
Technical Notes

Infineon XC800 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
4 Internal FLASH Programming	9
5 Access Breakpoints	9
6 Getting Started.....	10
7 Troubleshooting.....	10

1 Introduction

Infineon XC800 family is based on high-performance 8-bit microcontroller, which is compatible with the industry standard 8051 processor. While the standard 8051 core is designed around a 12-clock machine cycle, the XC800 core uses a two-clock period machine cycle.

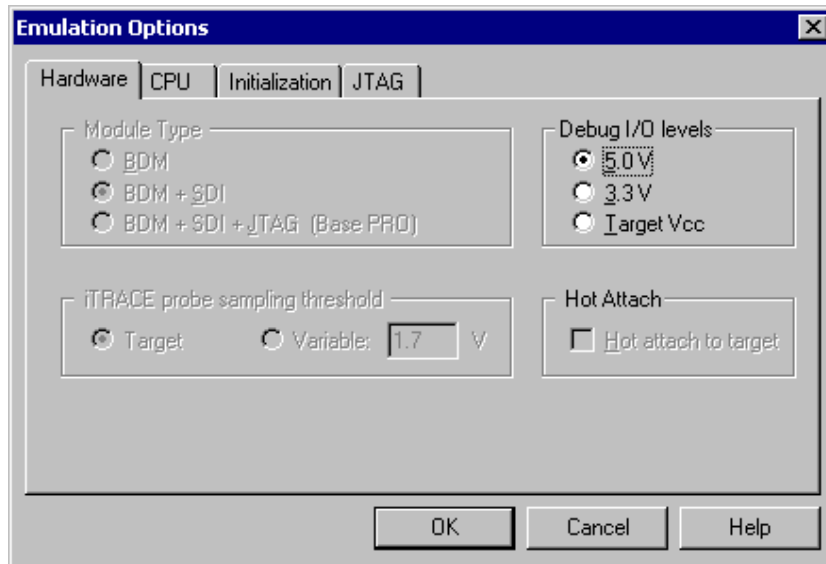
The debugger communicates with the on-chip debug module through the standard JTAG interface. The on-chip debugging functions include execution control (run, single step, stop, set breakpoint), device control (read/write registers, reset, clock configuration, device configuration), and low-level access to the advanced features.

Debug Features

- Up to 4 hardware breakpoints
- Unlimited software breakpoints
- Access breakpoints
- No Real-time access
- Fast flash programming

2 Emulation options

2.1 Hardware Options



Emulation options, Hardware pane

Debug I/O levels

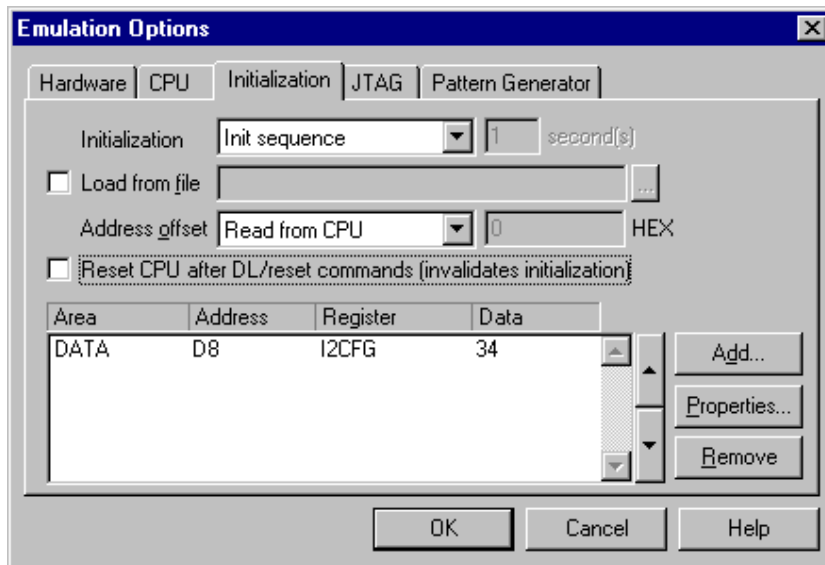
The development system can be configured in a way that debug (BDM/JTAG) signals are driven at 3.3V, 5V or target voltage levels. When 'Target Vcc' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin (target debug connector) is used as a reference voltage for driving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

2.2 Initialization Sequence

Before the flash programming or download can take place, the user must ensure that the memory is accessible. This is very important since there are many applications using memory resources (e.g. external RAM, external flash), which are not accessible after the CPU reset. In that case, the debugger must execute after the CPU reset a so called initialization sequence, which configures necessary CPU chip selects and then the download or flash programming can actually take place. The user must set up the initialization sequence based on his application.

The initialization sequence can be set up in two ways:

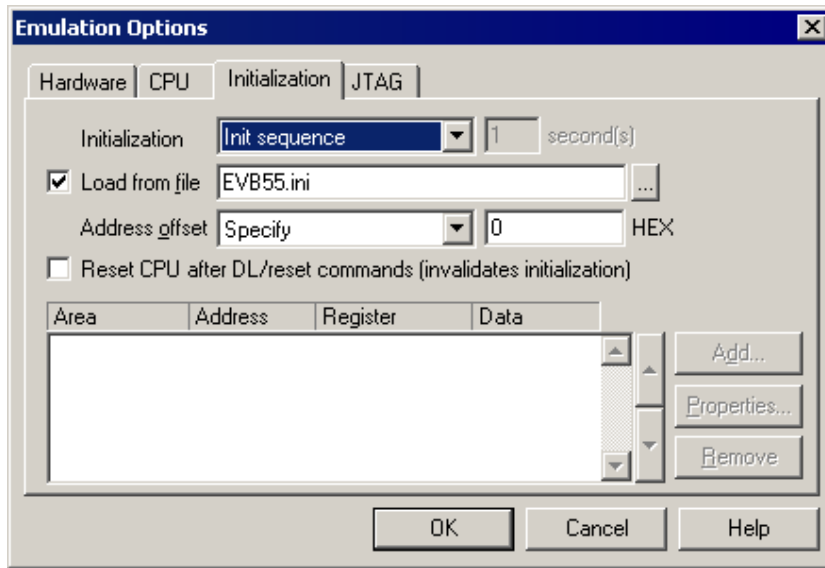
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



2. winIDEA accepts initialization sequence 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.

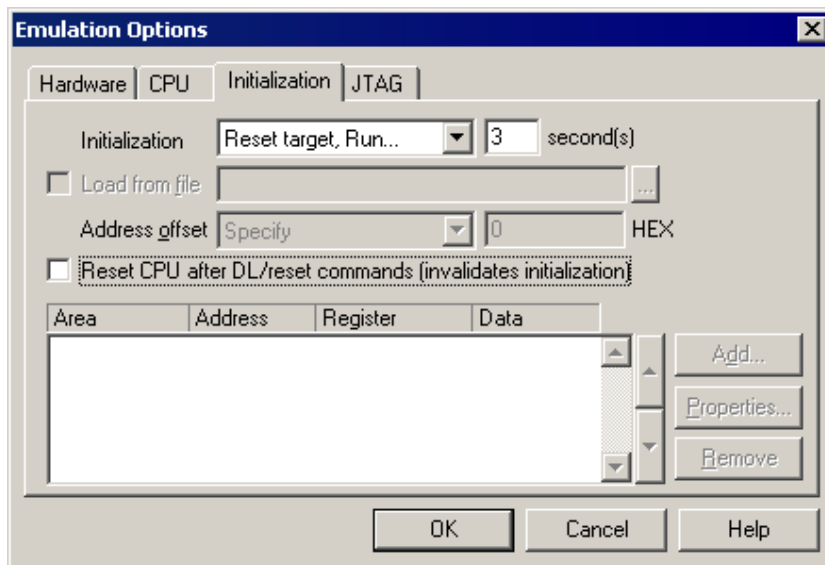
Excerpt from .ini file:

```
S I2CFG B 0x34 // I2C configuration register
```

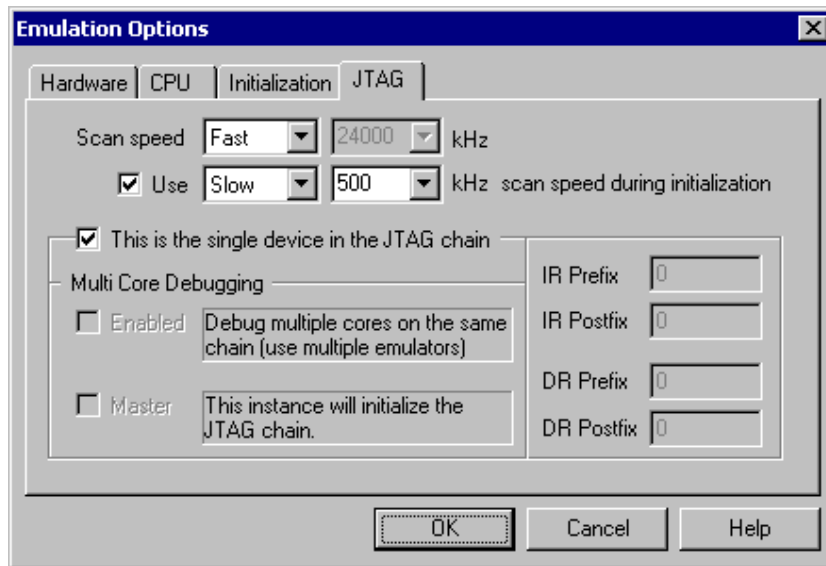


The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

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.



2.3 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.
- Burst – provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz.
- Burst+ - provides the ability to set the JTAG clock frequency varying from 4 MHz to 100 MHz

Slow and Fast JTAG scanning is implemented by means of software toggling the necessary JTAG signals. Burst mode is a mixture of software and hardware based scanning and should normally work except when the JTAG scan frequency is an issue that is when the JTAG scan frequency used by the hardware accelerator is too high for the CPU. In general, selecting an appropriate scan frequency usually depends on scan speed limitations of the CPU. In Burst+ mode, complete scan is controlled by the hardware accelerator, which poses some preconditions, which are not met with all CPUs. Consequentially, Burst+ mode doesn't work for all CPUs.

In general, Fast mode should be used as a default setting. If the debugger works stable with this setting, try Burst or Burst+ mode to increase the download speed. If Fast mode already fails, try Slow mode at different scan frequencies until you find a working setting.

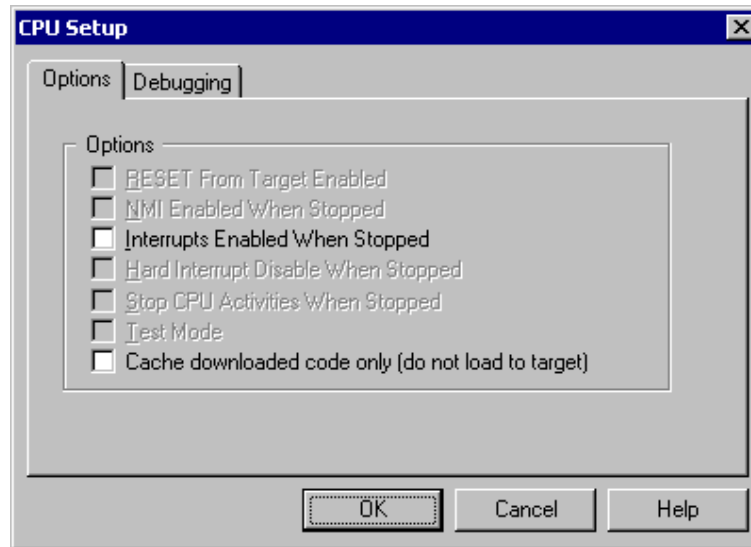
Note: Burst and Burst+ modes are implemented for PowerPC and ARM CPUs, including XScale.

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



XC800 Family 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 behaviour during single step.

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

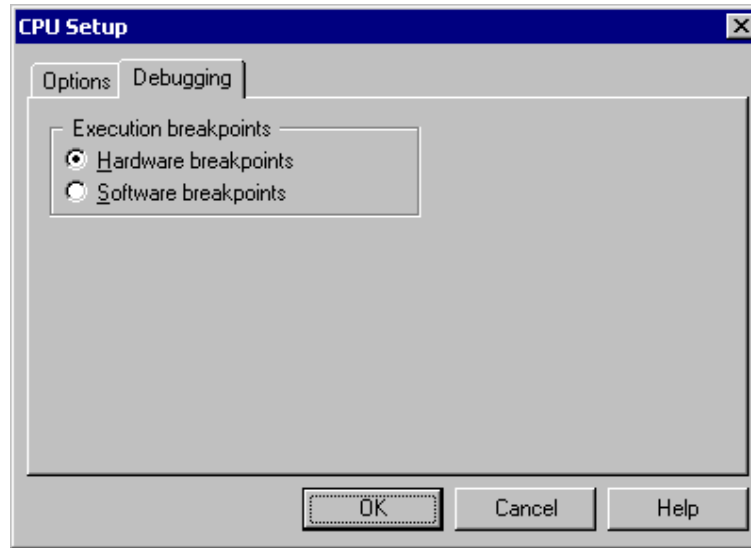
If this option is enabled, the Emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

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 Debugging Options



XC800 Family Debugging Options

Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to four. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writable memory (ROM) or self-modifying code. If 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 and access breakpoints. Consequentially, debug resources used by one debug functionality are not available for the other debug functionality. In practice this would mean that no access breakpoint 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. Note that the debugger features unlimited software breakpoints in the XC800 internal flash too.

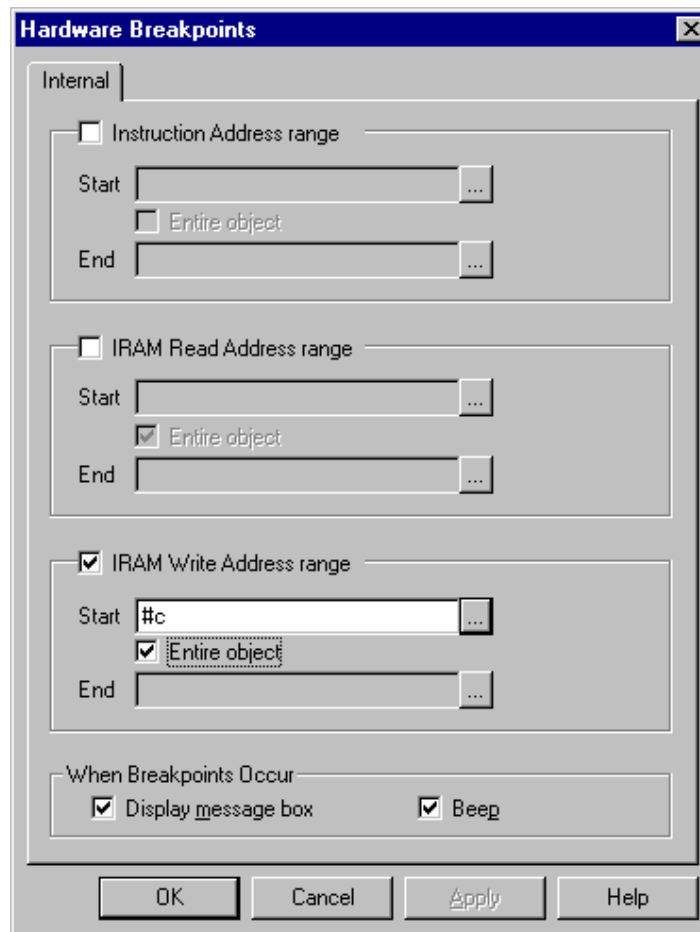
When a software breakpoint is being used, the program first attempts to modify the source code by placing a break instruction into the code. If setting software breakpoint fails, a hardware breakpoint is used instead.

4 Internal FLASH Programming

The XC800 CPUs have internal FLASH which is programmed through standard debug download; thereby no standard FLASH setup dialog is available. The debugger recognizes which code from the download file fits in the FLASH. All necessary FLASH programming settings is done automatically.

5 Access Breakpoints

The same on-chip debug resources are shared among hardware execution breakpoints and access breakpoints. Consequentially, debug resources used by one debug functionality are not available for the other debug functionality. In practice this would mean that no access breakpoint can be set for instance on instruction address range, when four execution breakpoints are set already.



Access breakpoint can be set on:

- Instruction Address range
- IRAM Read Address range
- IRAM Write Address range

6 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 the reset location.
- 6) Open memory window at internal CPU RAM location(s) 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.

7 Troubleshooting

When performing any kind of checksum, remove all software breakpoints since they may impact the checksum result.

Make sure that the power supply is applied to the target JTAG 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.

Try 'Slow' JTAG Scan speed if the debugger cannot connect to the CPU.

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.