

---

---

## Technical Notes

# Freescal CPU32 Family In-Circuit Emulation

## Contents

|   |    |
|---|----|
| Contents.....                                     | 1  |
| 1 In-Circuit Emulation introduction .....         | 2  |
| 1.1 Differences from a standard environment ..... | 2  |
| 1.2 Common Guidelines.....                        | 2  |
| 2 Emulation Options.....                          | 3  |
| 2.1 Hardware Options .....                        | 3  |
| 2.2 CPU Configuration .....                       | 4  |
| 2.3 Power Source and Clock .....                  | 5  |
| 2.4 Initialization Sequence .....                 | 6  |
| 2.5 Pattern Generator .....                       | 7  |
| 3 Setting CPU options .....                       | 10 |
| 3.1 CPU Options .....                             | 10 |
| 3.2 Debugging Options .....                       | 12 |
| 3.3 Advanced Options.....                         | 13 |
| 3.4 Chip Selects.....                             | 16 |
| 3.5 AUX Trace.....                                | 17 |
| 3.6 Debug Areas.....                              | 20 |
| 3.7 Memory Mapping .....                          | 21 |
| 4 Debugging Interrupt Routines .....              | 22 |
| 5 Memory Access .....                             | 23 |
| 6 Emulation Notes .....                           | 24 |
| 6.1 Debug Areas and Memory Mapping .....          | 24 |
| 6.2 Reserved CPU Resources.....                   | 24 |
| 7 Troubleshooting Execution Breakpoints .....     | 24 |

# 1 In-Circuit Emulation introduction

## Debug Features

- Unlimited breakpoints
- Real-time access
- Trace
- Execution coverage (with pipeline effects)

### 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.

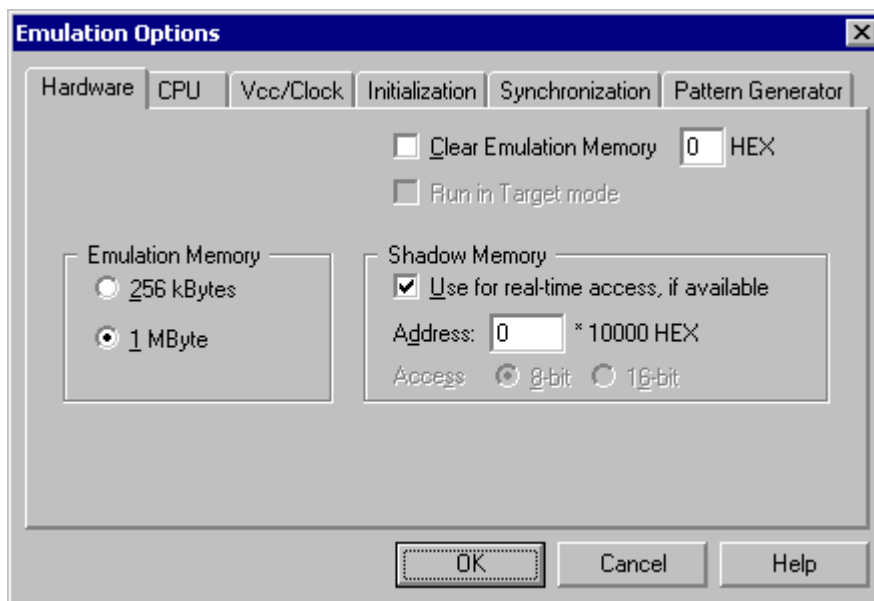
### 1.2 Common Guidelines

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 the emulation memory available on the Power Emulator module. The memory size must be specified correctly, otherwise the emulator fails to initialize.

---

This option is not available for 68332 ActiveGT POD since it features 16MB emulation memory by default.

---

#### ***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 23) will be used for real-time access.

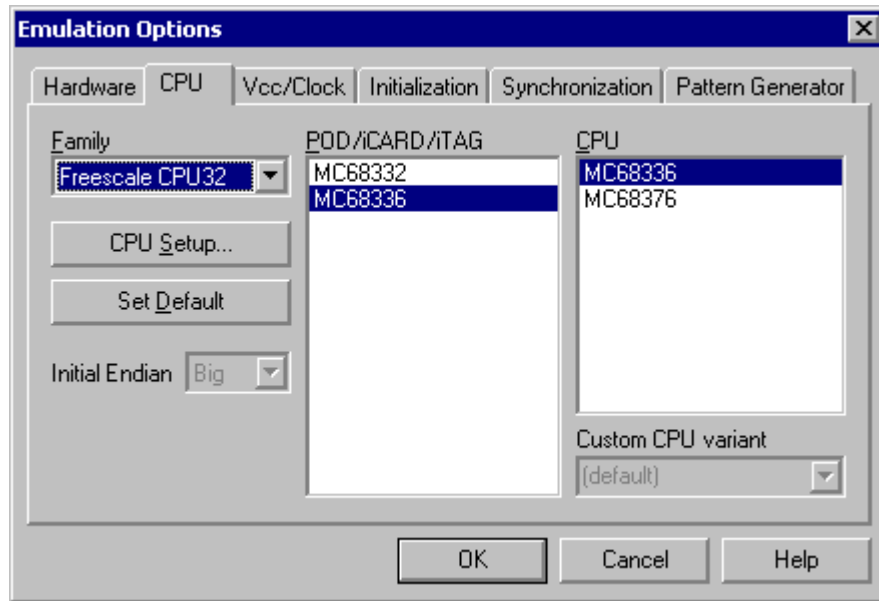
---

This option is not available for 68332 ActiveGT POD where all overlay memory is dual ported. Consequentially, real-time read access is available in complete 16MB address space by default.

---

## 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)

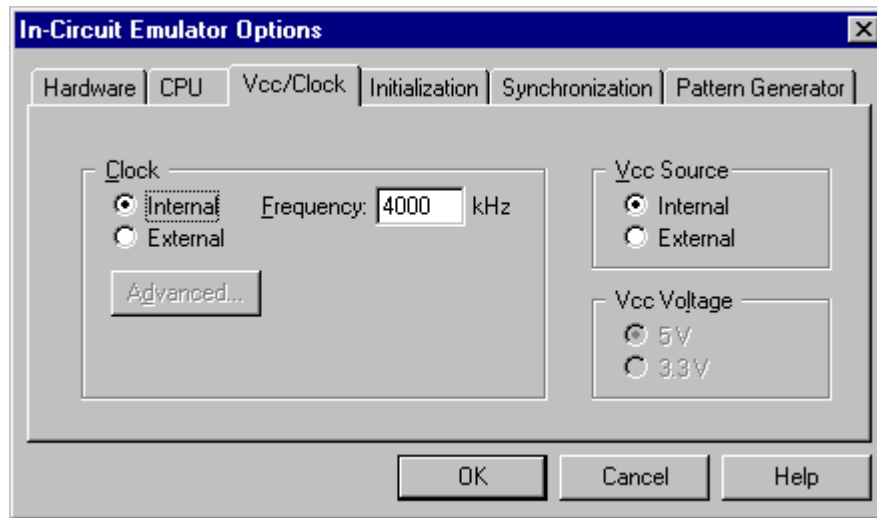
---

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***

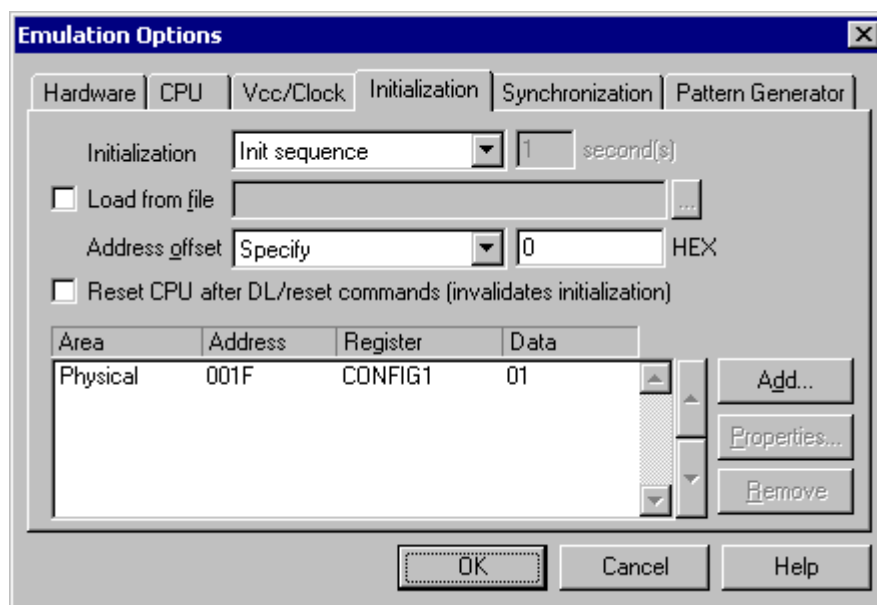
Determines whether Emulator or the target system provides power supply for the CPU.

## 2.4 Initialization Sequence

There is normally no need to use initialization sequence when debugging with an In-Circuit Emulator. 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. Normally there is no need at all to use the initialization sequence in case of the In-Circuit Emulator emulating Single Chip mode. Initialization sequence is required only for some CPU families when it is required by the application being debugged. That can be e.g. either to enable memory access to the CPU internal EEPROM memory or to some external target memory, which is not accessible after the CPU reset. In such case, the debugger executes initialization immediately after reset and then downloads the code. Additionally, the user can also disable CPU internal COP using initialization sequence if there is a need for that, etc.

The initialization sequence can be set up in two ways:

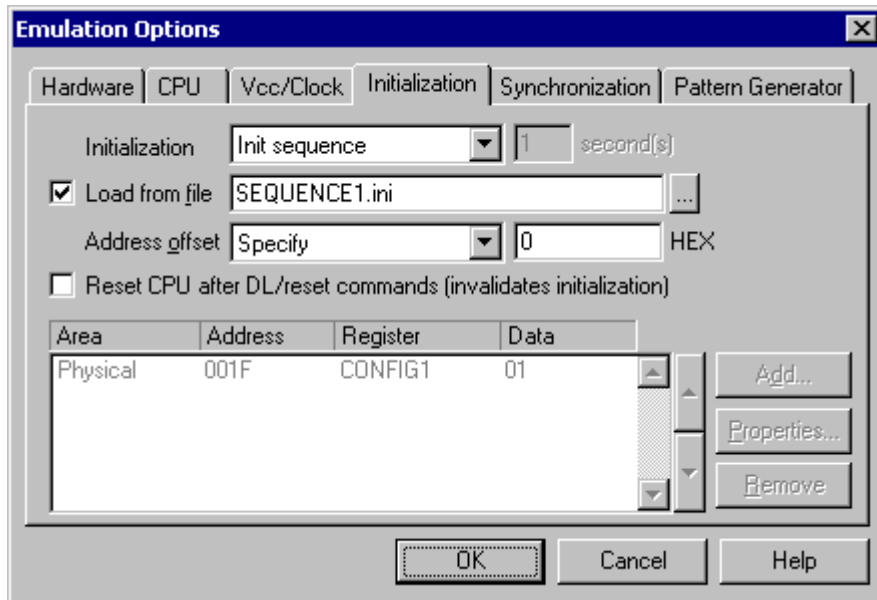
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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

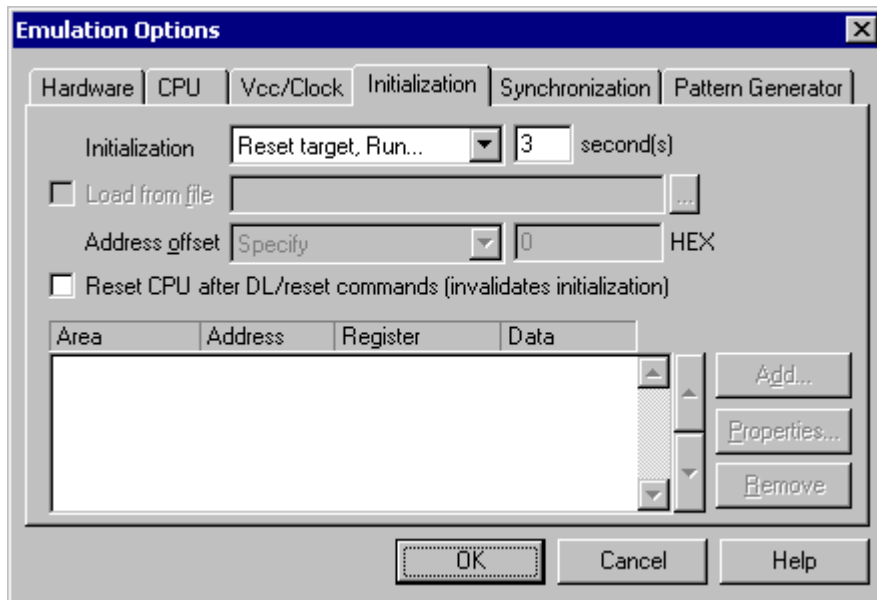
Excerpt from the sample SEQUENCE1.ini file:

```
S PTBD B 12          //comment
S PTBDD B FF
```



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

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



## 2.5 Pattern Generator

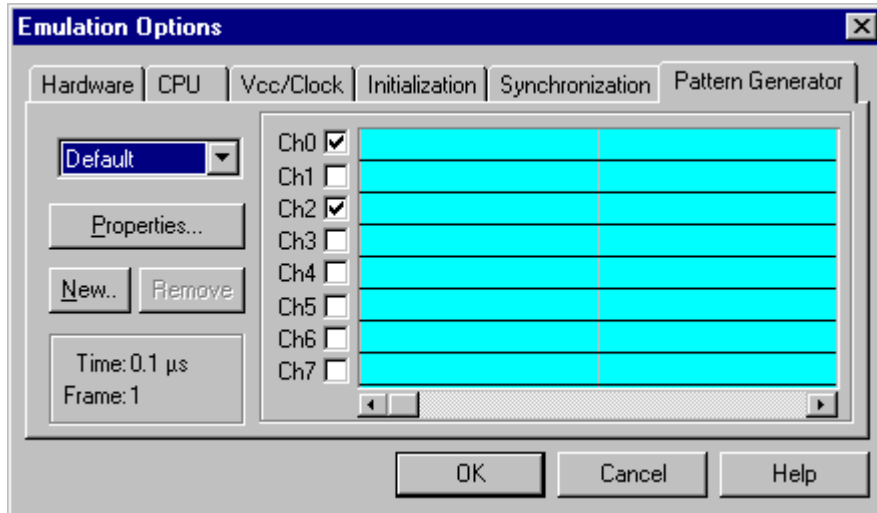
iC2000 Power Emulator module provides an 8-channel waveform programmable pattern generator capable of continuous or single shot operation at up to 10MHz-clock rate with up to 512 samples.

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.



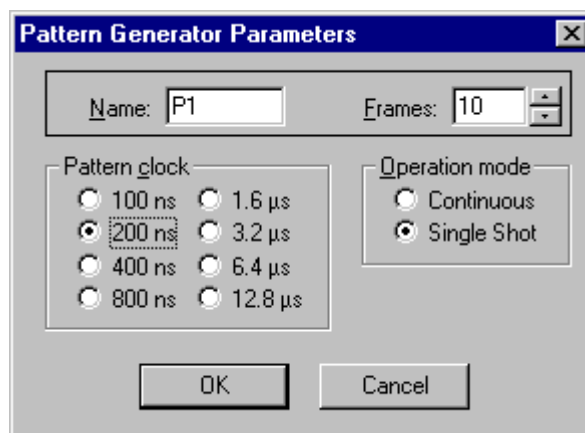
*In-Circuit Emulator Options dialog, iC2000/iC4000 Pattern Generator page*

### **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.

---

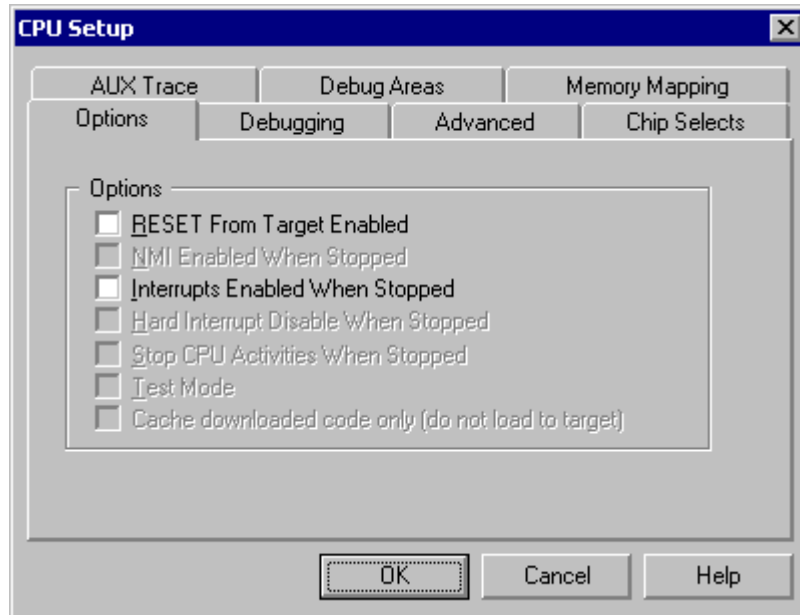
Note: Pattern generator operation can be controlled by an external device through the TRIG/CLKEN pin on the pattern generator connector. Refer to the Hardware User's Manual for more information.

---

## 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*

#### ***RESET From Target Enabled***

When checked, active target's RESET line resets the CPU while the CPU is running.

Note: 68332 Active GT POD has reset from the target enabled unconditionally and has this option is grayed.

#### ***“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.

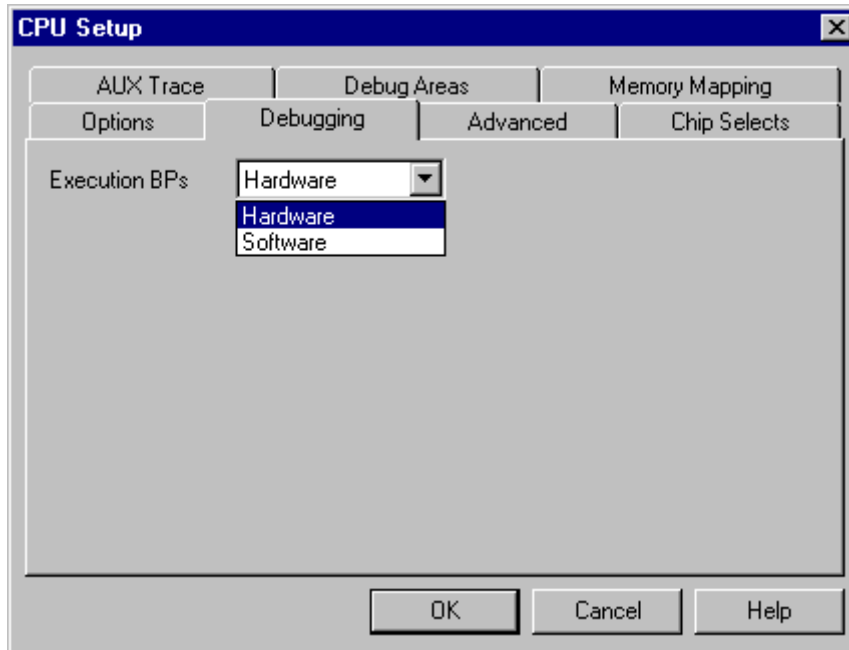
While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of Interrupt Enable flag and handles it correctly.

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.

## 3.2 Debugging Options



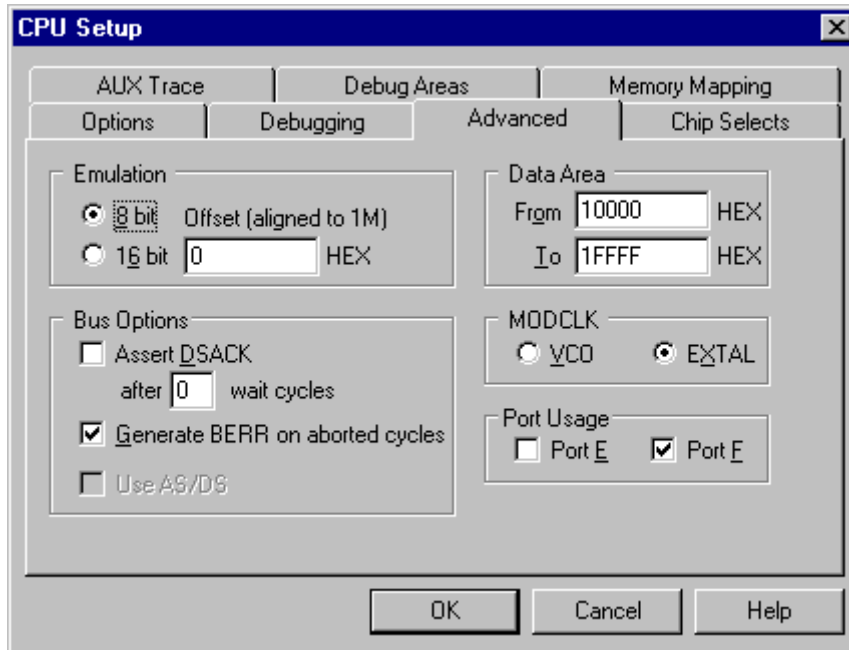
*CPU32 Debugging Options*

### ***Execution Breakpoints***

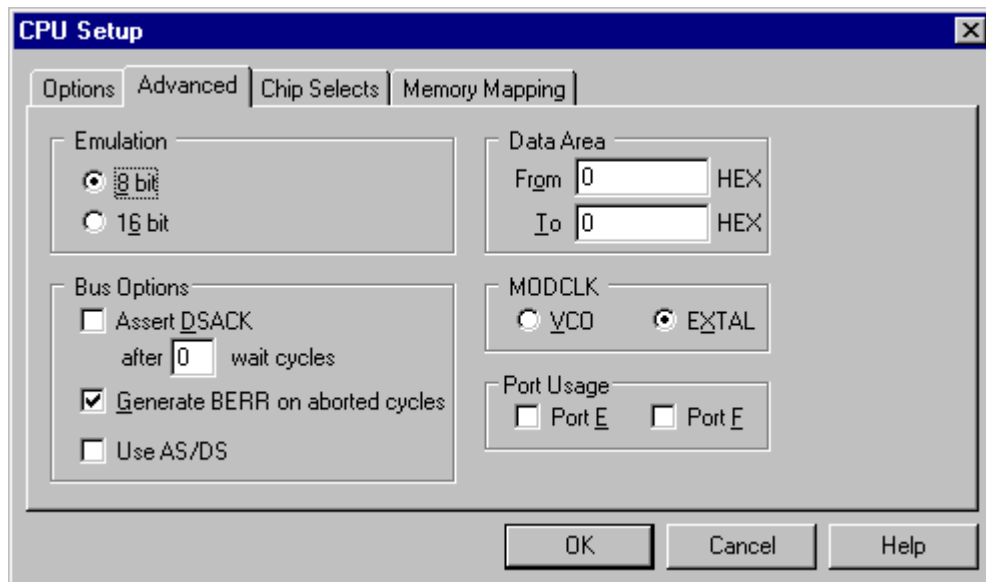
Hardware breakpoints are breakpoints that are already provided by the hardware (the CPU). The number of hardware breakpoints is limited to two, but in some cases they are the only option. This is especially the case with FLASH memory, non-writeable memory (ROM) or self-modifying code. If the option 'Use hardware breakpoints' is selected, only hardware breakpoints will be used, software breakpoints will be disabled.

The number of software breakpoints is not limited. 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 this fails (for example, the memory area in, which the breakpoint should occur is not writeable), a hardware breakpoint will be used instead.

### 3.3 Advanced Options



68332 and 68332 II Power POD



68332 ActiveGT POD

#### Emulation

- 68332 and 68332 II Power POD

8-bit and 16-bit memory devices can be connected to the target CPU. With the emulator, every chip select can be individually mapped to the target or to the emulator. The emulator can emulate only 8-bit or 16-bit devices at one time. This means that any number of 8-bit devices or any number of 16-bit devices can be emulated, but not the mix of 8- and 16-bit devices. Hence, the user can map to the emulator only either 8-bit or 16-bit memory devices. Other devices must be mapped to the target.

This setting defines a bus width of the memory devices mapped to the emulator. It does not affect resources (memory, chip selects), which are mapped to the target. Hence the user is free to use 8-bit target devices, while emulating 16-bit devices, and vice versa.

#### Offset

Determines the start address of the debug area block, which can be only a multiple of 1M. Typically the application is debugged below 1M, therefore the offset is 0. Within the debug area block you can move available debug areas in the 'Debug Areas' tab depending on the emulation memory size, e.g. if the 256k Power Emulator is used and the application code should be debugged on address 0x280000 (2M + 512k), then the offset must be set to 0x200000. Additionally, two available 128kBytes debug areas should be moved to 0x80000. 0x8 and 0xA values must be written in the 'Debug Areas' tab. Addresses in the 'Debug Areas' tab are always relative to the offset and cannot be greater than 1M since the debug block size is 1M. If one 128kByte area should be at 0x200000 and the other one at 0x280000, then the user should write 0x0 and 0x8 in the 'Debug Areas' tab. Note that all debug areas reside within a single debug area block.

- 68332 ActiveGT POD

As long as the CPU addresses memory via chip selects or as long as the memory is mapped to the target, this setting is irrelevant. With the 68332 ActiveGT POD, 8-bit and 16-bit memory devices can be emulated at the same time when mapped to the emulator. The bus width of the memory devices mapped to the emulator must be configured properly for every chip select line in the 'Chip Selects' tab.

When the emulated memory device does not connect to the CPU chip select line but rather address decoding is implemented in the target instead, the user must set accordingly '8 bit' or '16 bit', depending on the memory device bus width. Make sure that 'Use AS/DS' option is checked in the Bus Options field in such case. If there are 8-bit and 16-bit target memory devices with address decoding (no CPU chip select lines connected) in the target, only 8-bit (16-bit devices must be mapped to the target) or 16-bit (8-bit devices must be mapped to the target) devices can be emulated at a time.

### ***Bus Options***

#### *Assert DSACK*

If a CPU DSACK signal is expected from the target, but is not generated due to some problem, the emulator can assert the DSACK after a specified number of wait cycles, hence finishing the CPU cycle. Type of DSACK cycle (8-bit or 16-bit) generated from the emulator, depends on the setting in the Emulation field.

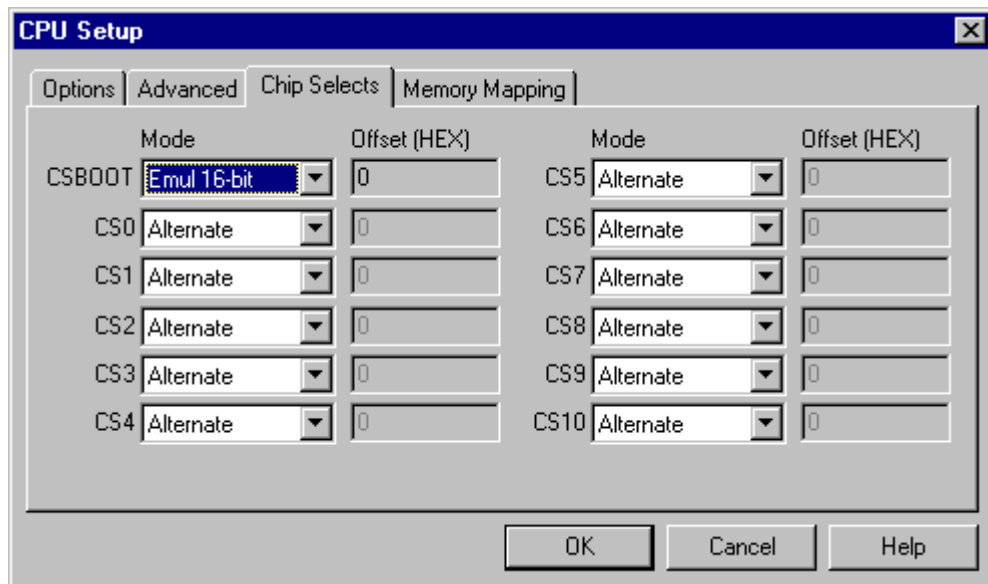
#### *Generate BERR on aborted cycles*

This option can be used if the target hardware does not generate one. This is the only way to finish a memory cycle, if the CPU did not receive the acknowledge for it.

#### *Use AS/DS (available on 68332 ActiveGT POD only)*

Use this option when it's required to emulate target memory devices, which don't connect directly to the CPU chip-select lines but have address decoder implemented in the target.

For instance, below screenshot shows 'Chip Selects' tab configuration of a target where the CPU boot chip select line connects to one memory device while other memory devices don't connect to any chip select line but connect to the CPU address lines. Then the target itself must feature some decoding logic in order for all the memory being accessible by the CPU.



### ***MODCLK configuration***

With this setting the clock source is selected. The options are:

- VCO
- EXTAL

Depending on whether internal VCO is used, the proper option should be checked. If VCO is used note that the external clock source has a maximum frequency, which is not the same as the MCUs maximal operating frequency. When using VCO, the clock connected to the EXTAL pin is multiplied internally to generate the MCUs system clock.

A typical application using the VCO uses a 32.768 kHz crystal. In such case the internal Emulator clock source cannot be used since its minimum frequency is 1MHz.

### ***Port Usage***

If you use ports E and F as I/O ports, check the option. If alternate functions on the port are used, uncheck the appropriate checkbox.

### ***Data Area***

The Data Area setting is required for correct trace analysis. You should specify the memory range of the RAM device in the target.

### 3.4 Chip Selects

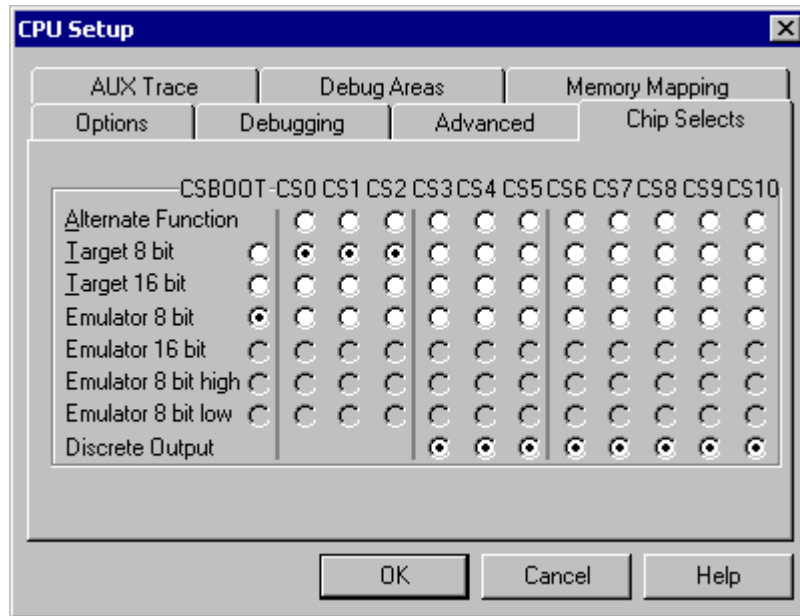
In the 'Chip-Selects' tab the user must define the configuration of certain CPU lines in his application. Some physical lines can operate as chip select, discrete output or can have an alternate functionality (e.g. address line). When used as chip select, the user must also define the bus width (8 bit or 16 bit) where it's active and the mapping type (emulator or target).

For every chip-select signal of the CSU unit, you must specify its usage, as it is used in the target system (or the emulation can fail). Depending on the Emulation setting and chip-select itself, different options are available.

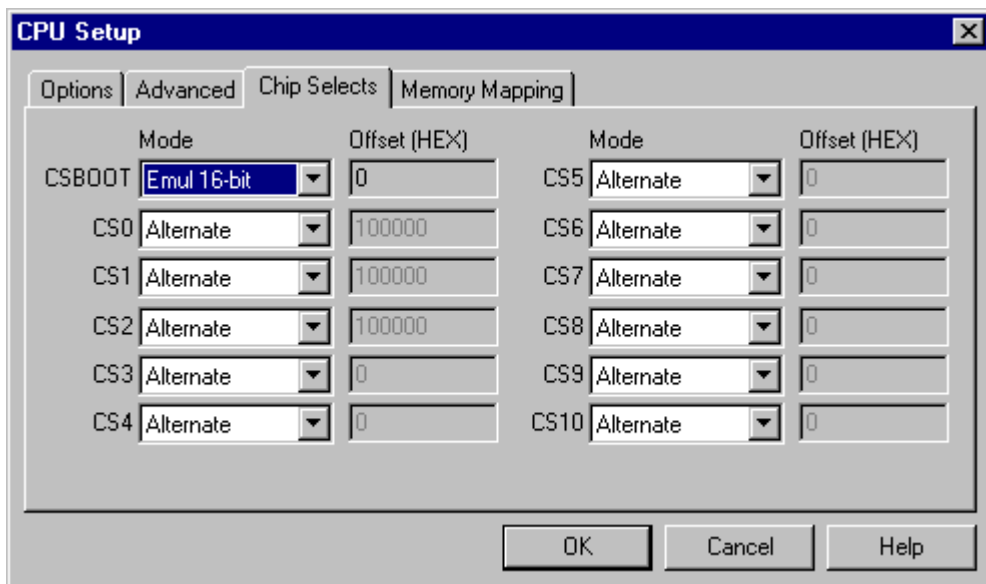
---

Note: Changing usage of a chip select can affect the type of other chip selects.

---



68332 and 68332 II Power POD



68332 ActiveGT POD

The total amount of memory covered by chip-selects mapped to Emulator must not exceed the size of emulation memory. In case the emulation memory cannot cover the entire memory, some chip-selects must be mapped to target. The chip-select areas should not overlap.

Some CPU32 applications use more memory than the emulator covers. In such case the application cannot be fully debugged in all used memory space. This is not a drawback, since in such application typically the code resides in the memory not larger than 1MByte, which can be fully covered by the emulator. The remaining memory is used for data and hence can be mapped to target without losing any crucial debug capabilities. In the debug area all standard debug capabilities like setting breakpoints and high-level source debugging are supported. Additionally, memory can be mapped as 'emulation ROM', 'emulation RAM' or as target. Memory areas outside of the debug boundaries are mapped to the target by default, and only run, single step and stop debug commands are available on the disassembly level. Used debug areas must be defined in the 'Debug areas' tab and required mapping for associated debug areas can be set in the 'Memory Mapping' tab.

Example:

- CSBOOT covers 1MB of memory from 000000h.
- CS0 covers 1MB of memory from 200000h.

You have two possibilities:

- Set CSBOOT to Emulator, emulation offset to 0h and CS0 to target.
- Or, set CSBOOT to target, CS0 to Emulator and emulation offset to 200000h.

### **3.5 AUX Trace**

---

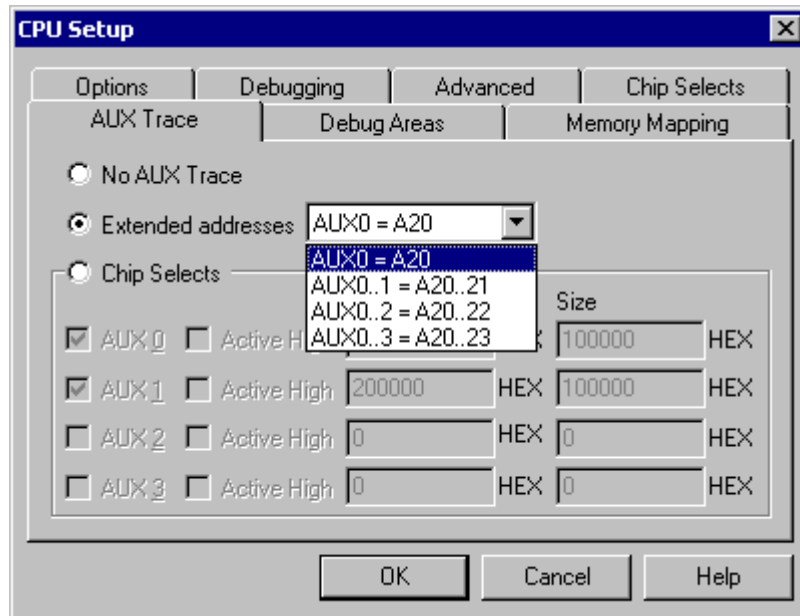
This configuration tab is not available on 68332 ActiveGT POD where 16MB emulation memory is available.

---

By default the PowerEmulator together with the trace module can trace only within the 1M area that resides on the debug area boundaries – default setting 'No AUX trace'. Since this can be a limitation when emulating 68332, a special feature was implemented, which extends the traceable memory area: the CPU chip-select or address lines can be connected to the trace AUX inputs,. Trace Aux lines are available on the trace module and first four trace Aux lines also on the POD.

#### ***AUX inputs used as additional address lines***

With this option, the trace range can be extended up to 16MByte.



68332 Aux Trace dialog

First, the number of AUX lines used as additional address lines must be specified. The AUX lines always assume one address line – this way AUX0 always assumes A20, AUX1 can assume A21, and so on. With more memory lines connected, the larger memory range can be traced.

### ***Chip-Select lines connected to AUX inputs***

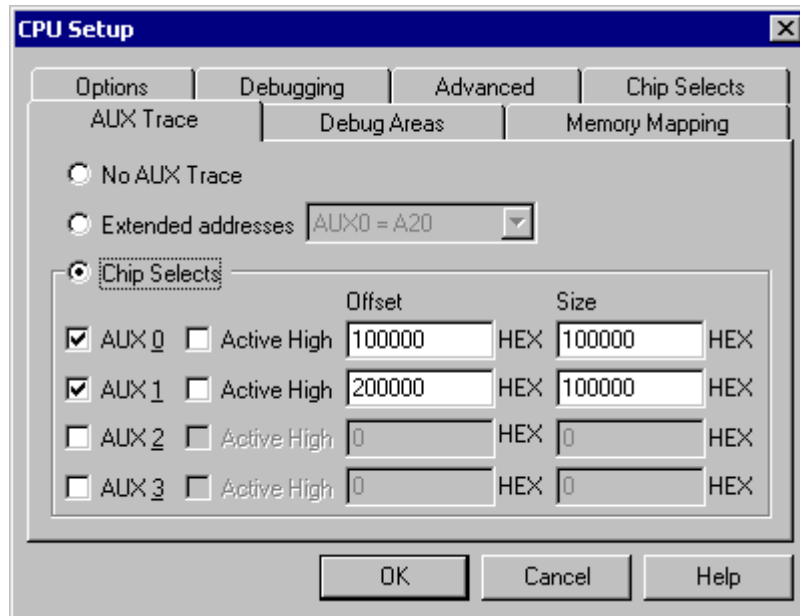
With this option, the trace range can be extended up to 5MByte. If no chip-select (CS) lines are connected to AUX inputs, only the area of 0x0-0x100000 can be traced. For all other debug boundaries, CS lines must be connected to the AUX inputs.

For example:

CSBOOT is active from 0x0 to 0x100000. The 1M debug area is located on 0x0 as well. Trace can record this area by default. Additionally the target uses CS1 and CS2. CS1 is active from 0x100000 to 0x200000 and CS2 is active from 0x200000 to 0x300000.

To trace execution in the CS1 and CS2 areas, the user must connect CS1 to the AUX0 input and CS2 to the AUX1 input. AUX0 and AUX1 are present on the front panel of the Analyzer module. In general a maximum four additional lines can be connected (AUX0-3).

In the 'AUX Trace' tabs CS0 and CS1 must be activated as additional trace address lines.



68332 Aux Trace dialog

Now the trace records all execution within 0x0 and 0x300000. When a program is executed on the address 0x210045, the CS2 is active and recorded by trace. In the trace window AUX1 (CS2) is added internally to the address bus and the proper address value is displayed – 0x210045. All source information is displayed correctly as well. Vice versa, trigger can be set on the 0x210045. The trace will use AUX1 internally to set the proper trigger condition.

---

Note: All additional lines (chip-selects or address) connected from the target to the AUX inputs must be delayed for ~20ns. Signals A0-A19 are connected to the trace buffer via Xilinx FPGA and therefore they are delayed. Signals from AUX inputs are connected directly to the trace buffer and not via the FPGA. Consequentially they must be delayed for proper recording. In case of IC30100 POD use two inverters (e.g. HCT14) for 20ns delay on each connected AUX input or in case of MC68332A POD (IC30102) you can delay these lines with the POD. 20ns delay is implemented between 'Input line x' and 'Output line x', which are available at connector P2.

---

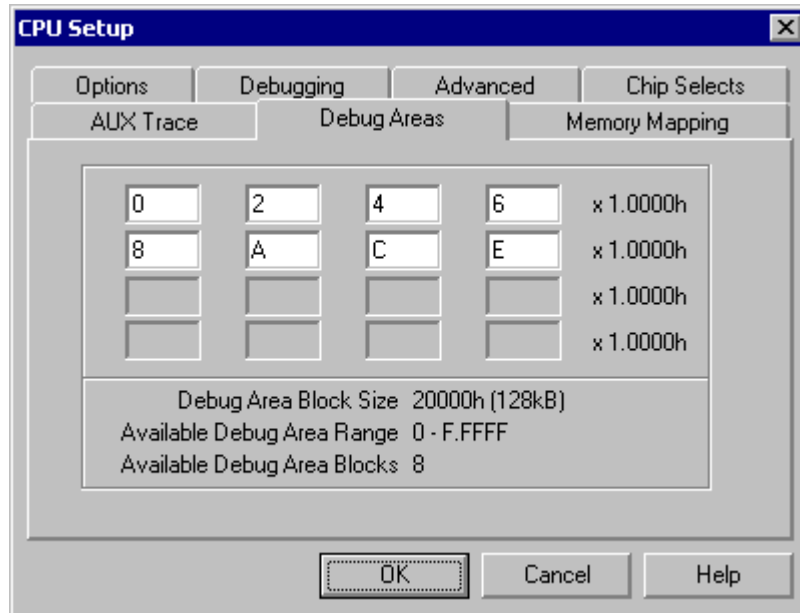
### 3.6 Debug Areas

---

This configuration tab is not available on 68332 ActiveGT POD where 16MB emulation memory is available.

---

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.



*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 change on this page resets memory mapping page settings.

---

#### Large Memory Applications (>1MB)

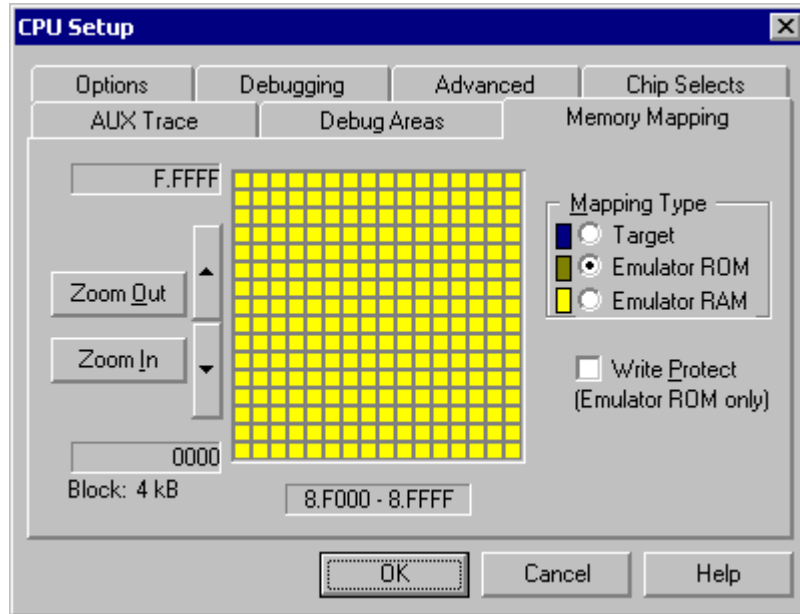
On large memory applications (determined by the POD type) the debug area block size is 256 KB or 1MB, depending on the emulator used.

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

CPU32 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 PowerEmulator unit with shadow memory and 68332 ActiveGT POD. Data area can only be read in real-time.

---

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

---

There is an alternative solution to shadow memory. The debugger can access debug memory almost in real time using debug monitor, however only when BREQ is not used by the application. A variable read takes approximately 30 us for first byte and 4-5 us for following bytes. Note that CPU internal memory cannot be accessed using this method since it's limited to debug (ICE overlay) memory.

### 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

### 6.1 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.

### 6.2 Reserved CPU Resources

No CPU resources are used.

## 7 Troubleshooting Execution Breakpoints

When reading memory, on the breakpoint address a breakpoint instruction is inserted. Since the Emulator doesn't 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.

---

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.