

---

## Technical Notes

# Freescal MPC55xx & MPC56xx 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.....	10
3.4 MPC55xx Options.....	11
4 FLASH programming.....	12
5 VLE .....	12
6 Hot Attach .....	13
7 eTPU Debugging.....	14
8 e200z0 Debugging.....	15
9 Real-Time Memory Access .....	16
10 MMU Support .....	16
11 Access Breakpoints .....	18
12 Trace, Profiler and Execution Coverage.....	19
13 Getting Started.....	19
14 Troubleshooting.....	20

# 1 Introduction

This document covers MPC55xx and MPC56xx on-chip debugging while Nexus trace port use is explained in separate documents. Nexus trace port is explained separately for MPC551x & MPC56xx devices (Nexus Class 2+) and for other MPC55xx devices (Nexus Class 3).

The MPC55xx Family built on Power Architecture technology, comprises 32-bit microcontroller devices designed for engine management, advanced driver assistance, central body, and gateway applications.

The MPC55xx family is based on e200z1 core, which features a memory management unit (MMU) and the full 32-bit Power ISA instruction set as well as the ability to implement variable length encoding (VLE) instructions. A small footprint e200z0 core is added to the MPC5510 devices, which is designed to run the variable length encoding (VLE) instructions only, which delivers a high level of code density, significantly reducing memory requirements.

According to the Nexus standard, MPC5500 family contains multiple Nexus clients that communicate over a single IEEE-ISTO 5001-2003 Nexus Class 3 (or 2+) combined JTAG IEEE 1149.1 auxiliary out interface. Combined, all of the Nexus clients are referred to as the Nexus development interface (NDI). Class 3 Nexus allows for program, data and ownership trace of the microcontroller execution without access to the external data and address buses. Class 2+ Nexus has no data trace comparing to the Class 3 Nexus.

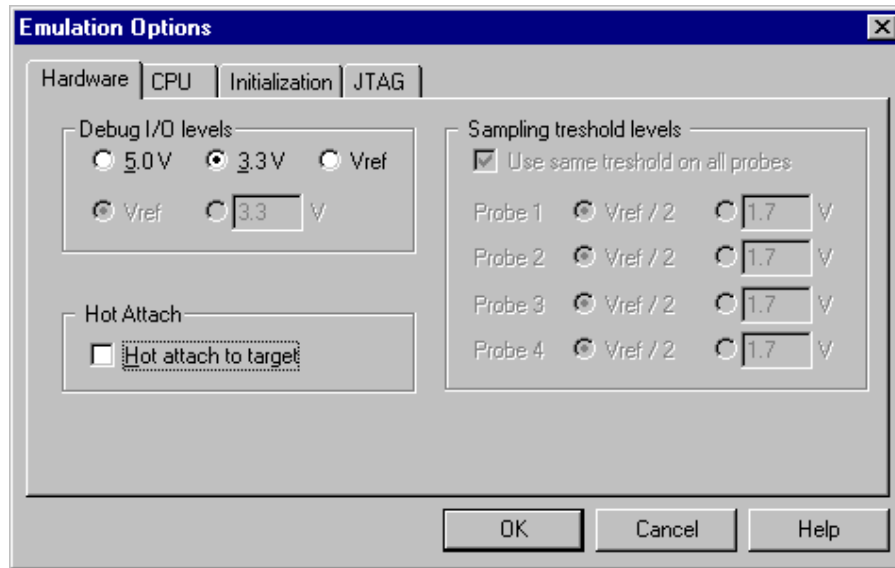
Communication to the NDI is handled via the auxiliary port and the JTAG port. Debug JTAG clock frequency must not exceed half CPU system clock frequency.

## Features

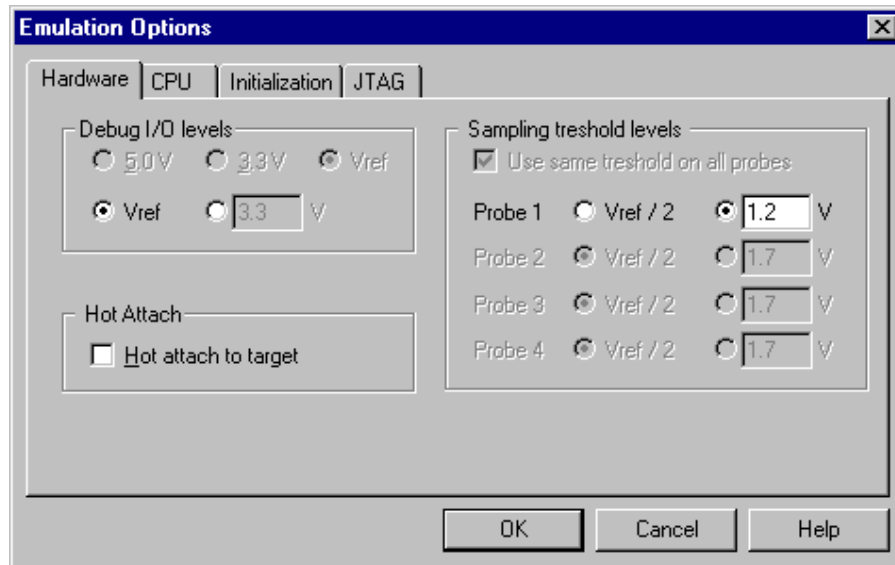
- Four hardware execution breakpoints
- Unlimited software breakpoints including in the internal CPU flash
- Access breakpoints
- Real-time memory access
- Flash programming
- Hot Attach
- MMU support
- eTPU debugging
- e200z0 debugging (MPC551x)
- On-Chip Nexus Trace (e200z1, e200z0, eTPU1, eTPU2, eDMA, FlexRay)
- Nexus RTR Trace (e200z1 core – full 32-bit Power ISA instruction set)
- Profiler
- Execution Coverage

## 2 Emulation Options

### 2.1 Hardware Options



*Emulation options, Hardware pane (Debug iCARD)*



*Emulation options, Hardware pane (iTRACE PRO/GT)*

#### ***Debug I/O levels***

The development system can be configured in a way that the debug JTAG signals are driven at 3.3V, 5V or target voltage level (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.

### ***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!

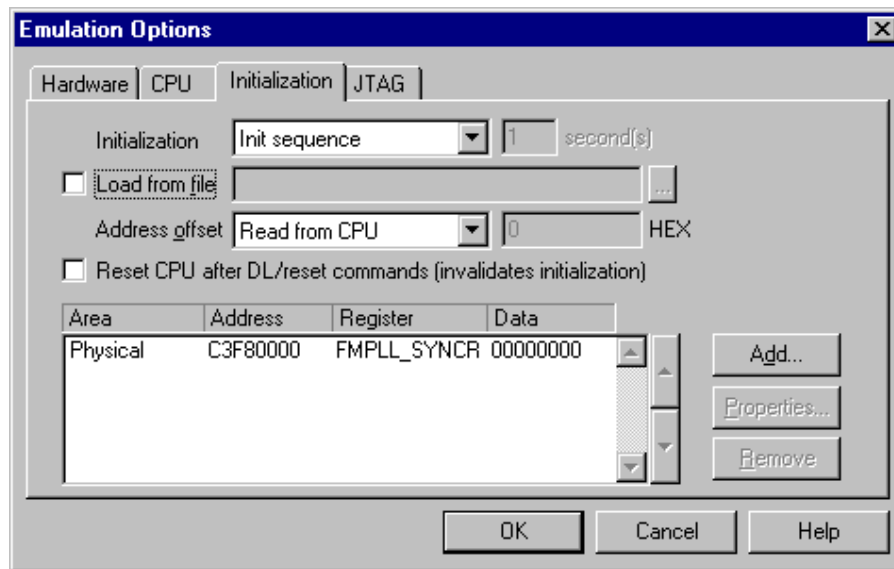
---

## **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 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

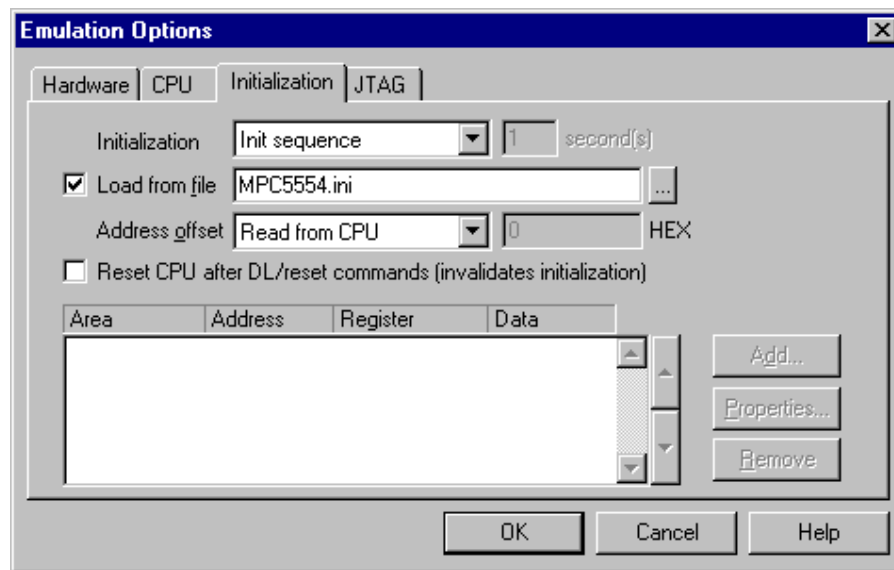
Excerpt from the example MPC5554.ini file

```
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.

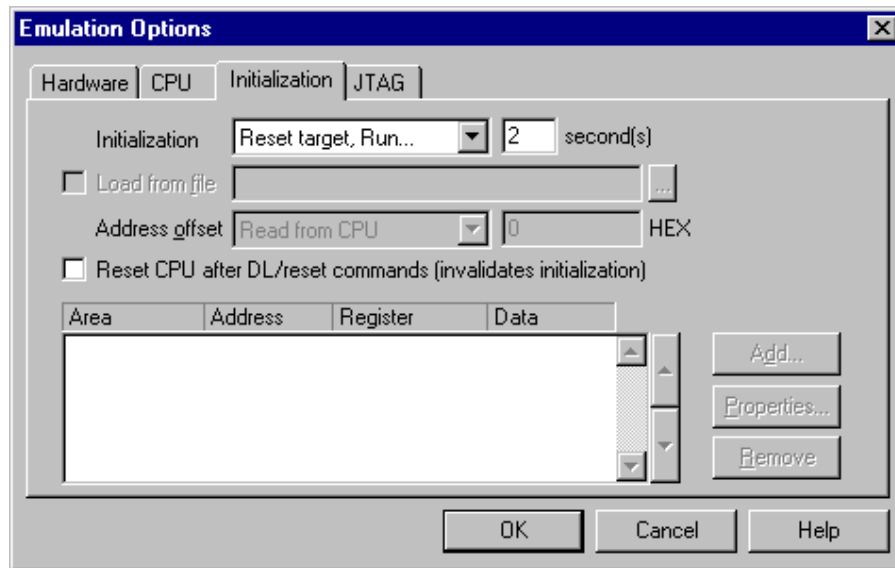
```
A "SPR":(270) L 0x10000000 // MAS0
A "SPR":(271) L 0xC0000500 // MAS1
A "SPR":(272) L 0xFFFF000A // MAS2
A "SPR":(273) L 0xFFFF003F // MAS3
I 7C0007A4 // tlbwe
```

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.

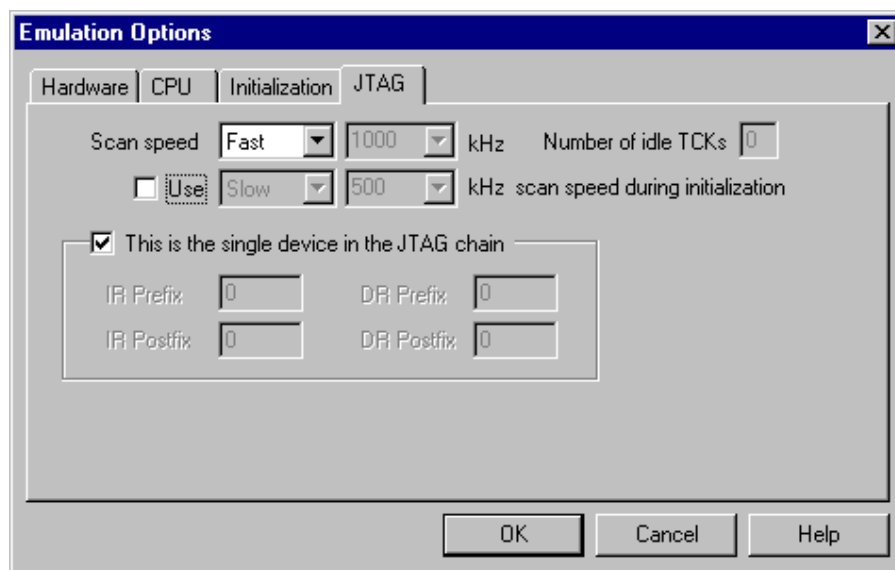


The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

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.



## 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.
- Other scan speed types (not supported for MPC5500 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.

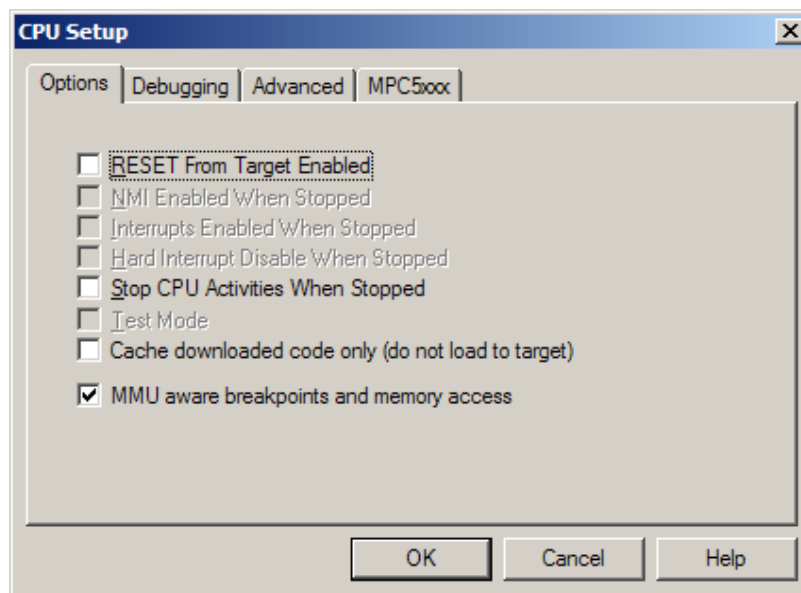
### ***Use – Scan Speed during Initialization***

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**

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*

#### ***RESET from Target Enabled (MPC56xx only)***

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 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.

### ***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.

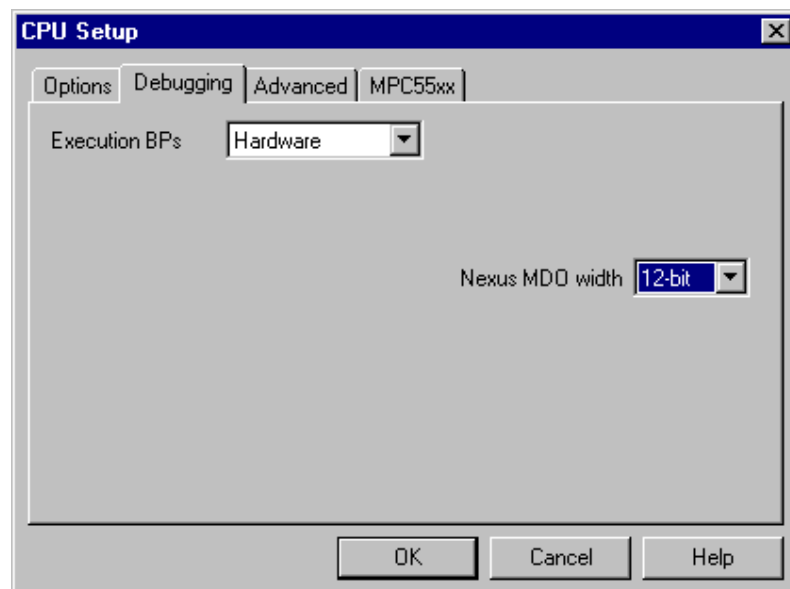
In cases, where the application is previously programmed in the target or it's programmed through the Flash Programming dialog, the user may uncheck 'Load code' in the 'Properties' dialog when specifying the debug download file(s). By doing so, the debugger loads only the necessary debug information 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.

## **3.2 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. 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 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.

---

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 MPC55xx 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.

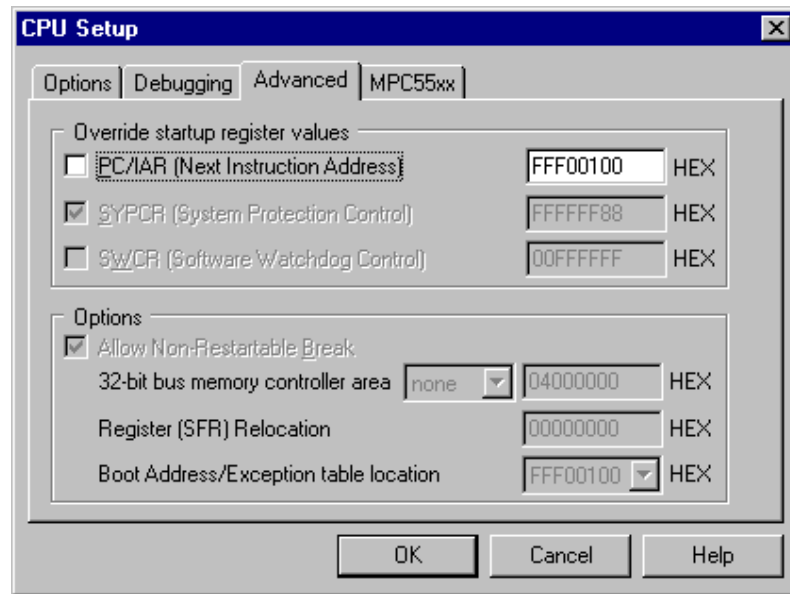
### ***Nexus MDO width***

This setting is available only for the development system supporting Nexus on-chip trace (iTRACE PRO/GT).

Nexus signals are located on the 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.

### 3.3 Advanced Options



*PowerPC Family Advanced Options*

#### ***Override Startup Register Values***

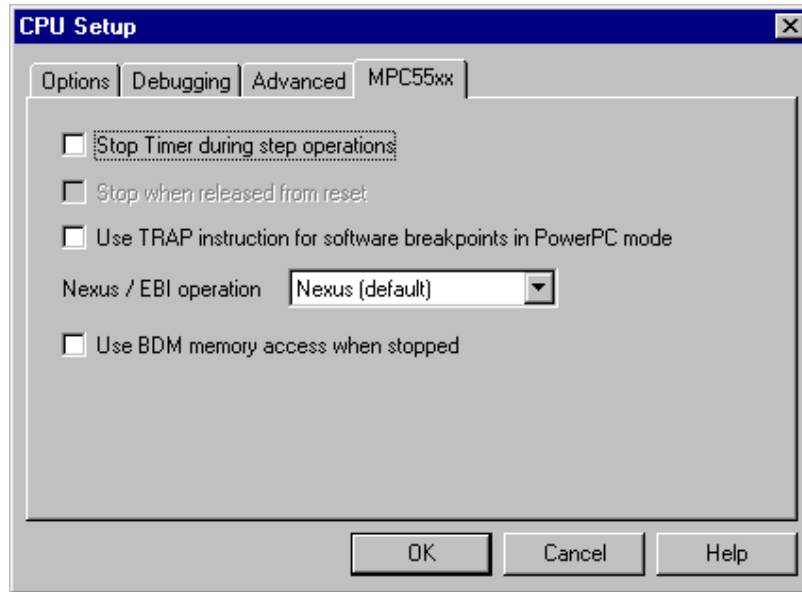
If required, the Emulator can change the checked register after the CPU is released from reset. These settings have typically to do with setting the default program starting point and disabling the watchdog.

---

Caution: If in Hardware/Emulation options/Initialization the one of the Reset&Run options is used the run after reset will be started from this address. A Run attempt will fail, if there is no valid code there right after reset. It is recommended that the two options, Override PC and Initalization/Reset&Run, are not used at the same time.

---

### 3.4 MPC55xx Options



#### ***Stop Timer during step operations***

When checked, the debugger disables the clock to the peripherals during the debug step operation. This yields more predictable behaviour of the timers and the interrupts depending on the timers during debugging. This option is typically used in combination with the 'Stop CPU Activities When Stopped' in the 'CPU Setup/Options' tab.

#### ***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.

#### ***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).

#### ***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.

#### ***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.

## 4 FLASH programming

The MPC55xx and MPC56xx devices have internal flash, which is programmed through the standard debug download. The debugger recognizes which code from the download file fits in the internal flash and programs it during the debug download.

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

## 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.

winIDEA 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.

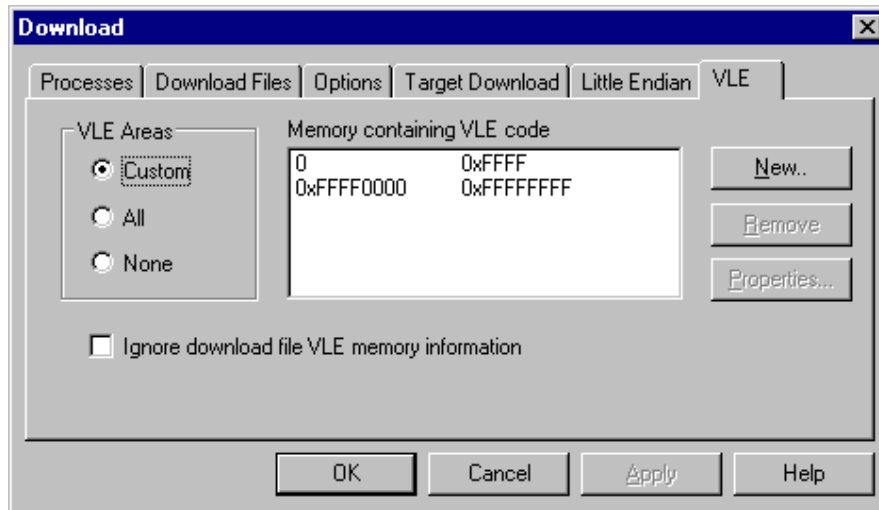
Address	Data	Disassembly	Registers
		LED	PC 00010140
		{	LR 00002054
00010140	9421FFF0	stwu r1,-10(r1)	CTR 00002254
00010144	7C0802A6	mflr r0	CR E
00010148	90010014	stw r0,14(r1)	XER 00000000
		if(++LEDdelay>0x200)	MSR 00000000
0001014C	806D8018	lwz r3,-7FE8(r13)	R0 0000000000000013
00010150	38030001	la r0,bam_rchw+1 (01)(r3)	R1 000000004000DFE0
00010154	900D8018	stw r0,-7FE8(r13)	R2 0000000040008020
00010158	28000200	cmpli 0,0,r0,0200	R3 0000000000000012
0001015C	40810014	bc 04,01,"LED.c"::61 (0001017	R4 0000000000000002
		LEDdelay=0;	R5 0000000000000003
00010160	38000000	li r0,bam_rchw (00)	R6 0000000000000005
00010164	900D8018	stw r0,-7FE8(r13)	R7 000000004000DFD0
		ToggleLED(3);	R8 0000000000000005

Standard PowerPC instruction set

Address	Data	Disassembly	Registers
		Type_Simple	PC 00002056
		{	LR 00002018
00002056	10210600	e_stwu r1,-30(r1)	CTR 00000000
0000205A	0000	se_mflr r0	CR L
0000205C	DD01	se_stw r0,#34(r1)	XER 00000000
0000205E	1D610030	e_add16i r11,r1,30	MSR 00000000
00002062	780004D5	e_bl _savegpr_25 (2536)	R0 0000000001000000
		char c=0;	R1 000000004000DFE0
00002066	480B	se_li r27,#00	R2 0000000040008020
		unsigned char uc=0;	R3 0000000000000002
00002068	480A	se_li r26,#00	R4 00000000FFFF0000
		int i=0;	R5 0000000000000003
0000206A	480E	se_li r30,#00	R6 0000000000002000
		unsigned int ui=0;	R7 000000004000DFD0
0000206C	480F	se_li r31,#00	R8 00000000FFFF0000

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.

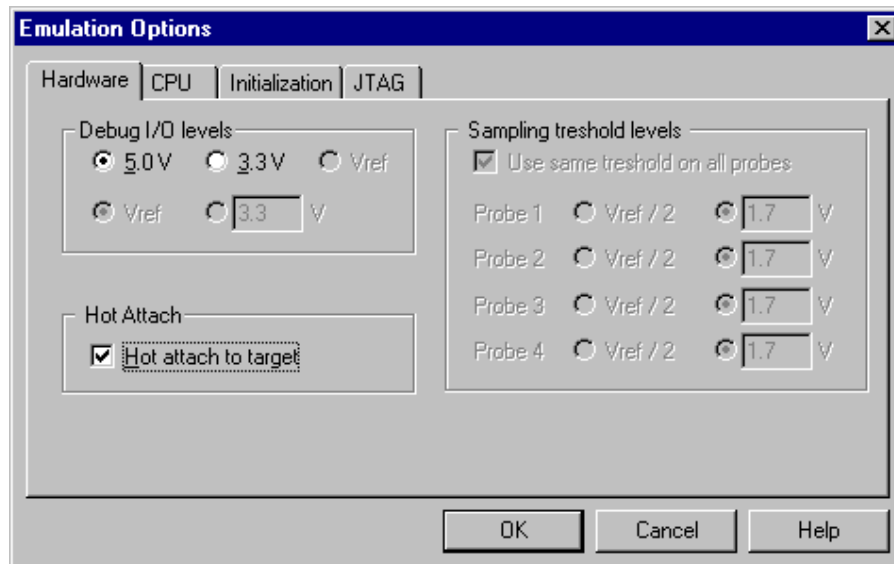


## 6 Hot Attach

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.

---

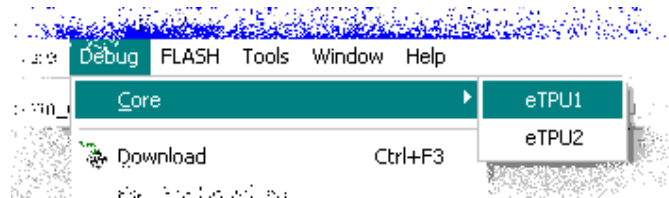
Note: Hot Attach function cannot be used for any flash programming or code download!

---

## 7 eTPU Debugging

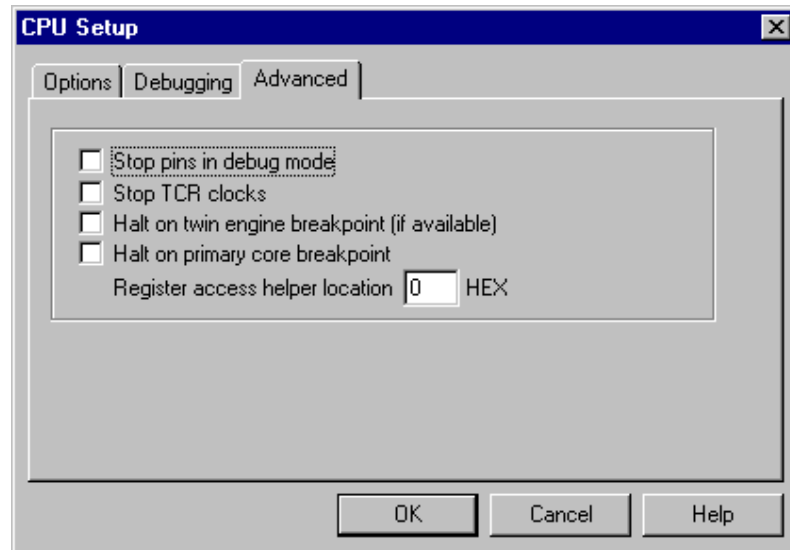
The Enhanced Timing Processor Unit (eTPU) has its own Nexus class 3 interface, the Nexus Dual eTPU Development Interface (NDEDI). The Nexus Dual eTPU Development Interface provides real-time development capabilities for the eTPU system including two engines and the coherent dual parameter controller (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***

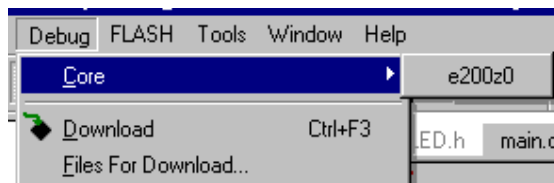
When checked, the eTPU module is stopped when the main e200 core is stopped.

### ***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 e200z0 Debugging**

In order to debug the second e200z0 core of the MPC551x, 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 e200z1 core. In order to debug the e200z0 core, a new winIDEA session is open for the e200z0 core from the 'Debug/Core' menu.



e200z0 core is debugged in the same way as e200z1 core. e200z0 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 e200z0 core is loaded by the main workspace, which loads the code into the internal flash. e200z0 winIDEA session must therefore download only symbols, which are required for example for a high-level e200z0 debugging. Don't forget to specify the necessary download file for the e200z0 winIDEA session.

## 9 Real-Time Memory Access

MPC5500 and MPC56xx family 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 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
```

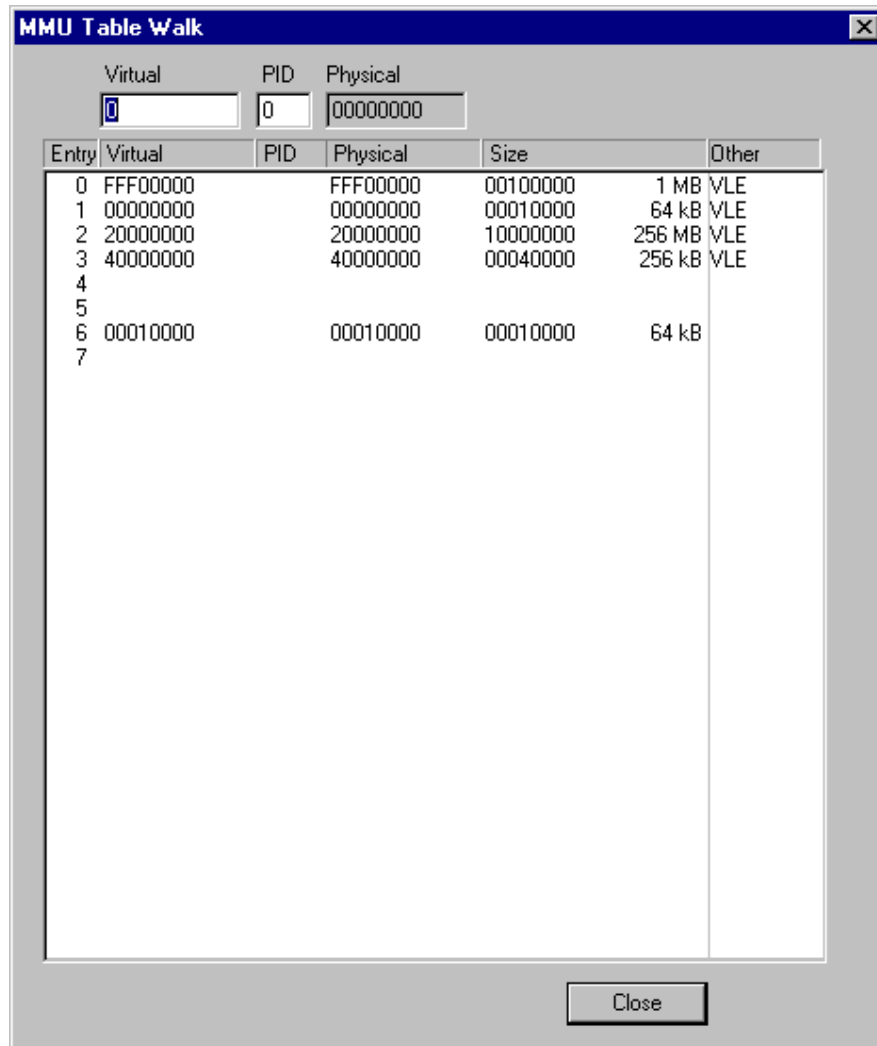
```

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

```

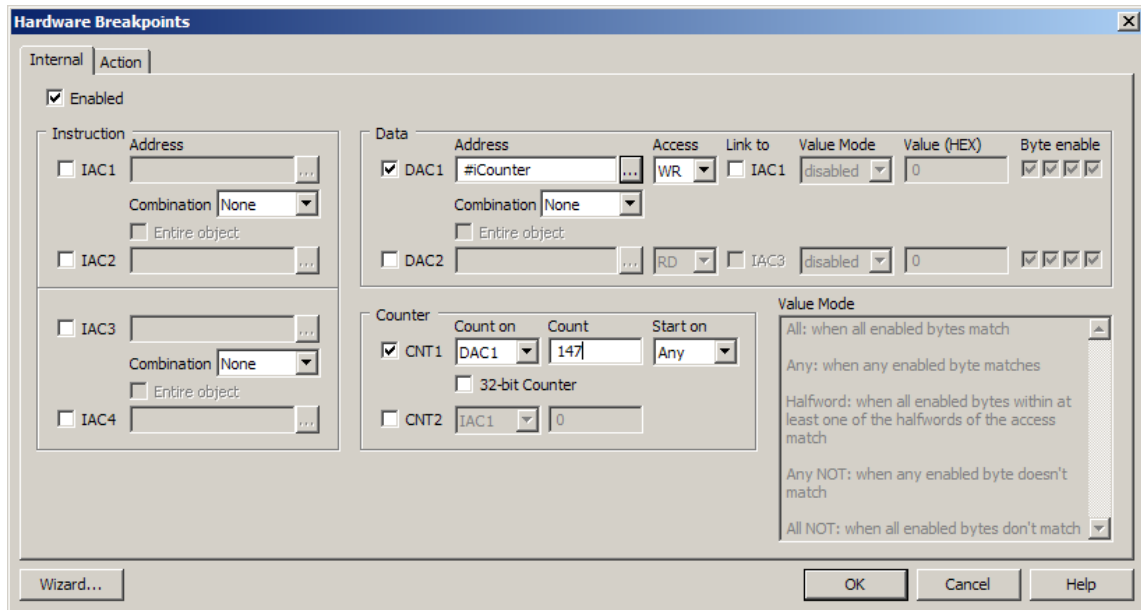
Virtual to physical address mapping of the individual memory pages is displayed in the MMU Table Walk window. The window can be selected from the 'Debug/MMU Table Walk' menu for all CPUs with MMU.

A direct entry of a virtual address is possible.



*The MMU Table Walk dialog*

# 11 Access Breakpoints



*MPC5500 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 MPC55xx family includes four Instruction Address Comparators (IAC1-4), which can be set to four addresses or two 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.

In case of MPC56xx, also data value can be set for each of the data address comparators (DAC1-2)

Further, a counter (CNT1-2) can be used to count the event occurrence. The breakpoint hits when the counter reaches 0.

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

## Wizard...

Use Wizard (button in the left bottom corner) in case of problems understanding and configuring the access breakpoints dialog. It helps setting a simple breakpoint on data access.

## 12 Trace, Profiler and Execution Coverage

Trace, Profiler and Execution Coverage are supported by the iTRACE PRO/GT development system. These functionalities are explained in a separate technical notes document, titled Freescale MPC553x/555x/556x Nexus Trace and Freescale MPC551x & MPC56xx Nexus Trace.

## 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: 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, write 0x80808000 to the `CGM_SC_DC0` register (address 0xC3FE037C) before accessing any SFR. This switches on clocks for all peripheral modules. If this is the problem, use the Hardware/Emulation Options/Initialization Sequence 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, some 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.

---

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.