Freescale 68K Family In-Circuit Emulation

This document is intended to be used together with the CPU reference manual provided by the silicon vendor. This document assumes knowledge of the CPU functionality and the terminology and concepts defined and explained in the CPU reference manual. Basic knowledge of winIDEA is also necessary. This document deals with specifics and advanced details and it is not meant as a basic or introductory text.

Contents

1 In-Circuit and Active Emulation introduction

Debug Features

- Unlimited breakpoints
- Access breakpoint
- Real-time access
- Trace
- Execution coverage

1.1 Differences from a standard environment

The In-Circuit Emulator and the Active Emulator can emulate a processor or a micro-controller. Beside the CPU, additional logic is integrated on the POD. The amount of additional logic depends on the emulated CPU and the type of emulation. A buffer on a data bus is always used (minimal logic) and when rebuilding ports on the POD, maximum logic is used. As soon as a POD is inserted in the target instead of the CPU, electrical and timing characteristics are changed. Different electrical and timing characteristics of used elements on the POD and prolonged lines from the target to the CPU on the POD contribute to different target (the whole system) characteristics. Consequently, signal cross-talks and reflections can occur, capacitance changes, etc.

Beside that, pull-up and pull-down resistors are added to some signals. Pull-up/pull-down resistors are required to define the inactive state of signals like reset and interrupt inputs, while the POD is not connected to the target. Because of this, the POD can operate as standalone without the target.

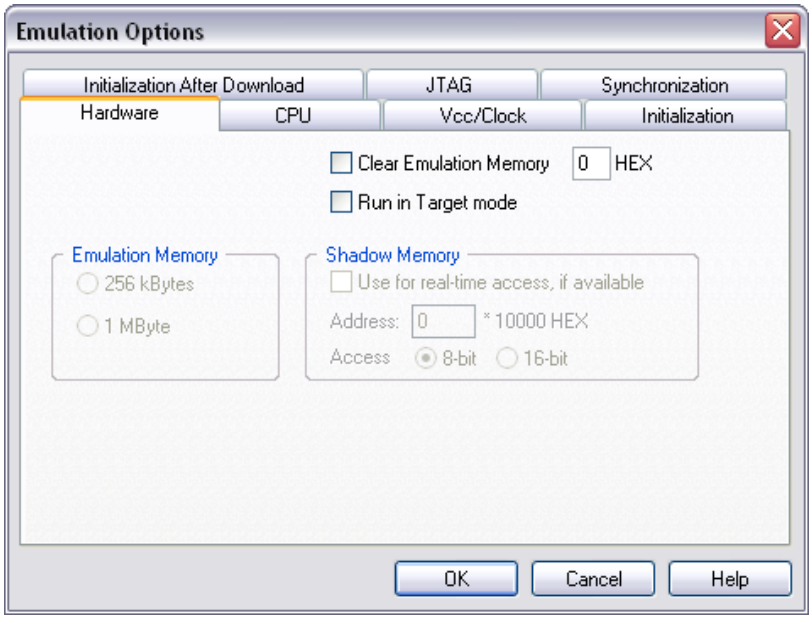
1.2 Common Guidelines

Here are some general guidelines that you should follow.

- Use external (target) Vcc/GND if possible (to prevent GND bouncing),
- Make an additional GND connection from POD to the target if the Emulator behaves strangely,
- Use the reset output line on the POD to reset the target whenever Emulator resets the CPU,
- Make sure the appropriate CPU is used on the POD. Please refer to the POD Hardware reference received with your POD.
- No on-chip or external watchdog timers can be used during emulation (unless explicitly permitted). Disable them all.
- When interrupts in background are enabled, take note that the interrupt routine must return in 25 ms, otherwise the Emulator will assume that the program is hung.

2 Emulation Options

2.1 Hardware Options



In-Circuit Emulator Options dialog, Hardware page

Emulation Memory

Defines the size of emulation memory available on the In-Circuit emulation module.

Note: You must specify the memory size correctly, otherwise the Emulator will not initialize.

Clear Emulation Memory

This option allows you to force clearing (with the specified value) of emulation memory after the emulation unit is initialized.

Clearing emulation memory takes about 2 seconds per megabyte, so use it only when you want to make sure that previous emulation memory contents don't affect the current debug session.

Shadow Memory

On-board shadow memory is provided that allows reading of memory without stopping the CPU or stalling it even for a single cycle. If you wish to use this memory for real-time access, check the 'Use for real-time access if available' option.

If you leave the option unchecked, the 'regular' real-time readout (if available – see "Real-Time Memory Access" on page 18) will be used for real-time access.

The 'Access' setting specifies whether 8-bit or 16-bit real-time access is used.

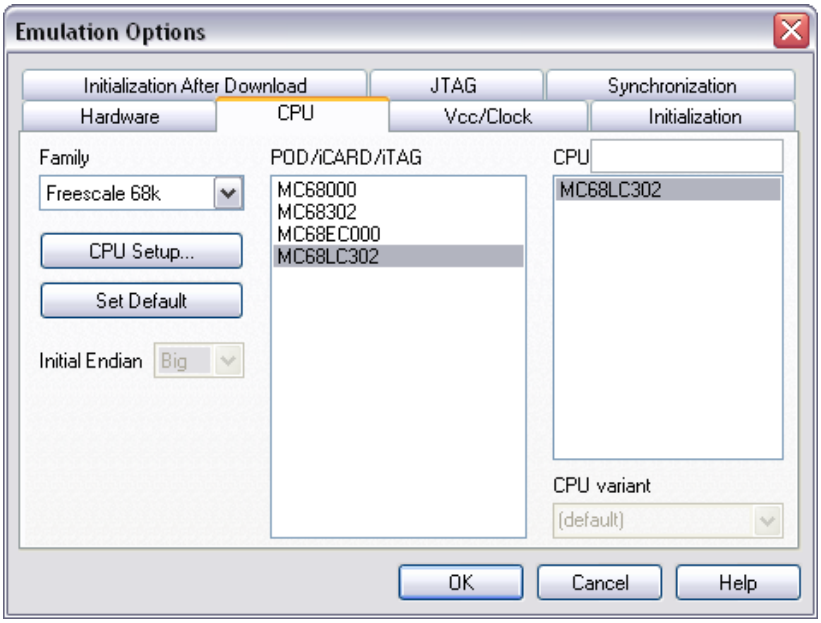
Run in Target mode

This mode is used for troubleshooting purposes only. This function turns off all Emulator functions and makes the system work identically as if the Emulator would not be connected to the target system. This setting is useful if the target system stops working when the Emulator is connected to it; with this setting you can check whether the problem is in the connections between the POD and the target or if the problem lies somewhere else.

When this option is set, the Emulator only controls the clock and the reset line, all other signals are mapped to the target.

2.2 CPU Configuration

With In-Circuit emulation besides the CPU family and CPU type the emulation POD must be specified (some CPU's can be emulated with different PODs).



In-Circuit Emulator Options dialog, CPU Configuration page

CPU Setup

Opens the CPU Setup dialog. In this dialog, parameters like memory mapping, bank switching and advanced operation options are configured. The dialog will look different for each CPU reflecting the options available for it.

Set Default

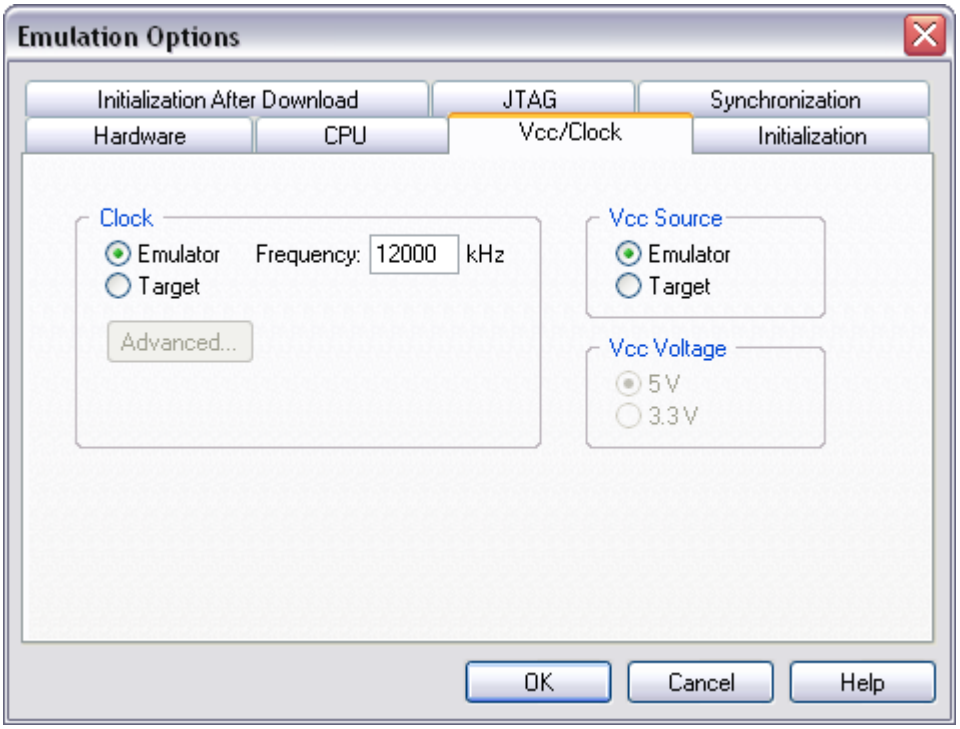
This button will set default options for currently selected CPU. These include:

- Vcc and clock source and frequency
- Advanced CPU specific options
- Memory configuration (debug areas, banks, memory mapping)

Note: Default options are also set when the Family or a POD is changed.

2.3 Power Source and Clock

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



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

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

Clock Source

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

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

When the clock source is set to Internal, the clock is provided by the emulator and you may control its frequency in steps of 1kHz. Depending on the Emulator and its oscillator version, you will be able to use clock from 1MHz to 33 MHz (on iC181) or up to 100MHz on iC2000 emulation units.

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

If the clock source is set to external, the clock is provided by the target system. In certain applications, for instance, a 32.786kHz clock is used. Since the minimal clock the Emulator can generate is 1MHz, an external clock source must be used and the clock source set to external.

Vcc Source

Determines whether emulator or the target system provides power supply for the CPU.

2.4 Initialization Sequence

Usually, there is no need to use initialization sequence when debugging with an In-Circuit Emulator (ICE) a single chip application. Primarily, initialization sequence is used on On-Chip Debug systems to initialize the CPU after reset to be able to download the code to the target (CPU or CPU external) memory. With an ICE system, the initialization sequence may be required for instance to enable memory access to the CPU internal EEPROM or to some external target memory, which is not accessible by default after the CPU reset. The user can also disable CPU internal COP using initialization sequence if there is a need for that, etc.

Initialization sequence is executed immediately after the CPU reset and then the code is downloaded. Detailed information may be found in the Initialization Sequence help topic.

2.5 Pattern Generator

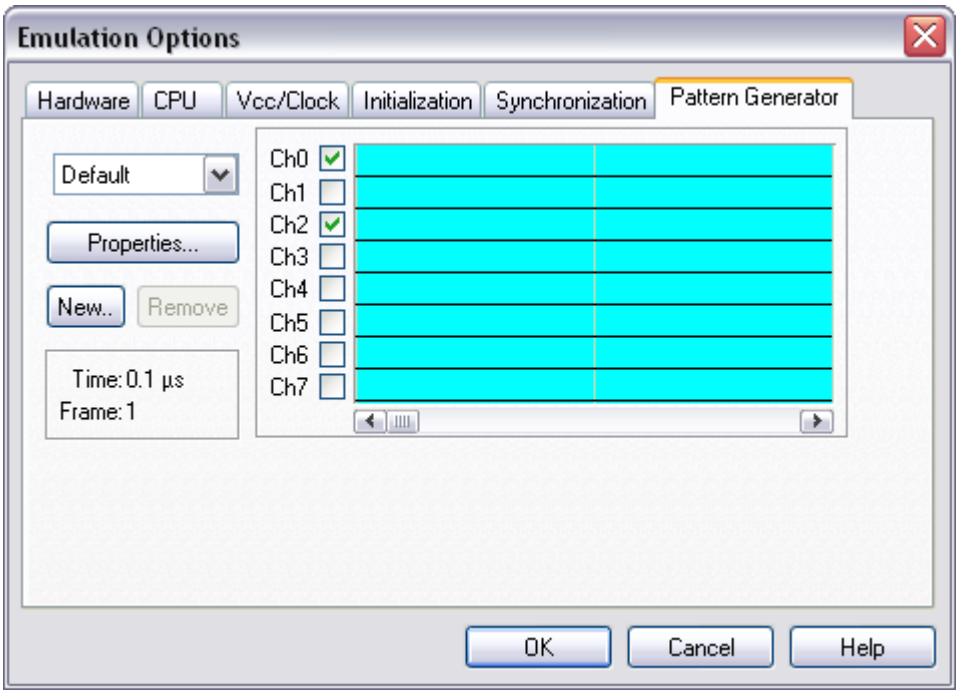
iC2000 and iC4000 provide an 8-channel waveform programmable pattern generator capable of continuous or single shot operation at up to 10MHz-clock rate with up to 512 samples.

Note: when using the iC4000 system, it has in certain configurations two Pattern Generators: one on the base module and one on the Power Emulator module. The Pattern Generator on the base module is active when the debugging type is set to ‘Active Emulation’ or ‘BDM/JTAG Emulation’, the Pattern Generator on the Power Emulator Module is active when ‘In-circuit Emulation’ is selected.

You can configure any number of patterns using 'New…' and 'Remove' buttons. The currently selected pattern is displayed in the combo box as indicated in the above figure.

State of a disabled channel can be configured either to high or low.	Every individual channel can be enabled or disabled by configuring the check box next to its name. When a channel is disabled you can still configure its state, which remains unchanged throughout its period.

Waveforms are configured easily by clicking and moving the mouse cursor on the desired channel and position.



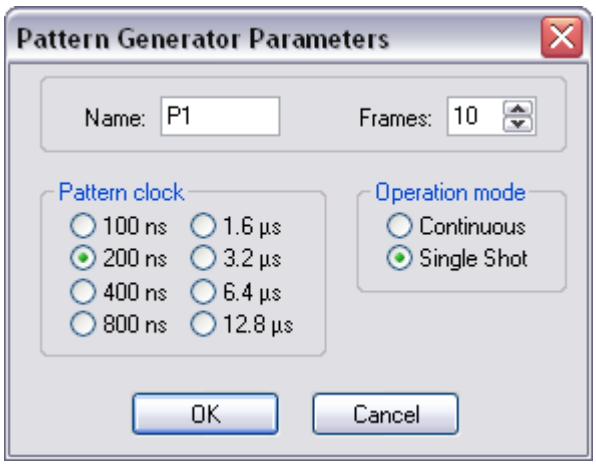
In-Circuit Emulator Options dialog, iC2000/iC4000 Pattern Generator page

Properties

This button opens a dialog where parameters for the current pattern can be configured.

Pattern Generator Parameters

Parameters of a pattern are valid for all of its eight channels. This means that all channels are of the same length and all use the same clock.



iC2000/iC4000 Pattern Generator Parameters dialog

Name

Defines the name of the current pattern

Frames

Defines number of frames used in the pattern. Frames multiplied by pattern clock define the period of the pattern. The number of frames is limited to 512.

Pattern clock

Defines the clock rate by, which the waveform progresses.

Operation mode

Defines whether the pattern is to run continuously or to execute only a single shot on demand. In any case, pattern operation is controlled from the Hardware menu by selecting the 'Run Pattern' command.

When continuous mode is selected, the 'Run Pattern' command will either stop pattern execution (at the last frame), or resume it.

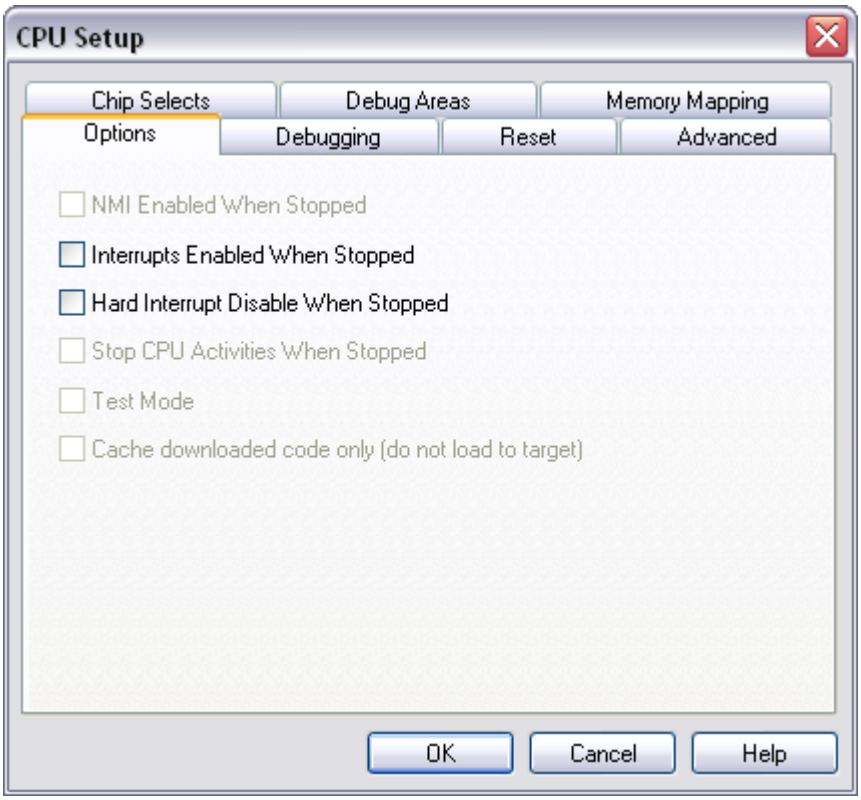
In single shot mode selecting the 'Run Pattern' executes a single pattern shot.

Note: Pattern generator operation can be controlled by an external device through the TRIG/CLKEN pin on the pattern generator connector. Refer to the Hardware User's Manual for more information.

3 Setting CPU options

3.1 CPU 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.



CPU Setup, Options page

"Interrupts Enabled When Stopped" checked

When this option is checked, the Interrupt Enable (I (interrupt) on Freescale CPUs) flag is never modified by the emulator. When the user's program is stopped the emulator doesn't influence the state of Interrupt Enable flag. During program stop any interrupts will always be serviced with the exception when BDM, JTAG or SDI is used. When the CPU enters the BDM mode, the CPU itself cannot service interrupts. Thereby they become pending interrupts and are serviced first after the user's program proceeds with execution.

Note: On all 8 bit CPUs the emulator allows interrupt nesting up to 15 levels in depth, representing no limitations in practice. Nesting will occur only if interrupt servicing is interrupted by another interrupt before the servicing is completed. While any nested interrupt is serviced by the CPU, the emulator has no access to the CPU therefore debug windows cannot be refreshed in the meantime.

To allow background interrupt execution on 8 bit CPUs, interrupt routines must meet the following conditions:

- All CPU registers must be preserved,
- Interrupt routines must return with the corresponding return-from-interrupt instruction (RETI, RFI, etc.). Do not assume that your compiler always gets it right. Interrupt routine exiting with jump or call instruction cannot be debugged.

- The return address must not be changed in the interrupt routine.

"Interrupts Enabled When Stopped" unchecked

After the user's program is stopped (STOP), the emulator remembers the current Interrupt Enable flag status and disables interrupts. When the program is set back to run, the emulator restores the interrupts (Interrupt Enable flag) back and proceeds with program execution (RUN).

There is no problem when the 'Run' command is being used, but a problem can occur under certain conditions when a single step command is being used.

While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of Interrupt Enable flag and handles it correctly.

For example, interrupts are active and the program is stopped. The emulator remembers the Interrupt Enable flag state and disables interrupts. Now the user executes single steps in the disassembly window and, for example, once the SWI instruction (software interrupt) is stepped. At this moment, the CPU pushes the content of the CCR register to the stack, where the Interrupt Enable flag is stored and jumps to the address where the interrupt vector points to. Before the user's program was stopped (from running), the interrupts were active (Interrupt Enable flag) and after the program was stopped, they were disabled (Interrupt Enable flag) by the emulator. Therefore an incorrect Interrupt Enable flag value (CCR) is now pushed to the stack. Since the emulator can detect such an instruction it modifies the stack with the proper Interrupt Enable value. If this would not be done, the program execution would be changed after RETI instruction in the software interrupt routine is executed. Interrupts in the user's program would now be disabled and not enabled as before while the program was running.

When using step in the source window the above-mentioned problem becomes relevant and the user should never forget it. The source step is actually executed with RUN command with prior setting of breakpoint on the required source line. If SWI (software interrupt) occurs during one source step the CCR with disabled interrupts will be pushed to the stack and after returning from software interrupt routine (RETI) the same value is popped up from the stack. When the user re-runs his program, interrupts are disabled and not enabled, as before the user's program was stopped.

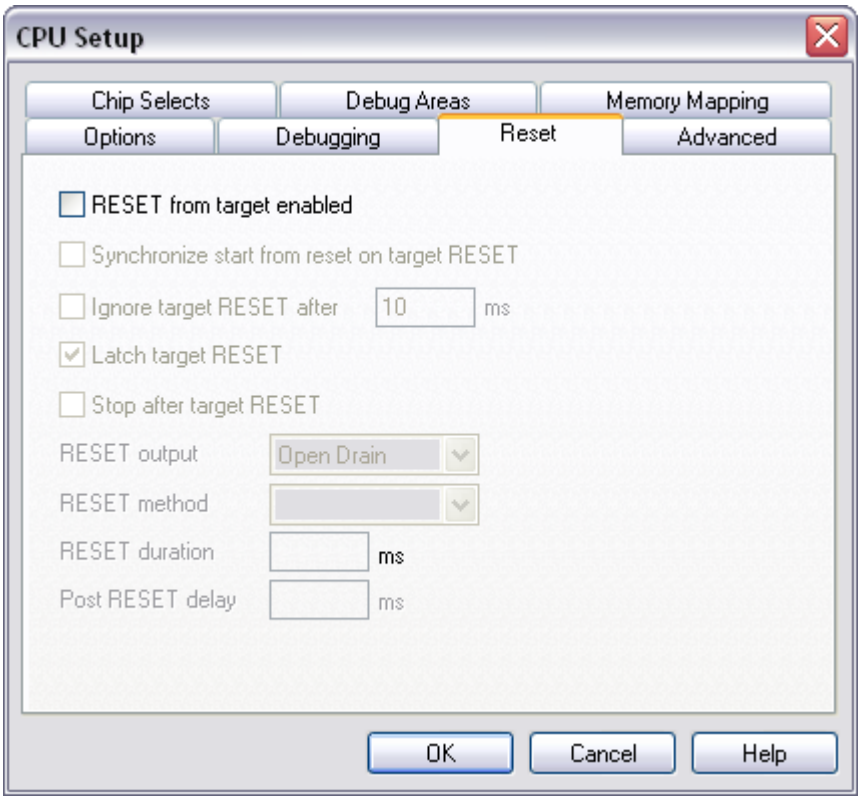
During the source step the emulator cannot detect instructions that changes the state of Interrupt Enable flag as it is the case with single step in the disassembly window.

Hard Interrupt Disable When Stopped

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

3.2 Reset Options

The CPU Setup, Reset 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.

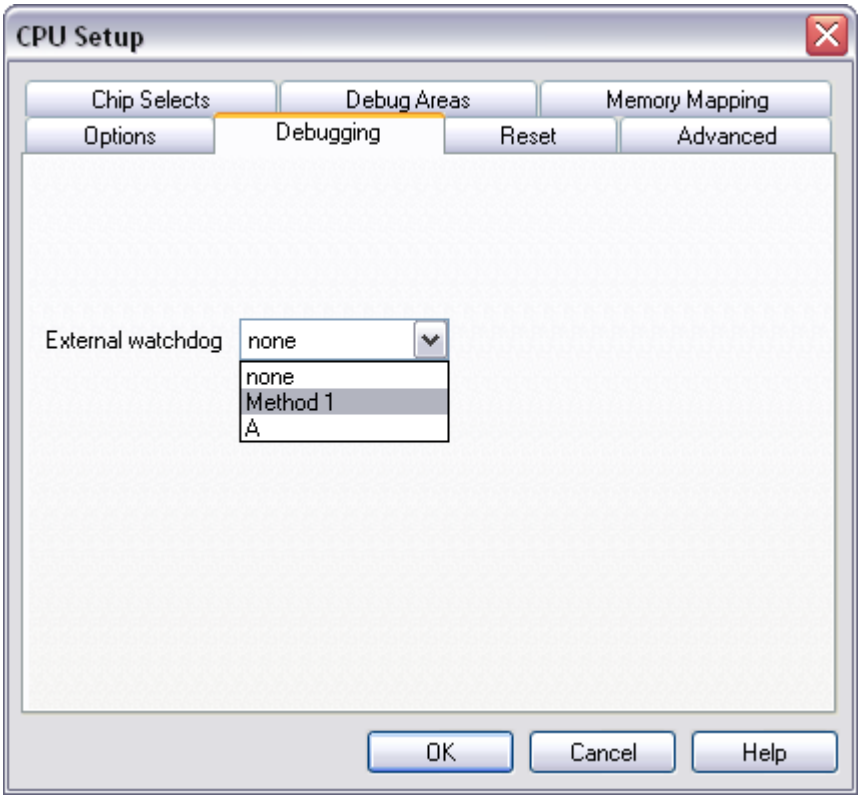


CPU Setup, Options page

RESET From Target Enabled

When checked, the target's RESET line can reset the CPU while the CPU is running.

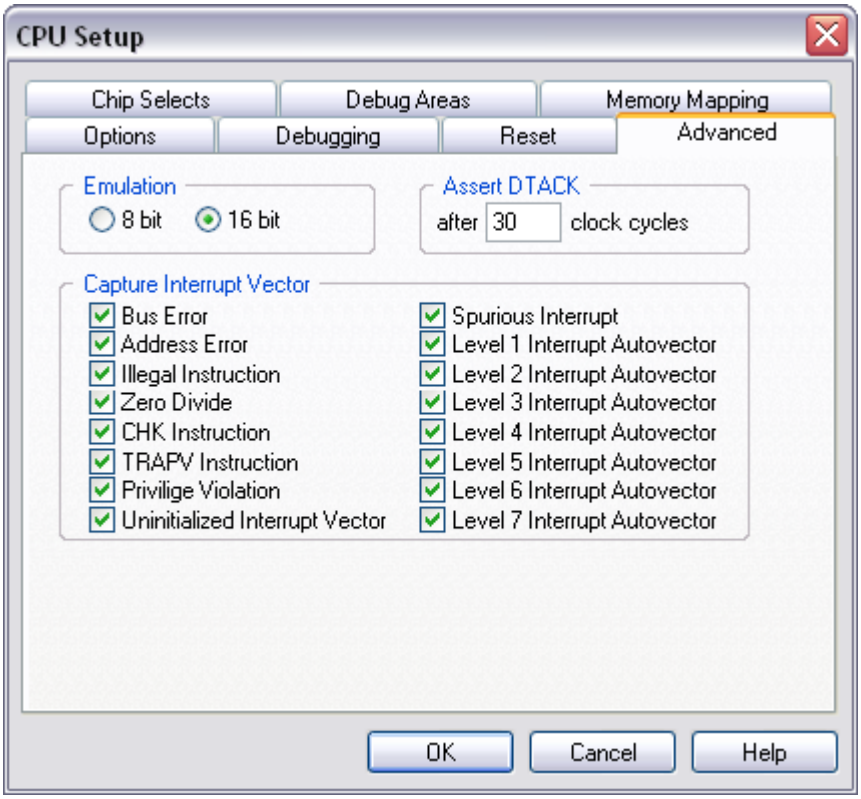
3.3 Debugging Options



Freescale 68302 Debugging Options

Normally, the external watchdog should be disabled in all cases. A special method called Method 1 has been created to allow a specific external watchdog refresh method on some specific targets. Select this option only if you are instructed to do so by your iSYSTEM distributer.

3.4 Advanced Options



Freescale 68302 Advanced Options

Emulation

The Emulator can work in either 8 or 16-bit emulation mode. You must select the mode that you wish to work in.

Assert DTACK

Defines number of clock cycles after, which the Emulator will assert a DTACK (if CPU and other sources fail to do so). The default value should be 256.

Capture Interrupt Vector

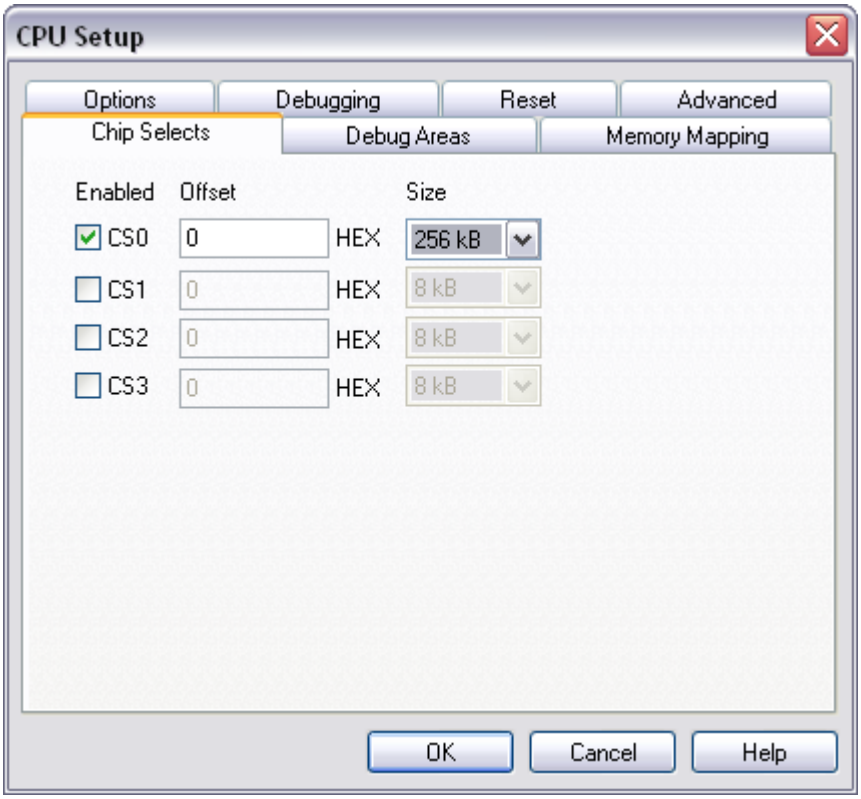
The checked interrupts will instead of executing the associated interrupt routine cause the CPU to stop and a warning message will be displayed.

You can use this feature if a certain exception routine is not implemented yet.

3.5 Chip Selects

The 'Chip Selects' page is available for CPUs with external chip selects.

Note: This option is available for the 68LC302 POD only.



Freescale 68LC302 Chip Selects

Enabled

Check if the chip select is used by the application.

Offset

Specify the offset of the chip select. This is the first address that is covered by the chip select as seen by the CPU core.

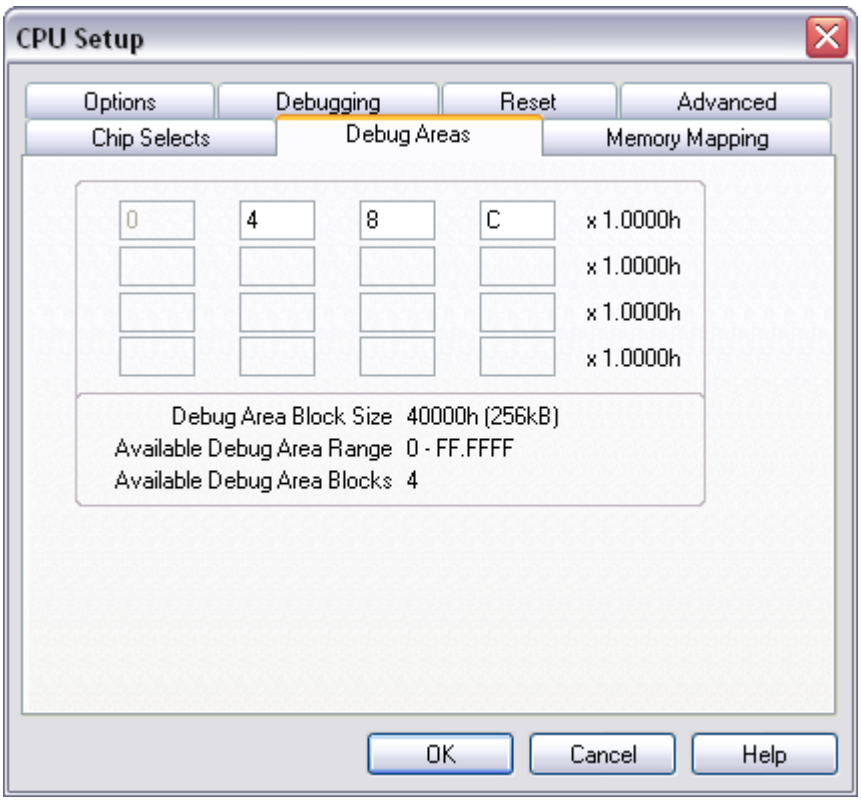
Size

Specify the size of the chip select.

3.6 Debug Areas

The Debug Areas page determines address ranges within, which the Emulator can debug. Depending on the Emulator that you are using and its emulation memory size, different numbers of such areas are available. The size of the debug area block is 128kB.

This setting applies only when the PowerPOD is used. ActivePOD covers the entire 16MB CPU address range.



CPU Setup dialog, Debug Areas page

This page is visible only if the size of emulation memory cannot cover the entire memory addressable by the CPU.

Only a high word part of the address must be entered since debug area blocks can only be moved on multiples of 128k.

Note: Any changes on this page will reset memory mapping page settings.

Large Memory Applications (>1MB)

On large memory applications (determined by the POD type) the debug area block size is 256 KB or 1MB, depending on the emulator used.

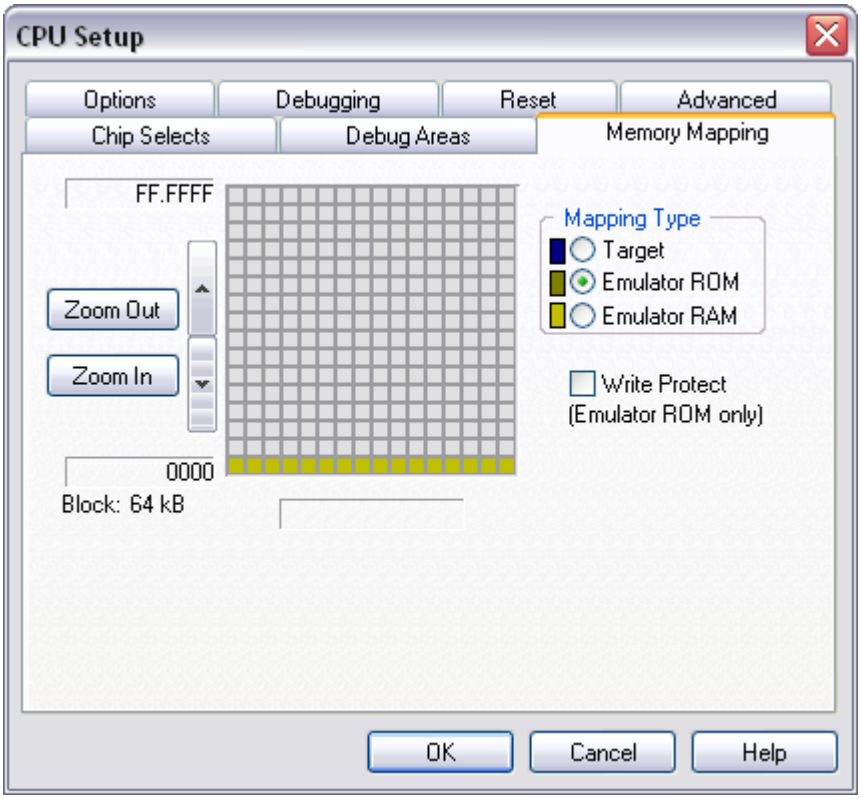
Note: this setting applies only when the PowerPOD is used. The ActivePOD covers the entire address range of 16MB.

Outside debug areas all memory accesses are routed to target. Additionally the following restrictions apply:

- No breakpoints are available
- The trace cannot trigger on addresses outside a debug area
- Shadow memory cannot be used

3.7 Memory Mapping

The mapping page displays currently configured memory mapping.



CPU Setup dialog, Mapping page

Gray blocks in mapping configuration area indicate memory ranges that are either outside the CPU's range (bank systems) or aren't covered by emulation memory.

Colored blocks define current mapping of the covered area:

- dark blue for target
- brown for Emulator ROM - CPU can read from it but not write to it.
- yellow for Emulator RAM - read and write access
- cyan for blocks with mixed mapping - use zoom to view where exactly such blocks map.

To change the mapping type of a block, select desired Mapping Type and click on the block that you wish to map to the select type.

Note: Clicking on a block with mixed mapping, clears all underlying mapping configuration and sets mapping for the entire block to selected mapping.

To configure and view mapping at higher resolution:

- Click the 'Zoom In' button.
- Position the mouse cursor over the block that you wish to zoom in; the mouse cursor will change to indicate zoom mode.
- Click on the block.

You can configure mapping options with 4k resolution on iC181 and 2 bytes resolution on iC1000, iC2000 and iC4000, while the ActivePOD does not provide memory mapping since this is a single-chip CPU. The mapping configuration area always shows a grid of 256 blocks.

In the bottom left corner the current block size is displayed and current ranges are visible to the left of the mapping configuration area. You can zoom in and out and scroll the current range to reach the desired address and resolution.

In general you should configure your mapping as follows:

- where read only devices containing target program are located, set mapping to Emulator ROM. This allows you to download the program quickly without programming EPROMs, while preventing the program from overwriting itself.
- areas occupied by on-chip or off-chip, memory addressable peripherals must always be mapped to target. Otherwise the CPU will not be able to write to them.
- areas occupied by RAM devices can be mapped either to target or to Emulator RAM. You will want to have them mapped to Emulator if the target system is not being used, or when using advanced debugging features like real-time watches.
  Otherwise map them to target.

Write Protect

Prevents the memory, mapped to the Emulator ROM, from being written to. If this option is checked, a write to the Emulator ROM area results in an error message.

Note: This option is available only for the Emulator ROM type of memory.

4 Debugging Interrupt Routines

An interrupt routine can only be debugged when the interrupt source for this routine has been disabled, otherwise you will keep reentering the routine and thus run out of system stack.
For example, there is an interrupt routine with 10 source lines. Let's say that interrupt routine is called periodically by free running timer is an interrupt source. A breakpoint is set on the first source line in the interrupt routine. Program execution stops at the breakpoint. Now source step is executed. Source step is actually executed using RUN command with prior setting of breakpoint on adequate source line. In this particular case, while source step is executed, the CPU executes the code and before source step finishes, new interrupt call occurs. New values are pushed on to the stack and the CPU stops on breakpoint again. If you repeat source steps in such interrupt routine new values are pushed to the stack and you can easily run out of stack.

An interrupt source can be disabled in two ways:

- Disable the interrupt process in the stopped mode. The stopped mode is entered whenever CPU is stopped, and the emulator remains in stopped mode until the Run command is executed. (During Step, Step over, etc. commands, the stopped mode persists).
- Do not place a breakpoint on any instruction in the interrupt routine where interrupts are not yet disabled.
  Also, you must not step over any instruction that re-enables the current interrupt, but run the program before the instruction is executed.

5 Memory Access

68K development tools feature standard monitor memory access, which require user program to be stopped and real-time memory access based on shadow memory, which allows reading the memory while the application is running.

Real-Time Memory Access

Real-time memory access is available on Active POD through the dual ported memory and the PowerEmulator unit with shadow memory. Data area respectively CPU internal RAM area can be read in real-time only.

Real-time write memory access on the PowerPOD is not possible due to shadow memory use. Monitor access must be used to write to the memory.

When the ActivePOD is used, there is no limitation for write access, since real-time write is fully supported.

Monitor Access

When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

Since all accesses are performed using the CPU, all memory available to the CPU can be accessed. The drawback to this method is that memory cannot be accessed while the CPU is running. Stopping the CPU, accessing memory and running the CPU is an option, which, however, affects the real time execution considerably.

The time the CPU is stopped for is relative and cannot be exactly determined. The software has full control over it. It stops the CPU, updates all required windows and sets the CPU back to running. Therefore the time depends on the communication type used, PC's frequency, CPU's clock, number of updated memory locations (memory window, SFR window, watches, variables window), etc.

6 Emulation Notes

Availability of Emulation

Note that 68000 and EC000 core CPU emulation is supported on PowerEmulator revisions F2 and later.

Debug Areas and Memory Mapping

The lowest 128KB of address space must always be mapped since they host interrupt vectors. The first debug area is therefore fixed on address 0.

Mapping type in this area is not important.

Reserved CPU Resources

- CLKO must be set to default (full).
- Bit 15 (TRACE MODE) in the Status Register. User code must leave it cleared.
- CPU's Internal memory space should not be located to bottom of memory space (0).

The following apply only if the 'Interrupts Enabled When Stopped' is selected in the CPU Setup/Options menu:

- 3 word of user stack are used when CPU is stopped
- vectors 10 and 11 (on 28h and 2Ch)
- instructions that throw exception 11 (line 1111 instructions)

Downloading Program to Target RAM

To be able to download the program to target RAM you must configure the appropriate initialization sequence, which properly configures chip-select lines and allows access to target RAM.

After the download is completed, the CPU is reset again and the initialization sequence is not applied.

This ensures that the CPU will start as if ran without Emulator.

Note: Verify after download is legal after chip selects have been initialized by user program code.

68LC302

Following restrictions apply to emulation of the 68LC302

- For every memory area intended to be debugged, a Debug Area must be selected and also Chip Select, which covers this area. Areas not specified are mapped to the target. In case chip selects are swapped by user code(e.g. after reset CS0 = 0, CS1 = 0x100000, after some instructions, CS0=0x100000, CS1 = 0), the Emulator will ignore chip selects and debug areas must not overlap.
  Note that without chip selects the LC302 can only cover 1MByte address space (20 address lines).

Example:

CS0 = 0, CS1 = 0x100000	swapping of CS0 and CS1 not permitted
CS0 = 0, CS1 = 0x180000	swapping of CS0 and CS1 permitted

- Chip Selects must be initialized at the beginning of the program startup code. For proper operation the CFC bits in ORx registers should be cleared (the CFC bit is cleared by Emulator after reset).
- If no CS is generated for the current cycle, the memory request is routed into the first 256KB, no write is possible however.

Note: The 68LC302 POD can operate only with a disabled PLL.

Things to remember

When using 68302 or 68LC302 POD, all MCU's Chip Selects must be defined as read/write. If any Chip Select is marked as read only or as write only, the monitor will not function after the user program stops and the emulation will fail.

If some other external bus master accesses the memory area being emulated, then this area should be mapped to target.

The user's application must not access the following vectors, which are reserved for emulation: 0x24 – Trace, 0x28 – line 1010 of the Emulator, 0x2C – line 1111 of the Emulator. The emulation will fail otherwise, therefore please check this carefully.

Troubleshooting Execution Breakpoints

When reading memory on the breakpoint address a breakpoint instruction is inserted. Since the Emulator does not know whether this is an execution address or a data address (for example reading data when calculating checksum at startup), it always inserts a breakpoint instruction, which results in wrong data read.

Workaround:

- When using execution breakpoints on a CPU with no fetch signal, the breakpoint must always be set to the first byte/word of the instruction. It is recommended that it be set in the source.

In a development stage, such safety checks (like checksum calculation) should not be executed or all breakpoints should be deleted before calculating checksum.

7 Trace

68K development systems offers a featureful trace, which is implemented externally to the microcontroller. 68K Active PODs feature a so called Bus Trace, which additionally offers following features comparing to the 68K Power PODs:

- Enlarged trace buffer
- Duration Tracker
- Watchdog trigger
- Pre/Post Qualifier
- Q between B and C
- Data Change

For more information on these trace functionalities and use refer to winIDEA Contents Help describing Bus Trace in details.

8 Coverage

Refer to winIDEA Contents Help, Coverage Concepts section for Coverage theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Coverage user interface and use.

9 Profiler

Refer to winIDEA Contents Help, Profiler Concepts section for Profiler theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Profiler user interface and use.

Note: Profiler can be configured for profiling data events only. Code cannot be profiled since the results would be wrong due to the microcontroller pipeline. The microcontroller doesn't provide the necessary control signals which would allow the analyzer to distinguish the executed bus cycles from the pre-fetched bus cycles, which are only visible to the analyzer recording hardware.