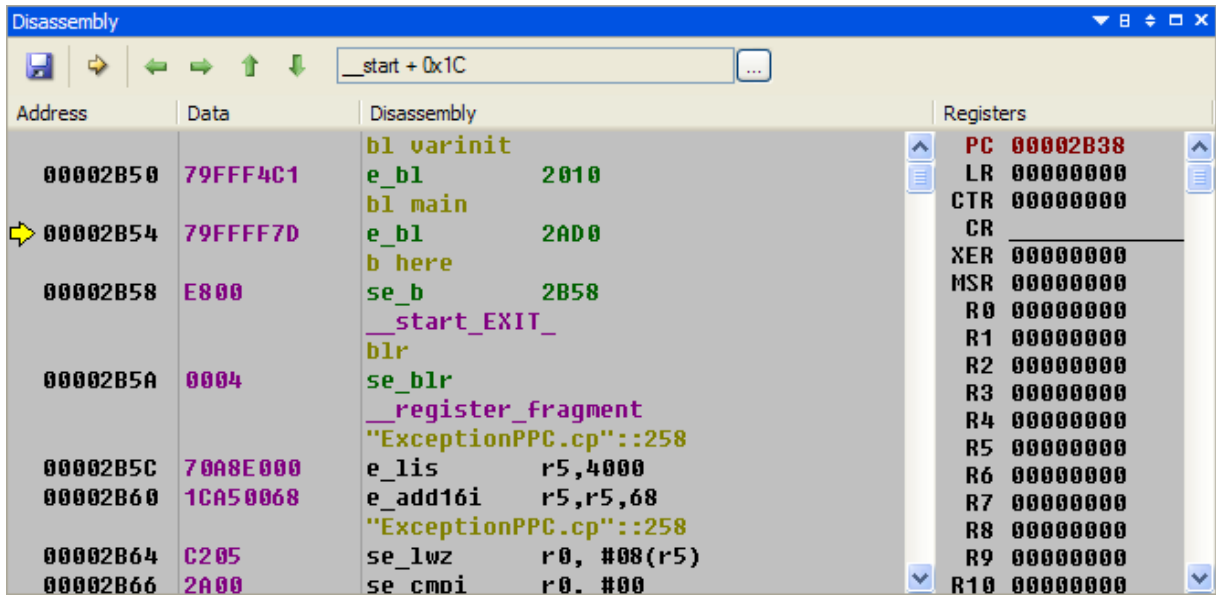
5 VLE

VLE stands for a variable length encoding and is an instruction set enhancement allowing reduced code size footprint. MPC5500 family (e200z1 core) introduces VLE besides a standard 32-bit Power ISA instruction set. Note that there are few MPC5500 devices which do not have VLE (MPC5553 and MPC5554). Check specific CPU reference manual for supported VLE. Second core (e200z0) of the MPC551x has VLE instruction set only.

The debugger supports both instruction sets. Some compilers (e.g. Metrowerks) generate a debug info which allows the debugger to distinguish between the code belonging to one instruction set or to the other. This allows the debugger to color the code differently, so the user evidently sees which instruction set is executed. VLE code is colored purple.

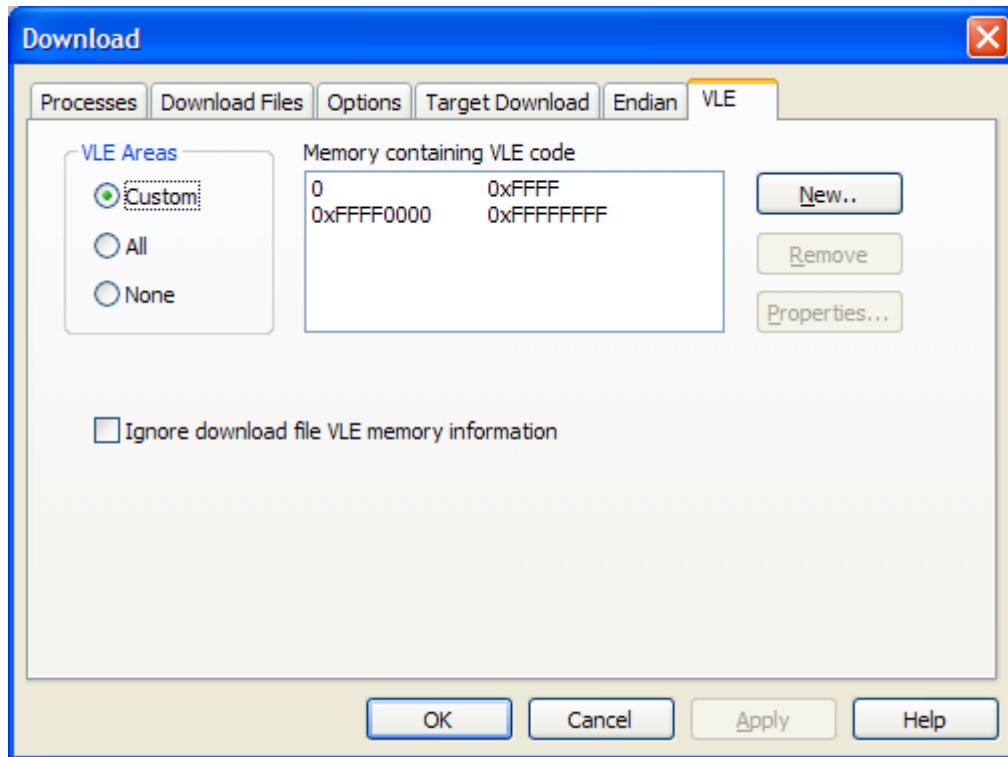


Standard PowerPC instruction set



VLE instruction set

If the debugger cannot automatically recognize which code from the download file is the standard PowerPC or VLE type, the user can manually define the VLE areas in the 'Debug/Files For Download.../VLE' tab.

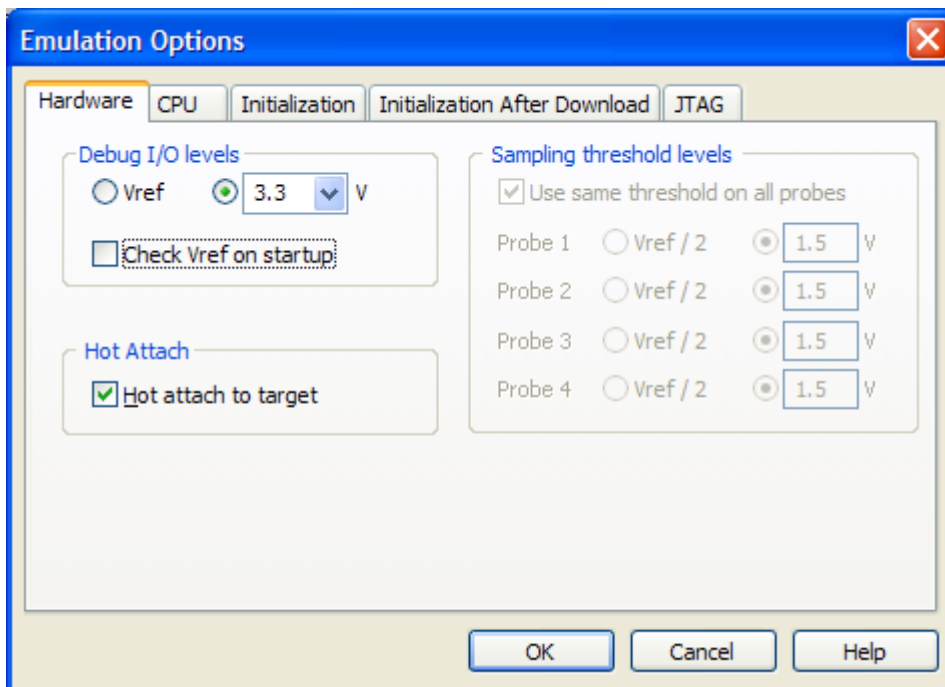


6 Hot Attach

The development system allows attachment to a running target system without affecting its operation. Such operation is called Hot Attach.

It's assumed that there is a running target with no debugger connected. To hot attach:

- Check the 'Hot attach to target' option in the 'Hardware/Emulation Options/Hardware' tab.
- Execute Download debug command.
- Connect the debug 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 if necessary.

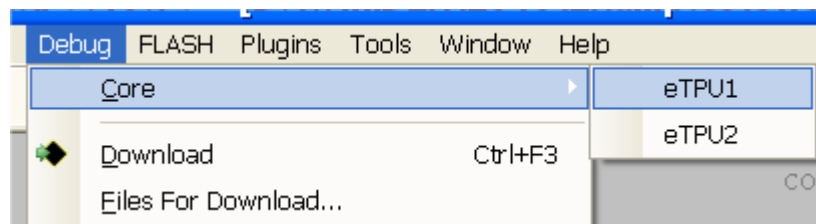
Select 'Detach' debug command in the 'Debug' menu to disconnect from the target application. If the microcontroller was stopped before detach, it will be set to running.

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

7 eTPU Debugging

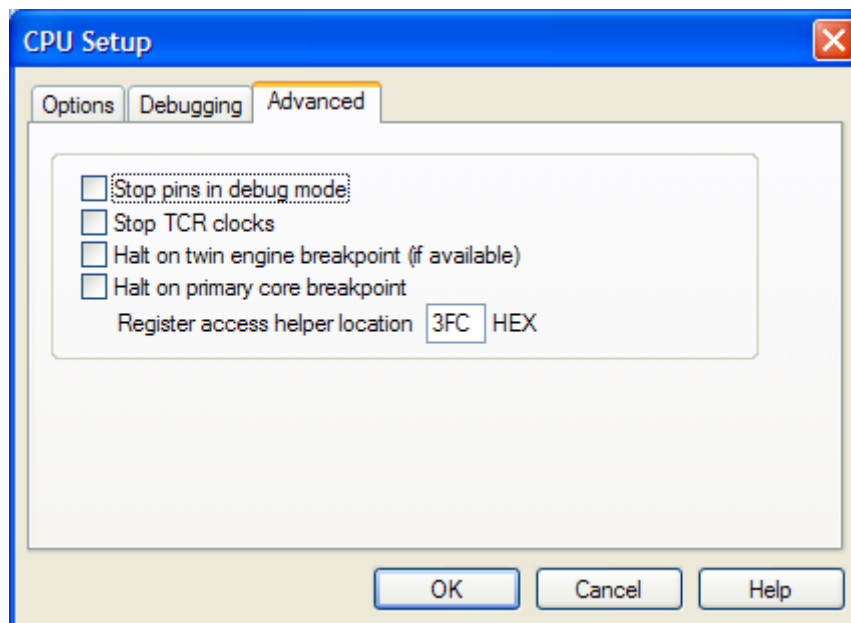
The Enhanced Timing Processor Unit (eTPU) has its own Nexus class 3 interface, the Nexus Dual eTPU Development Interface (NDEDI). The Nexus Dual eTPU Development Interface provides real-time development capabilities for the eTPU system including two engines and the coherent dual parameter controller (CDC) in compliance with the IEEE-ISTO 5001-2002 standard. The main development features supported are hardware execution breakpoints, register and memory access, instruction and source step, run/stop control, branch (program) trace, data trace and ownership trace. Combined, these features make the interface for each engine compliant with Class 3 of the IEEE-ISTO 5001-2002 standard.

In order to debug the MPC5500 application, first a main winIDEA workspace window must be open, which allows compiling the complete project, downloading the code in the CPU internal flash and debugging the e200 core, eDMA and FlexRay. In order to debug the eTPU1 or the eTPU2, a new winIDEA session is open for each eTPU being debugged from the 'Debug/Core' menu.



In general, eTPU1 and eTPU2 modules are debugged in the same way as e200 core. Each eTPU module has its own winIDEA instance with standard debug windows as disassembly window, memory window, watch window, source window, trace window, etc. However, note that the code for eTPUs is loaded by the main workspace, which loads the code into the internal flash. eTPU winIDEA session must therefore download only symbols, which are required for instance for high-level eTPU debugging. Don't forget to specify the necessary download files for each eTPU winIDEA session.

The 'Hardware/Emulation Options/CPU Setup/Advanced' tab introduces few new options, which are eTPU specific:



The Nexus eTPU Development Interface provides real-time development capabilities for the eTPU system including two engines (one for each eTPU module) and the coherent dual-parameter controller (CDC). Refer to the Nexus Dual eTPU Development Interface chapter in the eTPU Reference Manual for more details on below options.

Stop pin in debug mode

This option controls whether the eTPU engine pins are sampled when any of the eTPU engine enters debug mode. When it's checked, the pins are not sampled during debug mode or when executing a forced instruction from the microinstruction register.

Stop TCR clocks

This option controls whether the TCR clocks from the eTPU engine stop running when the eTPU enters debug mode.

Halt on twin engine breakpoint (if available)

It controls the action taken on breakpoint occurrences at the twin engine. If the twin engine that is causing a breakpoint exit debug state, the engine shall resume its operation.

Halt on primary core breakpoint

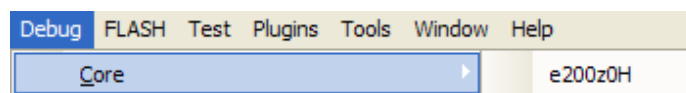
When checked, the eTPU module is stopped when the main e200 core is stopped.

Register access helper location

The shared data memory (SDM) works as data RAM that can be accessed by the device core and up to two eTPU engines. This memory is alternatively referred to as eTPU shared parameter RAM (SPRAM). When any of the eTPU modules is in debug mode, one 32-bit SPRAM location is hidden used by the debugger. Before the eTPU operation is resumed, the debugger restores the used location SPRAM with the original value. The problem arises when one eTPU is stopped (in debug mode) while the other is still running and using exactly the same SPRAM location, which the debugger uses to control the first eTPU. In order to prevent this, the user should use this option and tell the debugger, which SPRAM address is not used by the application and can be freely used for debugging.

8 Debugging 2nd Core

When microcontroller features two or more cores, each core is debugged in an individual winIDEA instance. In order to debug the second e200 core (for example 2nd e200z0 core of the MPC551x), first primary winIDEA workspace window is open, which allows compiling the complete project, downloading the code in the CPU internal flash and debugging the e200 primary core. In order to debug the 2nd core, a new winIDEA session is open for the 2nd core from the 'Debug/Core' menu.



2nd core is debugged more or less in the same way as primary core. 2nd core winIDEA instance has all standard debug windows as disassembly window, memory window, watch window, source window, trace window, etc. However, note that the code for 2nd core is loaded by the primary workspace, which loads the code into the internal flash. 2nd core winIDEA session must therefore download only symbols, which are required for example for a high-level debugging. Don't forget to specify the necessary download file for the 2nd winIDEA session.

9 Real-Time Memory Access

The 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.

Note: Real-time memory access is not supported on MPC5601D, MPC5602D, MPC5602P and SPC560P40 due to on-chip debug restrictions..

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 disabled.

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 CPU 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 behaviour 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.

10 MMU Support

Normally MMU is initialized by Boot Assistant Module (BAM). After reset the CPU first executes code from BAM, which initializes MMU, then reads reset configuration word and starts running user code.

However, the debugger must initialize MMU when some debug option is used that circumvent BAM to be initialized upon reset (e.g. overriding startup PC using 'CPU Setup/Advanced' dialog). The debugger can configure MMU through Initialization sequence use (see Section 2.2). When no such option is used, the debugger doesn't need to configure MMU.

Below is an excerpt from the initialization file (.ini), which initializes MPC5554 MMU in case when the program counter is preset (overridden) after the CPU reset.

```
S "SPR":(270) L 0x10000000
S "SPR":(271) L 0xC0000500
S "SPR":(272) L 0xFFFF000A
S "SPR":(273) L 0xFFFF003F
I 7C0007A4
S "SPR":(270) L 0x10030000
S "SPR":(271) L 0xC0000400
S "SPR":(272) L 0x40000008
S "SPR":(273) L 0x4000003F
I 7C0007A4
S "SPR":(270) L 0x10040000
S "SPR":(271) L 0xC0000500
S "SPR":(272) L 0xC3F00008
S "SPR":(273) L 0xC3F0003F
```

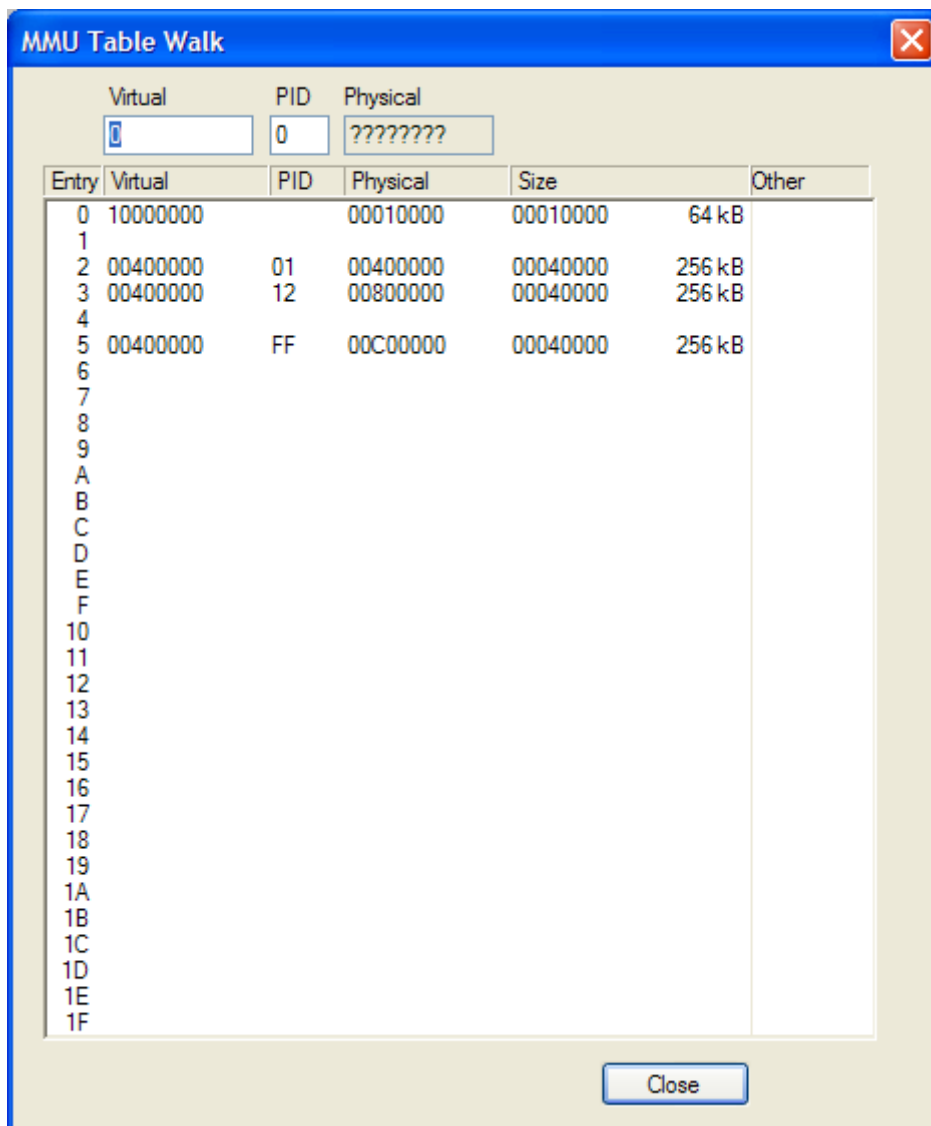
```

I 7C0007A4
S "SPR":(270) L 0x10010000
S "SPR":(271) L 0xC0000700
S "SPR":(272) L 0x00000000
S "SPR":(273) L 0x0000003F
I 7C0007A4
S "SPR":(270) L 0x10020000
S "SPR":(271) L 0xC0000700
S "SPR":(272) L 0x20000000
S "SPR":(273) L 0x2000003F
I 7C0007A4

```

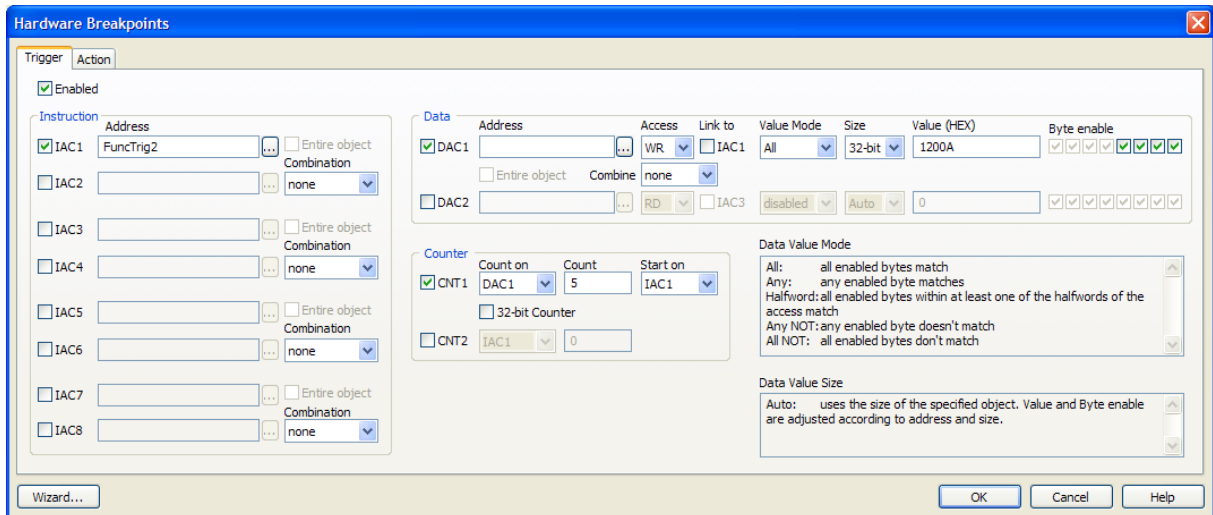
Virtual to physical address mapping of the individual memory pages is displayed in the MMU Table Walk window. The window can be selected from the 'Debug/MMU Table Walk' menu for all CPUs with MMU.

A direct entry of a virtual address is possible.



The MMU Table Walk dialog

11 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 of the MPC5xxx family includes up to eight Instruction Address Comparators (IAC1-8), out of which each can be configured for a specific address or two in pair can be configured for an address range, 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.

On some microcontrollers a data value (DVC1-2) can be set for each of the data address comparators (DAC1-2) and/or a counter (CNT1-2) can be additionally used to count the comparator matches. The breakpoint hits when the counter reaches 0.

Note: Hardware breakpoints dialog above shows maximum amount of possible on-chip debug resources. Some MPC5xxx devices implement only a subset of these resources. Only implemented on-chip debug resources are enabled and can be configured in the dialog depending on the select target CPU.

Microcontroller Nickname	IAC	DAC	DVC	CNT
Komodo	8	2	2	2
Fado	4	2	0	2
Leopard	8	2	2	2
Andorra	8	2	2	2
Mamba	8	2	2	2
Spectrum	4	2	2	0
Monaco	4	2	2	0
Rainbow	8	2	2	2
MPC556x	4	2	0	2
Bolero	4	2	2	0
3M Bolero	8	2	2	2
Pictus	4	2	2	0
MPC551x	4	2	0	0
MPC553x	4	2	0	2
MPC555x	4	2	0	2

Available access breakpoints comparators

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.

12 Trace, Profiler and Execution Coverage

Trace, Profiler and Execution Coverage are supported by iC5000 and iTRACE GT 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.

13 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 some 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 location to which the reset vector points
- 6) Open memory window at internal CPU RAM location and check whether you are able to modify its content.
- 7) If you passed all 6 steps successfully, the debugger is operational and you may proceed to download the code in the internal CPU flash.
- 8) Specify the download in the 'Debug/Files for download/Download files' tab.
- 9) Execute Debug download, which should download the code in the internal CPU flash.

14 Troubleshooting

- While debugging MPC551x devices, the internal watchdog must be disabled because the watchdog is not stopped when the application is stopped (microcontroller is in the debug mode). In such case, the watchdog could reset the microcontroller while the application is stopped and the debugger would lose the control over it. Set WTE bit in the RCHW (Reset Configuration Halfword) to 0. Write 0x01 to the RCHW instead of 0x05.
- Q: I have problems using 'Go To' address after the debug download and debug CPU reset. I'm getting this error message "MMU TLB Entry for this address not found. Ensure correct MMU configuration". What's the problem?

A: MMU is not configured for the address range to which you preset program counter after the CPU reset. Either don't use 'Go To' option and the application will configure MMU accordingly or refer to Initialize 'MMU for FLASH, internal RAM and SFRs (not recommended)' option description in the chapter 3.6.

- Q: When I run the code after the download it never reaches the main function. Flash programming doesn't report any verify error. What could be the problem?

A: Most probably the application remains in the BAM program because specific "startup" parameters were not programmed properly in the internal flash. After the reset, the CPU starts executing a so called BAM (Boot Assist Module) program. Refer to CPU reference manual for more details on BAM. The BAM searches the internal flash memory for a valid reset configuration halfword (RCHW). A valid RCHW is a 16-bit value that contains a fixed 8-bit boot identifier and some configuration bits, which define location of boot code and the boot configuration options. If the BAM program does find a valid RCHW, the watchdog is enabled, the BAM program fetches the reset vector from the address of the `BOOT_BLOCK_ADDRESS + 0x4`, and branches to the reset boot vector. Check if valid RCHW and valid reset vector are set in user application and programmed through the debug download.

- In case of problems when accessing SFRs on MPC56xx devices, check if clock(s) for peripheral modules are switched on.

For instance, for Bolero device with 512KB internal flash, write 0x80808000 to the `CGM_SC_DC0` register (address 0xC3FE037C) before accessing any SFR. This switches on clocks for all peripheral modules.

Some devices have more `CGM_SC_DCn` registers controlling microcontroller peripheral modules. For instance on one ST device, additionally 0x000000FE needs to be written to the `ME.RUNPC[0].R` in order to make things working. Refer to the microcontroller reference manual of the particular device for more details on how to enable access to the available peripheral modules.

If this is the problem, use the Hardware/Emulation Options/Initialization Sequence to enable access to the peripheral modules and don't forget to add this write also to the application startup code since the issue is not related to the debugging only. After reset, peripheral modules are powered off to save power. If these peripheral registers are read by the application or the debugger, a bus error is generated and the CPU gets stuck.

- Make sure that the power supply is applied to the target JTAG connector when 'Vref' is selected for Debug I/O levels in the 'Hardware/Emulator Options/Hardware' tab, otherwise emulation fails or may behave unpredictably.
- Try 'Slow' JTAG Scan speed if the debugger cannot connect to the CPU.
- When performing any kind of checksum, remove all software breakpoints since they may impact the checksum result.

- Q: Whenever we add the pointer in the watch window and expand the elements in it, and then try to continue debugging, all the values get corrupt and in the disassembly we see "Illegal Instructions". We don't try to modify the contents of the pointer, we only read it. What is the problem?

A: Yes, when the debugger reads from an unmapped memory region, the CPU resets or the debug interface freezes and a debug reset is required then. **You should avoid reading from unmapped memory regions.** In your case when you expanded ConfigPtr, some fields, which were read, point to an unmapped memory region. Afterwards you made a debug source step, which made the CPU irresponsive. This could also happen if you read from unmapped memory region using SFR, Memory or Watch window. This phenomenon has been observed on MPC560x, SPC560P and SPC560B devices.
- 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:

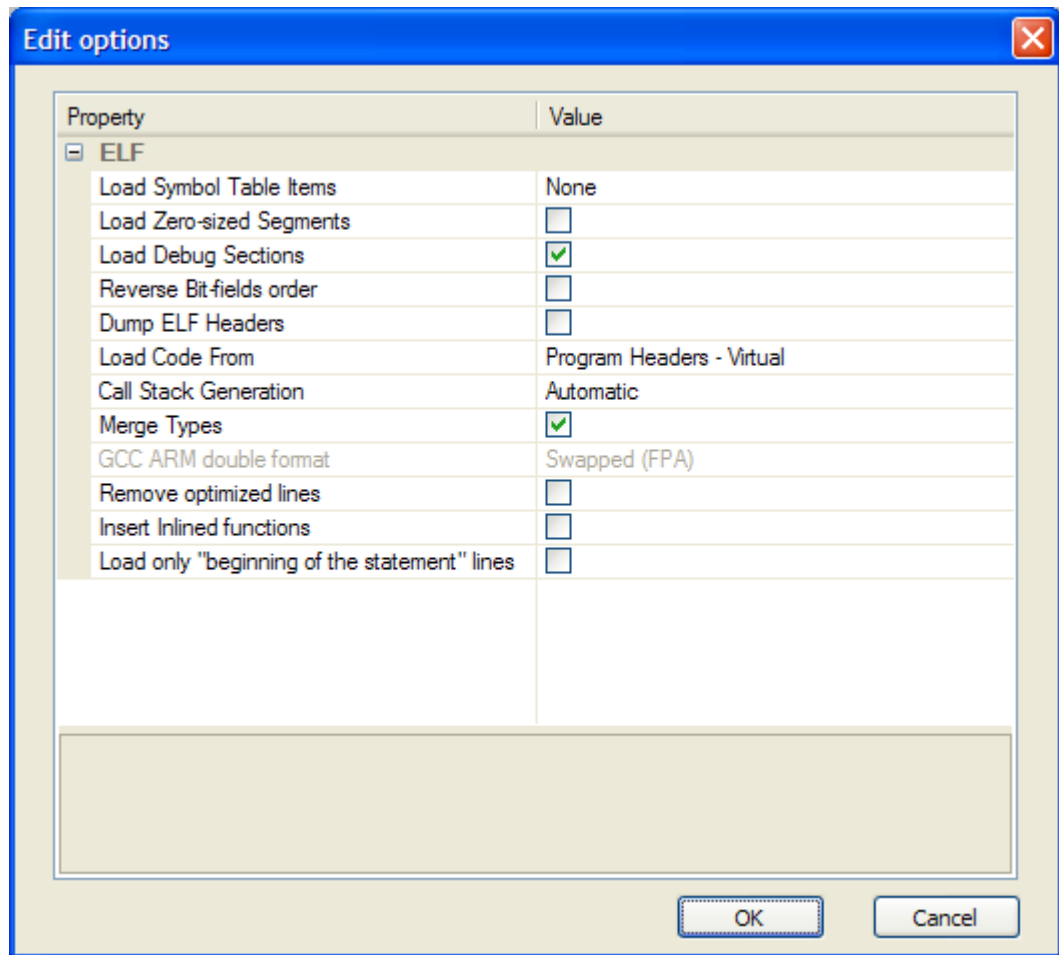
```

=====
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.

- MPC5675k can sometimes stay in SAFE mode which disables MCKO signal and trace cannot function. As a workaround use the below init sequence which switches CPU from SAFE into DRUN mode:

```
A RGM_FES W 0x4080
A ME_MCTL L 0x30005AF0
A ME_MCTL L 0x3000A50F
```

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.