
Technical Notes**Freescale CPU32 Family On-Chip Emulation****Contents**

Contents.....	1
1 Introduction	2
2 Emulation options.....	3
2.1 Hardware Options	3
3 CPU Setup	4
3.1 General Options	4
3.2 Advanced Options.....	5
4 Emulation Notes	5
4.1 Enabling the CPU's Bus Monitor.....	5
4.2 FLASH Programming	6
4.3 Things to remember	6
5 Getting Started.....	6
6 Troubleshooting.....	7

1 Introduction

The CPU32 family features a special debug module (BDM), which enables the access to CPU resources when active. The access to memory and to the registers is available through BDM.

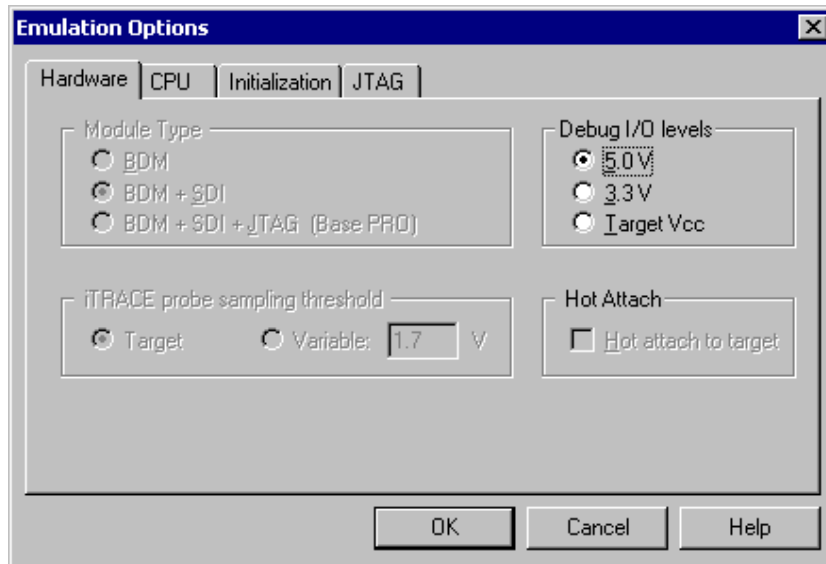
Considering that this is a pretty old implementation of on-chip debug module, it has many limitations comparing to the on-chip debug modules, implemented in the latest CPUs. To debug the application, the code must be loaded in the RAM type of memory where software breakpoints can be used. Thereby, in case of a target ROM, it has to be replaced either directly by RAM memory or by use of ROM emulator, which connects to the target instead of the ROM device.

Debug Features

- No on-chip breakpoints
- Unlimited software breakpoints
- No real-time access
- Fast FLASH programming

2 Emulation options

2.1 Hardware Options



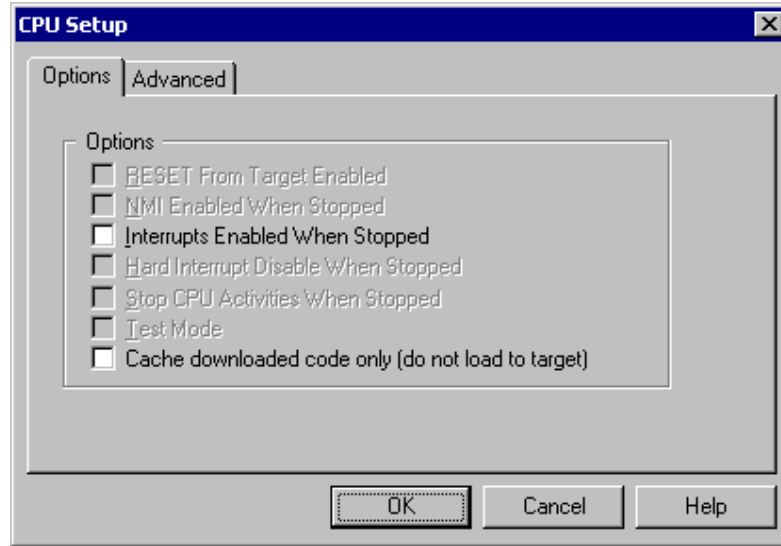
Emulation options, Hardware pane

Debug I/O levels

The development system can be configured in a way that debug (BDM/JTAG) signals are driven at 3.3V, 5V or target voltage levels. When 'Target Vcc' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin (target debug connector) is used as a reference voltage for driving debug (BDM/JTAG) signals. Make sure that the target reference voltage pin is connected when 'Target Vcc' Debug I/O level is selected

3 CPU Setup

3.1 General Options



General Debugging Options

Interrupts Enabled When Stopped

On-chip debug module itself doesn't support servicing interrupts while the application is stopped (interrupts in background). Setting of this option impacts only on the CPU behavior during single step.

Disabling this option makes the Emulator mask the interrupts between a debug step command, which normally results in more predictive behavior of applications using interrupts. This is a default setting.

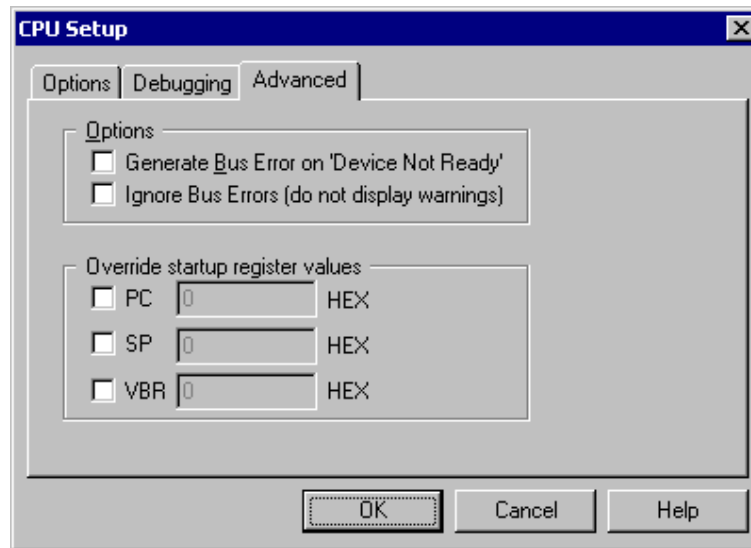
If this option is enabled, the Emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

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.

3.2 Advanced Options



CPU32 Family Advanced Options

Generate Bus Error on 'Device Not Ready'

Select this option if your target contains no external logic that would generate a BERR.

Ignore Bus Errors

Bus Errors are ignored if this option is checked.

Override startup register values

This option overrides the default Program Counter, Stack Pointer and Vector Base Register reset values with the values set.

4 Emulation Notes

4.1 Enabling the CPU's Bus Monitor

When a memory area, not covered by a valid memory device is accessed, a 'Bus error' exception occurs. Normally this is an indication that the program has performed an illegal memory access due to a bug in the software, where a CPU "hang" (if no bus error logic is used) is justified. But the BDM Emulator itself can provoke a "Bus Error" if an illegal memory access is attempted from the IDE (such as opening a memory window on a wrong location).

You can circumvent this by configuring the CPU's Bus Monitor to trap such accesses. This is best done by using the CPU initialization sequence (see above), or, if you prefer, by modifying the target program. The Bus Monitor is activated by

- setting the BME bit in the SYPCR register (FFFA21 on 68332).
- clearing the FRZBM bit in the SIMCR register (FFFA00 on 68332).

Refer to your CPU manual for physical addresses of these CPU registers.

4.2 FLASH Programming

When programming external FLASH 'through FLASH monitor', some things should be considered. It is always recommended to allocate the FLASH programming monitor in the CPU internal RAM. Following instructions should be followed when monitor is located in the internal RAM:

- watchdog must be disabled, or the monitor won't work
- do not initialize any units, e.g. TPU, serial channels,... or the internal RAM will be assigned to them
- the following initialization sequence should be used:

RAMBAH 0x0080

RAMMCR 0x0000

SYPCR 0x000C

Pay attention that chip selects must not overlap each other.

In case if "FLASH programming" monitor is allocated in the external RAM, it is recommended that the RAM is allocated below the FLASH memory.

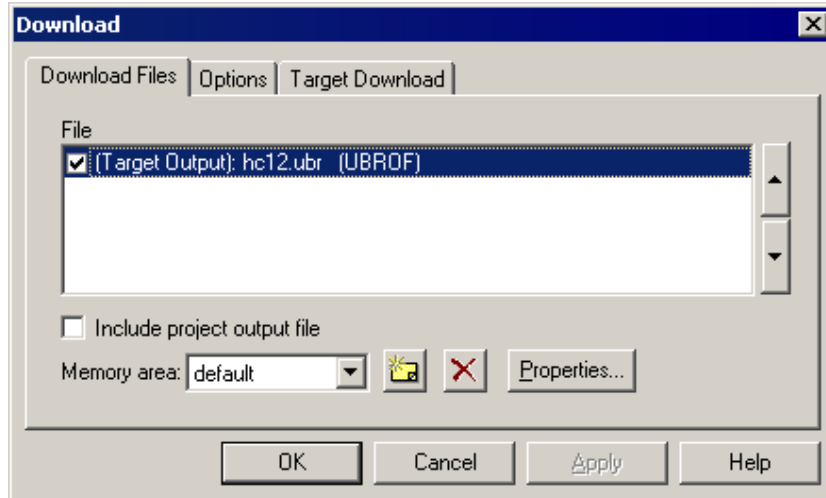
4.3 Things to remember

- If you accidentally attempt to access memory regions that are not covered by any device in the target system with a watch, memory window, etc., a Bus Error (BERR) will occur. If you do not supply Bus Error logic, or enable the CPU's Bus Monitor to handle such errors, the target CPU will hang until it is reset.
- When initializing a system with a 68360 target, you must enter the proper initialization sequence. The reset vector must be available immediately after reset. The first sequence in the initialization sequence must be a write to the MBAR register. With the MBAR offset all SFRs are defined. The same offset must be written in the 'Address offset' field in the 'Initialization' tab. The offset written in the MBAR register and 'Address offset' must always match or the initialization sequence will not work properly. You can write to the MBAR only immediately after reset. Later you cannot/must not read and write it at all, because it is locked.

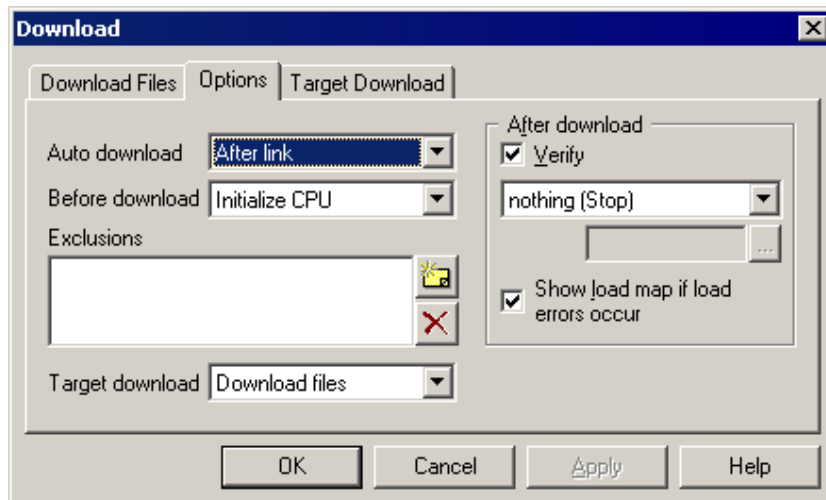
5 Getting Started

- 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 the necessary 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 the reset location.
- 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. Now you may add the download file and load the code to the target RAM.

- 8) To program the flash or download the code to the target RAM, which is not accessible after reset, make sure you use the initialization sequence to enable the access. First, the debugger executes reset, then the initialization sequence and finally the download or flash programming is carried out.
- 9) Code is loaded to the target RAM through the so-called target download. Add the file(s) to be downloaded in the 'Download Files' tab in the 'Download' dialog.



Select 'Download files' from the 'Target download' combo box.



- 10) Execute Download debug command, which downloads the code to the target RAM. There should be no verify errors in case of a successful download.

6 Troubleshooting

- If BDM does start (system is initialized), but system cannot perform run or single step check BDM DSI line - there shouldn't be any strong pull-up - remove it and try again.
- Should the error message "device not ready at the address 0xFFFFFFE" occur, do the following:

First, close all debug windows, except the disassembly window and try again. If the problem disappeared, a debug window, requesting invalid memory access, was opened. For instance, accessing of memory location, where no chip select is active or physically no memory allocated, is an illegal operation, therefore bus error occurs.

Normally, after the download, the debugger stops the CPU at the address where the reset vector points to. In this particular case, either reset vector is not accessible or the location where the reset vector points. Check where the reset vector is allocated. Check if there is memory available at the location where the reset vector points to. Check whether a chip-select signal is active at the address where the reset vector points.

While trying to solve the problem, make sure, you've checked the "Generate bus error on 'device not ready'" option in the 'CPU Setup/Advanced' dialog. If the option "Generate bus error on 'device not ready'" is not checked and a bus error occurs, the system cannot initialize at all. Therefore, the debugger is useless in such condition.

If the option is checked, the error message "device not ready at the address 0xFFFFFEE" still occurs, but the system gets initialized. The user can inspect the reset vector location (in case of 68332 CPU at address 4h) and the SFR registers. In this particular case, probably a reset vector contains 0xFF values and consequentially the PC points to a location above 1MB memory space. Note that a valid reset vector after reset points to the memory location below 1MB space and that after reset only CSBOOT is active up to 1MByte. Verify your application.

Pay attention to the address, which is part of the error message. This is the address being problematic.

- Make sure that the power supply is applied to the target BDM connector when 'Target VCC' is selected for Debug I/O levels in the Hardware/Emulator Options/Hardware tab, otherwise emulation fails or may behave unpredictably.
- 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.