Freescale 68HC12 Family On-Chip Emulation

Contents

1 Introduction

The term BDM stands for Background Debug Mode. It is used for the system development and FLASH programming. A BDM firmware is implemented on the CPU silicon providing a comprehensive set of debug functionalities.

Since BDM control logic does not reside in the CPU core, BDM hardware commands can be executed while the CPU is operating normally. The control logic generally uses CPU dead cycles to execute these commands, but can steal cycles from the CPU when necessary. Other BDM commands are firmware based, and require the CPU to be in active background mode for execution. While BDM is active, the CPU executes a firmware program located in a small on-chip ROM that is available in the standard 64-Kbyte memory map only while BDM is active.

To stop the CPU after reset, BDM must be enabled (first phase) and activated (second phase).

After reset, the BDM is neither enabled nor active, except when CPU is started in the special single-chip mode. The debugger enables and activates it when necessary. In special single-chip mode, BDM is enabled and active immediately after reset.

The BDM control logic communicates with an external development system serially, via the BKGD pin. This single-wire approach minimizes the number of pins needed for the development support.

BDM is mainly used as a low cost debugging solution or alternatively for FLASH programming

When the user's program is stopped, the CPU enters and operates in the BDM mode. Note that internal peripheral timers, previously active in the user's program, remain active after the BDM mode is entered. However, interrupts are disabled when BDM mode is entered.

Debug Features:

- Two hardware breakpoints
- Unlimited software breakpoints
- Real-time access
- Fast flash programming
- Hot Attach
- COP Support
- PLL Support

1.1 Single-chip Applications

Always, one hardware breakpoint is available for the user and can be set on any address in the CPU space. In fact, the on-chip firmware provides two hardware breakpoints. The second one can be used either by the user or by the debugger (high level source stepping).

Note: 68HC12A4 CPU doesn't provide any hardware breakpoints. If the code is executed in the internal EEPROM, the debugger sets breakpoints by modifying the code since the CPU can write to the EEPROM with no restrictions.

Using Debug Download, high-level debug info and code to the internal EEPROM and RAM can be downloaded. In typical applications, the code is always loaded into the FLASH.

Code can be loaded into the internal FLASH using the FLASH Programming dialog. Note that typically, number of FLASH programming cycles is limited and extra FLASH programming voltage must be provided to the target CPU, depending on the specific CPU.

1.2 Expanded Applications

The target contains external memory device(s) beside the CPU. External memory can be RAM or ROM type.

RAM

The debugger can set software breakpoints in code that resides in the internal and external RAM. The debugger modifies program code in places where breakpoints are set.

Note: Software breakpoints don't function well with self-modifying code.

The user's program can be loaded to the target RAM via BDM.

ROM

The default debugger functionalities are the same as in the single chip application. To debug the application where a code resides in the target ROM, use of ROM/RAM simulation unit becomes relevant. Note that the debugger cannot set software breakpoints in the target ROM memory neither can download the code. A number of available hardware breakpoints is limited to 2 and at HC12A4 no hardware breakpoints are available at all. In such case, the target ROM device should be replaced with PROM simulation unit, operating simultaneously in the interaction with BDM debugger. Such development system gains some in-circuit emulation features like:

- Fast download
- Unlimited number of breakpoints

IN ANY OTHER MODE THAN SPECIAL SINGLE CHIP, the on-chip BDM firmware is not active out of reset. Therefore, the CPU cannot be stopped immediately out of reset. So, after releasing reset, the CPU starts to execute the program and in the mean time, the debugger synchronizes with on-chip BDM firmware, activate and enable it and stop the CPU. As late as now, the debugger cannot download any code. However, the main problem is that the CPU starts to execute the code after releasing the CPU out of the first reset. Note that at that moment no code was downloaded yet (e.g. in the external RAM), so in fact, the CPU is executing garbage. Consequently, the CPU execution misbehaves and the debugger cannot synchronize with on-chip BDM firmware and the emulation/initialization fails.

Above behavior is the technique, how the on-chip BDM works and it has nothing to do with the debugger.

Two workarounds exist in such cases:

1) Either use the PROM simulation unit (BaseUnit+ or REmulator), so the debugger downloads the code to the PROM simulation unit, simulating external RAM/ROM, directly via PC. In this case, the CPU will execute already a valid program after releasing the CPU out of reset for the first time. In the mean time, the debugger synchronizes with on-chip BDM and takes over the control over the CPU. Meaning, the system is initialized and can be debugged.

The system initializes in the following order:

- The code is downloaded by the debugger to the PROM simulation unit being connected to the target.
- The CPU is released out of reset in the expanded mode.
- The code is executed and in the mean time, BDM communication established.
- The CPU is stopped by the debugger and the system is ready for debugging.

2) Start the application (target CPU) in the special single chip mode. The target must force the single chip mode and the emulator should force special mode. Note that the debugger can force either normal or special mode, selectable in the software. After releasing the CPU out of reset, the debugger has a control over the application at once, since the on-chip BDM is active and enabled already. Now, for the user to be able to download the code in the external target RAM, the initialization sequence must be used. Now, using the initialization sequence, the user must alter the operating mode from the single chip to the expanded mode, so the debugger can actually download the code to the external target RAM via external CPU bus.

The initialization sequence is executed after releasing the CPU out of reset and next the download is performed. After the download is finished, the debugger still has a control over the CPU and can either stop or run the program.

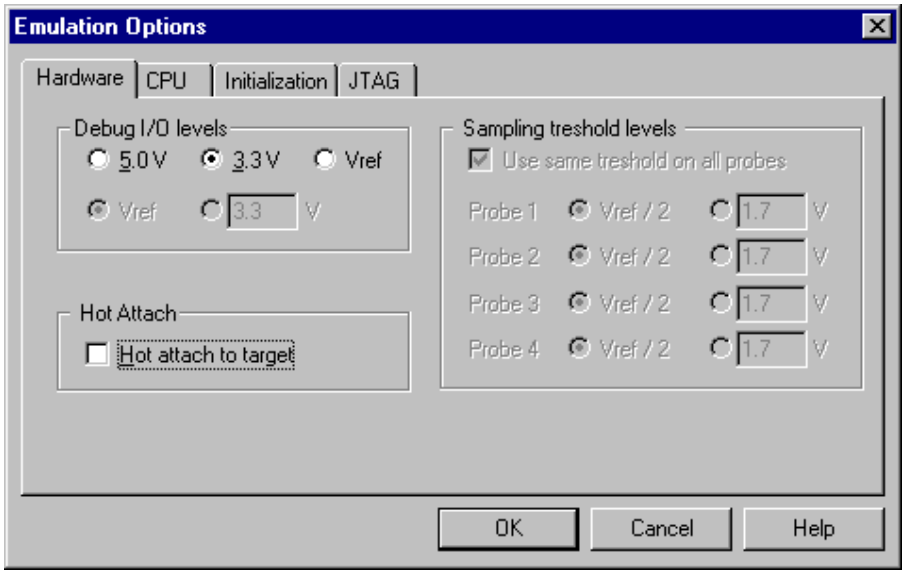
The system initializes in following order:

- The CPU is released out of reset in the special single chip mode.
- The CPU is stopped by the debugger.
- The initialization sequence is executed (single chip mode -> expanded mode).
- The download is performed.
- The CPU is either stopped or run and ready for debugging

Note that this workaround can be used only when using RAM type of memory in the target.

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 the debug BDM signals are driven at 3.3V, 5V or target voltage level (Vref).

When 'Vref' Debug I/O level is selected, a voltage applied to the belonging reference voltage pin on the target debug connector is used as a reference voltage for voltage follower, which powers buffers, driving the debug BDM signals. The user must ensure that the target power supply is connected to the Vref pin on the target BDM connector and that it is switched on before the debug session is started. If these two conditions are not meet, it is highly probably that the initial debug connection will fail already. However in some cases it may succeed but then the system will behave abnormal.

Hot Attach

The debugger supports Hot Attach function. This is a function, which enables the emulator to be connected to a working target device and have all debug functions available. See ‘Hot Attach’ chapter for more details on Hot Attach use.

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

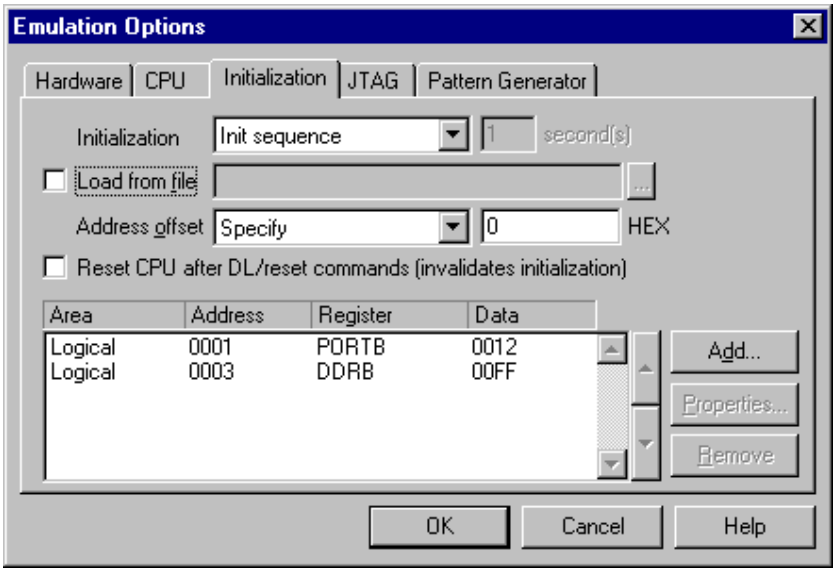
2.2 Initialization Sequence

Before the flash programming or download can take place, the user must ensure that the memory is accessible. This is very important since there are many applications using memory resources (e.g. external RAM, external flash), which are not accessible after the CPU reset. In that case, the debugger must execute after the CPU reset 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.

Note: Normally, there is no need for initialization sequence in case of a single chip application/CPU.

The initialization sequence can be set up in two ways:

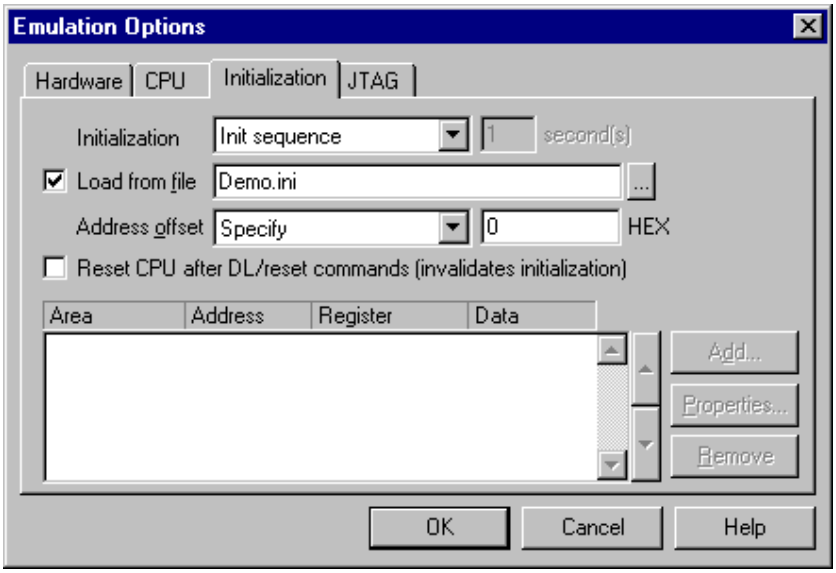
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

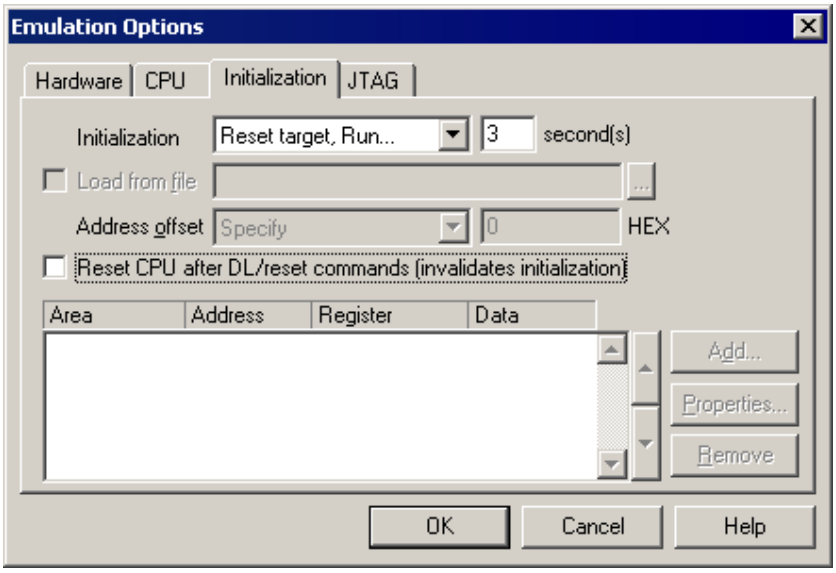
   Excerpt from the sample Demo.ini file:

```
S PORTB W 0012          //comment
S DDRB W 00FF
```



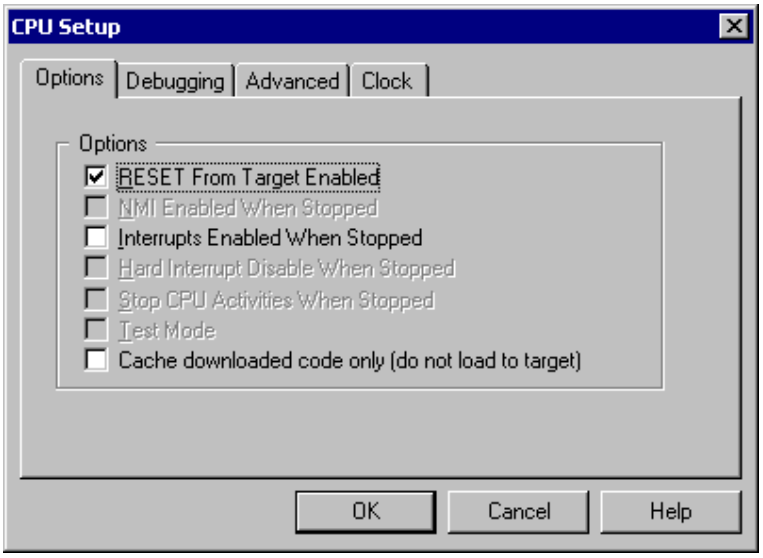
The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

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.



3 CPU Setup

3.1 General Options



CPU Setup dialog, Options menu

RESET from Target Enabled

Beside the debugger, the target can have additional external reset sources, like power-on reset, watchdog circuitry or even reset push-button. In general, it's recommended to disable all external reset sources in the target, which may disturb the debugger in a way that BDM communication is lost and complete system needs to be reinitialized.

It's recommended that all reset sources are designed as an open drain type. 'Reset from Target Enabled' option in the 'CPU Setup/Options' tab must be normally checked to assure safer debugging. Then the debugger can detect any reset source and service it properly.

Since target reset lines are designed as an open drain type, the debugger can detect all resets, even if they have been initiated by hardware other than the emulator itself. In certain applications, though, the requirement to disable this type of checking is required. When the CPU operates in an environment with interferences, it's recommended to add a capacitor to the target reset line. In order for the CPU to be able to initiate a reset and then to differentiate among the 'internal' and 'external target' reset (the length of less than 8 ECLK cycles), the capacitor must be relatively small. However, the interference can be in some cases so big that a bigger capacitor must be used. In such case, the debugger does not function correctly, if reset from the target is not disabled.

To disable reset sources from the target to be detected by the debugger, uncheck the 'RESET From Target Enabled' option. In this case, only the emulator will be able to generate a reset and the debugger will ignore all reset sources from the target.

Note: Wrong setting of this option can significantly change the operation of the target!

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.

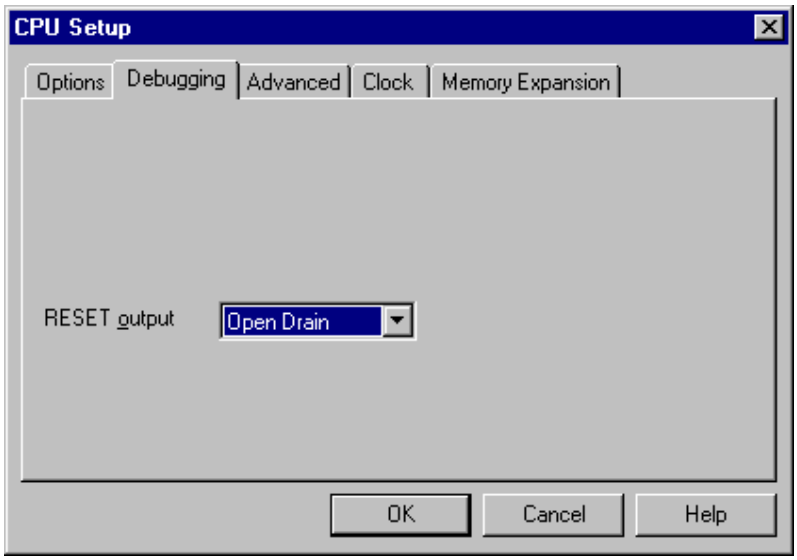
If this option is enabled, the Emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

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



CPU Setup, Debugging menu

RESET Output

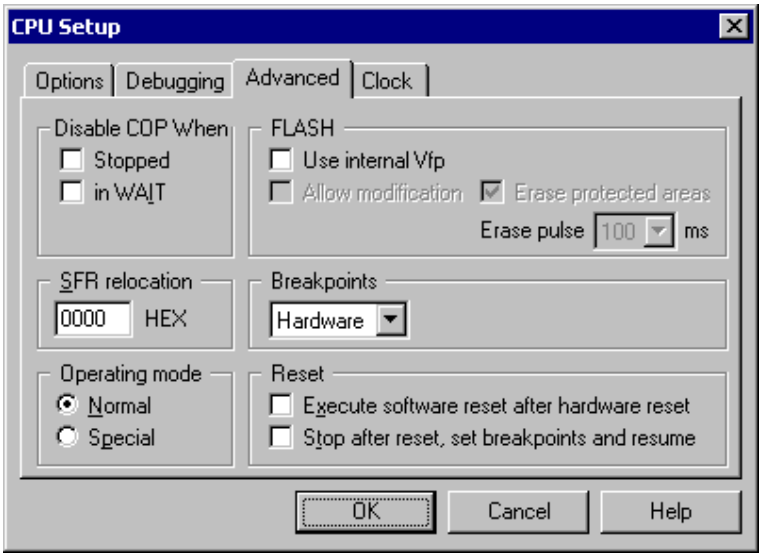
The reset line from the debugger can be driven as an open drain or as push-pull line. Typically, open drain reset signals are used, but in special cases, push-pull signals are preferred.

A special case is, if a large capacitor is located on the reset line because of interference.

Next case is, if the target contains an external watchdog that cannot be disabled. During the debugging phase, it must somehow be disabled. In this case, the watchdog line should be connected to the CPU reset line through a serial resistor (for example 1k). If the emulator's reset line is connected directly to the CPU reset line and its type is push–pull, then the watchdog can no longer reset the CPU by itself.

Note: Wrong setting of this option can significantly change the operation of the target!

3.3 Advanced Options



HC12 Family Advanced Options

Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to two. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writeable memory (ROM) or self-modifying code. If the option 'Use hardware breakpoints' is selected, only hardware breakpoints are used for execution breakpoints.

Note: Two hardware breakpoints are available on all HC12 CPUs except for the HC12A0 and HC12A4 CPUs.

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.

Using Hardware Breakpoints in a Banked Application

Note that when the on-chip hardware breakpoint is used as an execution breakpoint in the bank, it doesn't work as expected. The debugger sets specific breakpoint registers when hardware breakpoint is set. All registers are 16 bit, therefore when setting address breakpoint register, only lower 16 bits of address are compared. Higher address lines are ignored and have no effect in the banked application. Therefore, an execution breakpoint can be set on 16-bit logical address only.

In practice, when setting BP on bank address e.g. 4.8002 the CPU will stop on it, but also on address 1.8002, 2.8002, 3.8002, 5.8002, etc.

Note that above limitation becomes a problem only on HC12Dx128 devices, when memory expansion is used. The limitation does not concern HC12A0 and HC12A4, which have no on-chip hardware breakpoints.

Software Breakpoints

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation.

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.

PROM Breakpoints (iC2000 development system only)

PROM breakpoints are set by code modification to PROM pseudo devices. Select this option when emulating FLASH, EEPROM or any read-only memory and this memory is reproduced on the Emulator.

Operating Mode

The debugger can force two CPU operating modes in which the CPU behaves differently:

- Normal mode – some registers and bits are protected against accidental changes
- Special mode – allows greater access to protected control registers and bits for special purposes such as testing and emulation. The debugger can force the CPU to active BDM mode immediately after reset while CPU operates in the special single-chip mode only.

Additionally, the CPU can operate in single chip and expanded mode. Refer to the CPU datasheet for more details on CPU operating modes.

In special single chip mode, the on-chip BDM firmware is active immediately out of the reset. The CPU is stopped immediately after reset and the debugger has complete control over the CPU.

IN ANY OTHER MODE THAN SPECIAL SINGLE CHIP, the on-chip BDM firmware is not active out of the reset and the CPU cannot be stopped immediately. After releasing the reset line, the CPU starts to execute the program and in the mean time, the debugger synchronizes with on-chip BDM firmware, activates and enables it, stops the CPU and gains control over the CPU.

Now, the problem pops up when the flash is empty or contains the program not being operational. In such case, the CPU may hang already while the debugger synchronizes with the on-chip BDM and tries to gain the control over the CPU. If the CPU hangs, the debug connection cannot be established.

In such case, the only alternative is to program the flash using special single chip mode in which the debugger controls the CPU immediately after reset. After the flash contains a valid program, Normal mode can be used.

It's recommended to check '*Execute software reset after hardware reset*' option in the *CPU Setup* dialog in all operating modes, except in special single-chip mode. When the option is checked, the CPU starts executing the code and after approx. 500us after reset is released, a BDM communication is established, CPU stopped, reset vector read and program counter preset to that address.

Reset

These options control the CPU behavior after RESET. This is necessary because the CPU cannot be stopped immediately after reset, except for special single chip mode. The CPU runs for approximately 500 µs before the Emulator can enable BDM communication and issue a stop command.

'Execute software reset after hardware reset' option stops the CPU, reads the reset vector from the memory and presets CPU program counter to that address. Since the CPU runs for 500 µs before the debugger can stop it, this reset is not equivalent to a regular hardware reset. Whether this option is safe to use it depends on the target application. Note that it makes no sense to use this option in special single chip mode since BDM is active immediately after reset and the CPU can be stopped at reset program counter.

'Stop execution after reset to set breakpoints and resume' option is available if the above option is not checked. You will want to use this option to allow the debugger to set breakpoints (software or hardware), for which the CPU must be stopped. The internal breakpoint logic resets on every hardware reset and therefore the breakpoints must be written again.

SFR Relocation

User must specify new SFR base address offset when application reallocates SFR memory area from the default reset address. Read ‘SFR Relocation’ chapter for more details on how to use this option.

Disable COP When

These settings define COP behaviour during debugging. They should be used when the user's program doesn't disable the COP during debugging. Please refer to ‘COP Use’ chapter for more details.

Erase Pulse Duration

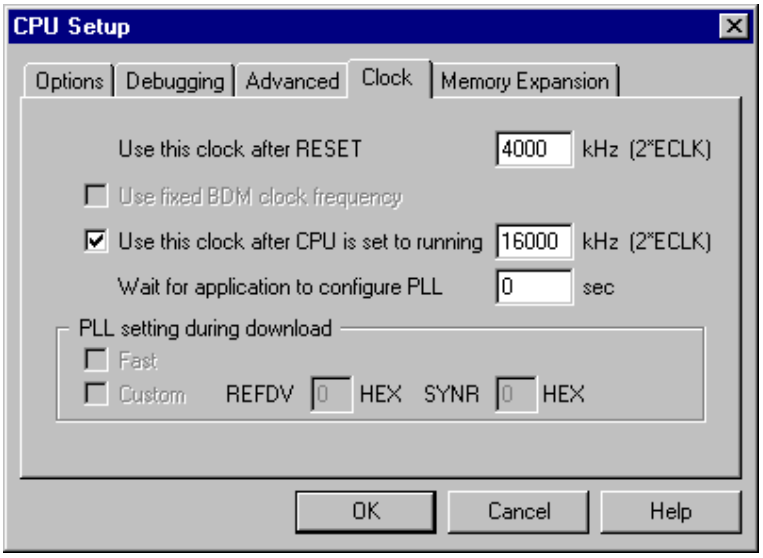
In some revisions of the 68HC12D60 the erase pulse duration is changed from 100 ms to 10 ms. With this setting it’s defined the flash erase pulse duration.

Use internal Vfp

If this option is selected, the Emulator generates 12V, which are required to program the flash memory. The Vfp is connected to pin5 on the BDM connector and on a special pin on the iCARD.

Note: This option is supported only on iC2000 BaseUnit+, REmulator with the BDM module revision H and iC3000/iC4000 Emulators.

3.4 Clock options



Clock options menu

Use this clock after RESET

The CPU clock must be specified here to allow BDM communication synchronization.

Use this clock after CPU is set to running

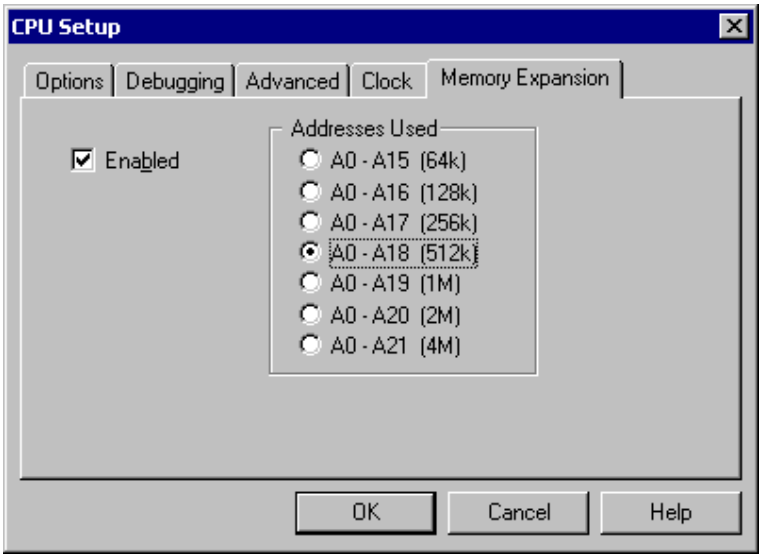
If the PLL is engaged, the clock rate is changed due to internal CPU errors. In order to keep the BDM communication synchronized, new clock (after PLL is enabled) must be defined here, which will be used after the CPU is set to running. See ‘PLL Use’ chapter for more details on PLL use.

This option should be unchecked when PLL is not used.

Wait for application to configure PLL

Specifies the time to wait for the application to configure PLL in seconds before the second (running) clock is applied.

3.5 Memory Expansion Options



68HC12 Family Memory Expansion Options

Enabled

Check if you are using a banked application.

Addresses Used

Specify how many addresses are used in the application.

Note: Make sure that this setting matches the value written to MXAR register in the target program. Otherwise the emulation can fail.

The addresses used can only be selected when using the HC12A4 CPU. With every other CPU, that supports memory expansion, the number of banks and the number of addresses is defined automatically.

4 Download

Internal FLASH

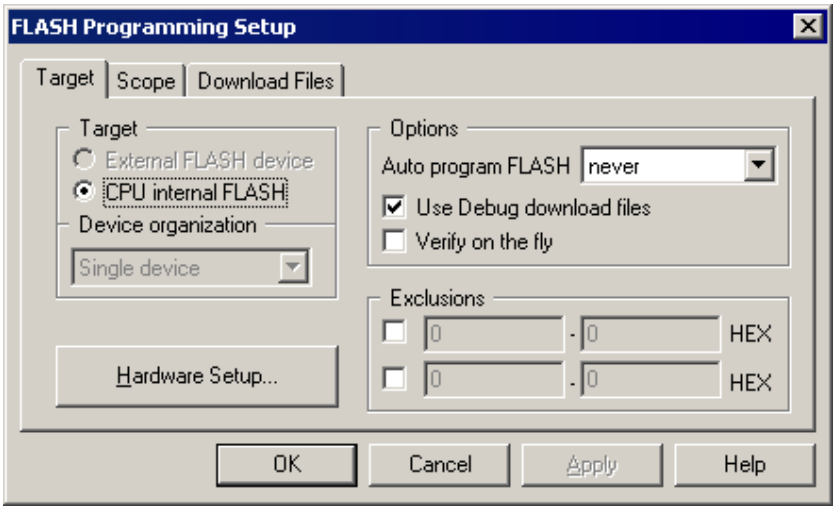
Code residing in the internal flash is programmed through the ‘FLASH Program’ dialog.

The debugger expects bank addresses in the download file by default. It’s recommended that the project is compiled in a way that the download file contains bank addresses.

If that's not possible and the file contains, for instance, linear addresses, the user must force linear memory space by selecting ‘Linear’ in the ‘Memory area’ combo box when specifying the download file.

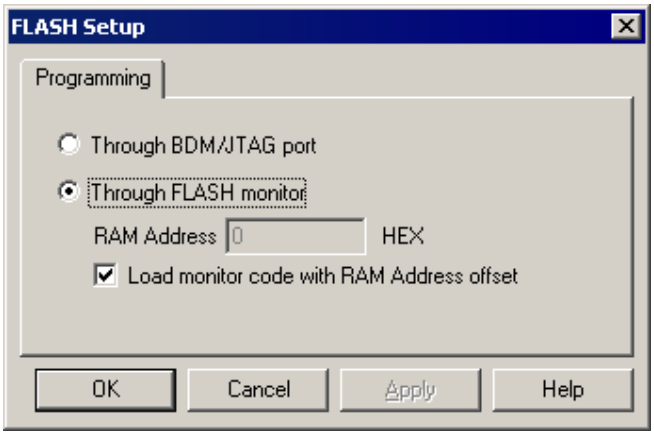
The 'Enabled' option in the Hardware/Emulation Options/CPU Setup/Memory Expansion’ tab must be checked if the code is linked for a bank application. It must be unchecked, if it’s a non-bank project.

Few parameters must be set in the ‘FLASH/FLASH Programming Setup’ dialog (FLASH/Setup…) before the internal flash can be actually programmed.



When the CPU internal flash is programmed, winIDEA takes care of most of the necessary settings. Don’t check ‘Verify on the fly’ option since this functionality cannot be supported on this CPU family.

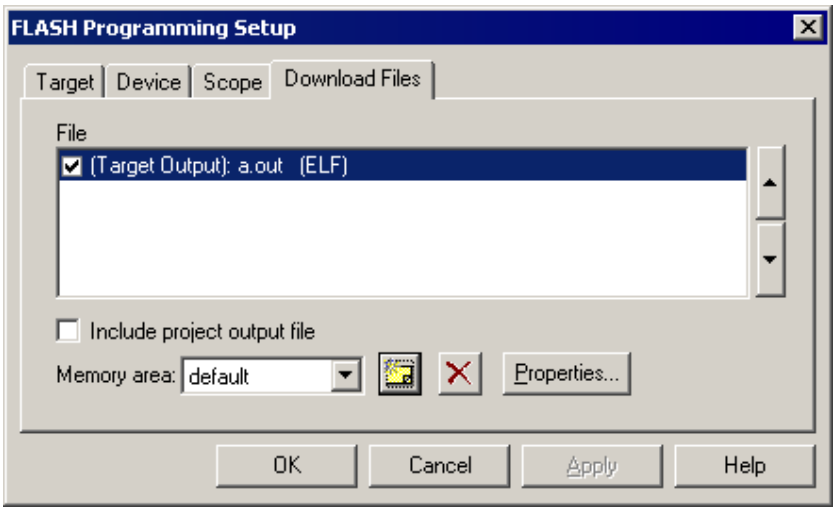
Next, flash programming type must be selected. WinIDEA supports flash programming through the debug BDM port and fast FLASH monitor. Press ‘Hardware Setup…’ button in the ‘Target’ tab in the ‘FLASH Programming Setup’ dialog for the selection.



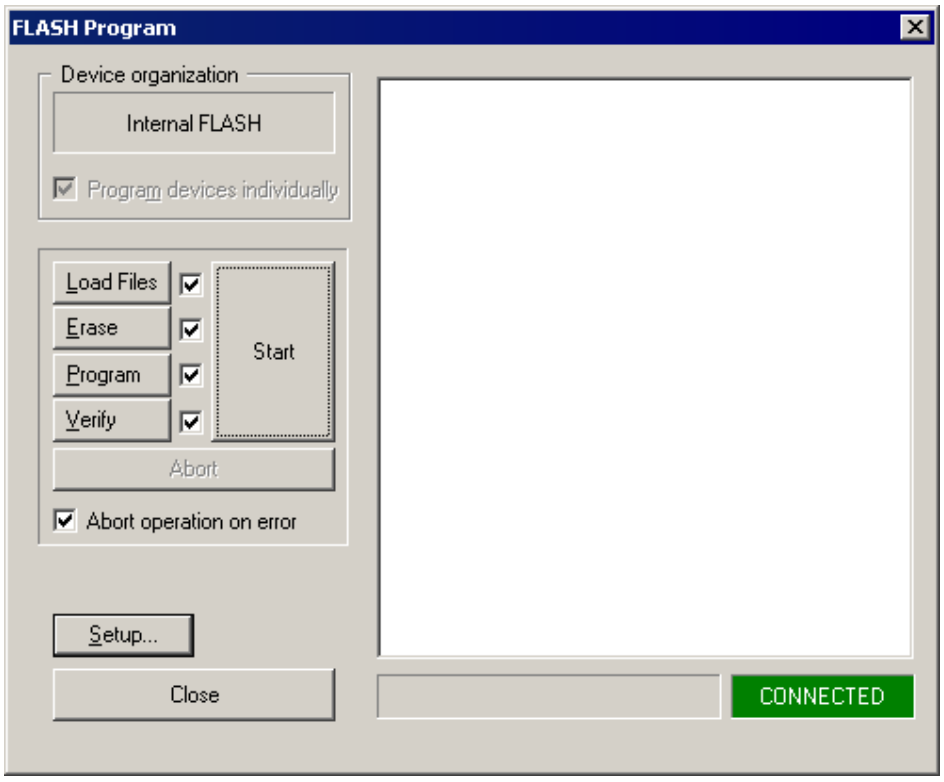
Normally, the user should go straight for fast FLASH monitor use. Programming through the BDM port is much slower and recommended to be used when troubleshooting flash programming.

When programming the flash through the monitor, make sure you don't relocate the internal CPU RAM. The debugger assumes default reset RAM location and allocates flash programming monitor adequately.

A file or more files to be programmed can be selected in the 'Download files' tab within the 'FLASH Programming Setup' dialog. The alternative is to specify file(s) in the 'Debug/Files for Download/Download files' tab, where normally files for debug download are specified. Make sure that 'Use Debug download files' option ('Target' tab in the 'FLASH Programming Setup' dialog) is checked then. In first case, the option must be unchecked.

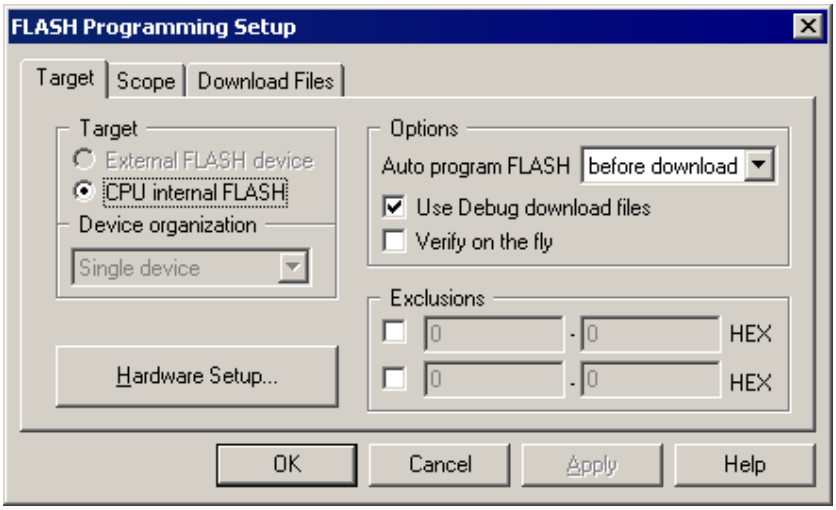


'FLASH Program' dialog should be invoked from the 'FLASH' menu after the flash programming is configured in the 'FLASH Programming Setup' dialog.



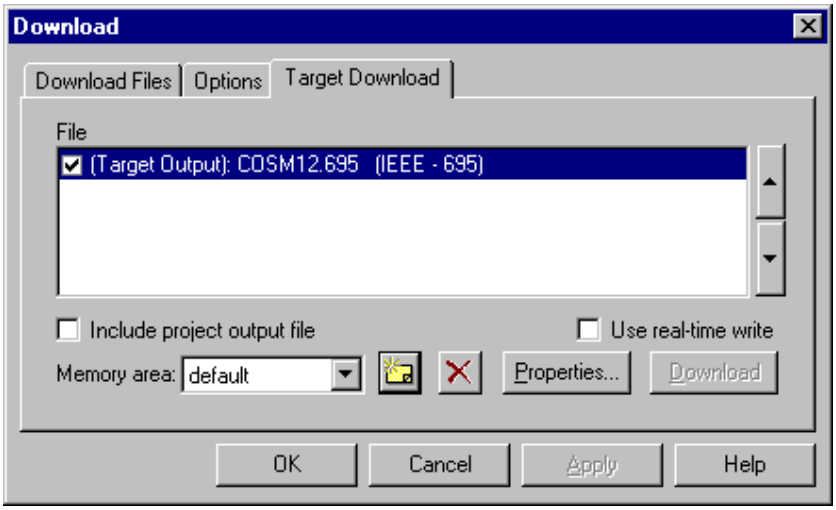
Normally, check boxes beside Load Files, Erase, Program and Verify buttons should be checked. Flash programming is started by pressing 'Start' button. During the flash programming, a status and eventual errors are displayed in the dialog.

The debugger can program the flash automatically before the download without any need for opening the 'FLASH Program' dialog by the user. Then 'before download' in the 'Auto program FLASH' combo box must be selected.

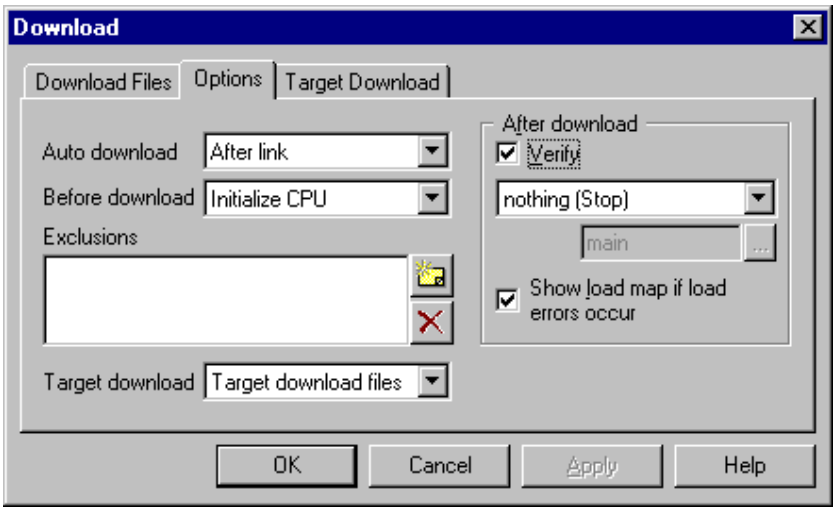


Internal RAM

Code, residing in the internal RAM, EEPROM or in the target RAM, must be downloaded using a so called *Target Download*. First, add files to be loaded in the 'Target Download' tab.



Next, select 'Target download files' selection in the 'Target Download' combo box. Now, Download debug command downloads all listed files.



Internal EEPROM

If the user needs to download certain data to the EEPROM memory in the download phase, the following must be done:

1) 'target download' must be used;

2) the ECLKDIV register must be set using the initialization sequence;

3) if the whole EEPROM is required, the EEPROM area must be relocated with the initialization sequence (by writing to the INITEE register). This is needed, because on some CPUs the EEPROM area is covered with the I/O area. Don't forget to set a proper offset for the download file when relocating EEPROM memory area.

4) The EEPROM writes must not be disabled.

5 Real-Time Memory Access

With this type of CPUs, real-time memory access is available. Watch window's ***Rt.Watch*** panes can be configured to inspect memory without stalling the CPU. Optionally, memory and SFR windows can be configured to use real-time access as well.

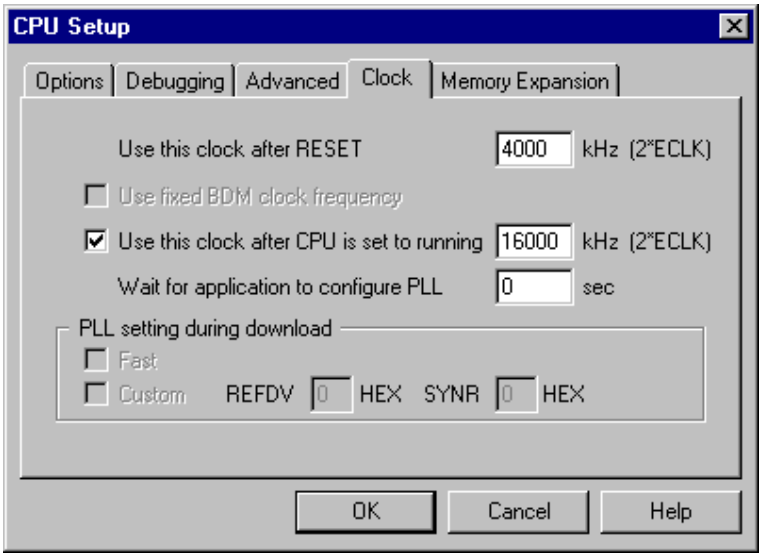
Please refer to the Software User's Guide for more information on Real-Time watches.

6 PLL Use

After the PLL is activated, the CPU system clock changes and consequently the BDM clock. The debugger must synchronize to a new frequency, otherwise BDM communication falls out. Reset and PLL operating frequency must be entered in the 'Clock' tab in the '*CPU Setup'* dialog.

After reset, the debugger connects to the on-chip BDM at 'reset frequency'. There are less startup problems when using special single chip mode since the CPU stops immediately after reset. With any other CPU operating mode, a small portion of the program is executed before the CPU can be stopped after reset. This portion of the program must not initialize and engage the PLL yet or the debugger will fail to establish the control over the application.

Use special single chip mode whenever it's possible. In any other mode, the user must assure that the PLL is not enabled within first 500µs of program execution. In worst case, the user must add a delay in his program to meet this requirement.



When the CPU is stopped after reset, the user can use *step* debug command to step through the program up to the PLL initialization routine since the debugger keeps using 'reset frequency'. It's dissuaded to step through the PLL initialization routine.

As soon as the CPU is set into running for the first time (this excludes source and code single step), the debugger will use the 'operating frequency' to communicate with the on-chip BDM. It's recommended that the PLL is engaged shortly after the application resumes, which is normally the case. However, if that's not the case, make sure that the application is not stopped before the PLL is engaged and the debugger is configured adequately. The user needs to estimate the time between the point when the program is resumed and when the PLL is actually engaged in the application and then enter it in seconds in the 'Wait for application to configure PLL' field.

Tip: Use '*After download run until*' option in the 'Debug/Files for download/Download Options' tab and define the source line located after the PLL initialization routine. By doing so, you won't need to worry about the PLL any longer since the debugger will already use new PLL clock when it stops.

7 SFR Relocation

Hardware breakpoints are implemented using breakpoint registers, which are part of special function registers. After SFRs are relocated, the debugger should detect new location of breakpoint registers otherwise the emulation would misbehave. Since auto-detection is not possible on the HC12 family, an extra option was implemented called '*SFR Relocation*'.

After reset, the debugger uses registers' reset address (0h) and uses new relocated registers' address after *run* debug command is used for the first time.

After reset, the user can use *step* command up to Relocate_SFRs routine. Additionally, the CPU must not be stopped before Relocate_SFRs routine is executed, when using *run* command for the first time. If the program would be stopped before Relocate_SFRs routine, the debugger would try to use breakpoint registers at new address, even though they are still at reset location. Hence, emulation would fail.

The same rules apply for SFR relocation and PLL use when CPU doesn't operate in the special single-chip mode. It's recommended to use special mode in single-chip applications. Otherwise, the user's program shouldn't execute Relocate_SFRs routine before the debugger activates on-chip BDM and stop the CPU. A delay routine (~1ms) should be added before Relocate_SFRs routine. This will guarantee that debugger activates BDM and stops the CPU prior to Relocate_SFRs routine execution, while SFRs are still located on a default reset address.

8 COP Use

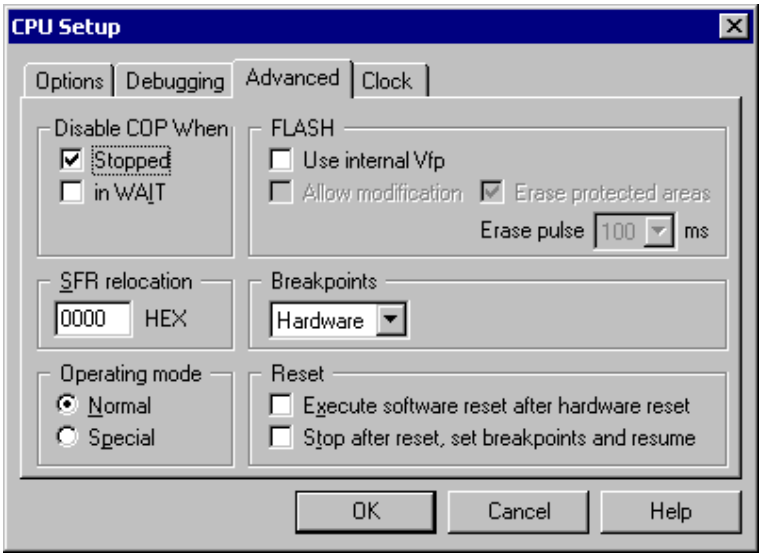
The debugger needs no watchdog awareness while watchdog is not used and disabled by the user's program. Therefore, following explanation is irrelevant for the application not using COP.

When using the watchdog, it must be serviced by the user's program properly. When the debugger stops the CPU for the first time, a specific register must be configured correspondingly, which is RTICTL register on HC12 family. When the user's program is stopped during the debugging, the CPU enters BDM mode in, which the COP must be disabled/stopped. Otherwise, the emulation fails.

When experiencing problems with the emulator when using COP, first check the RSBCK bit. Then please check, whether you have a correct/valid interrupt vector table. If some other interrupt vector is defined by mistake as a watchdog interrupt vector, it may look like the COP would reset the CPU when the interrupt occurs.

- Normal mode

In normal mode, COP is active after reset. Additionally, the CPU cannot be stopped immediately after reset. The CPU executes the user's program for approximately 500µs before it can be stopped by the debugger or by the user, by entering in the active BDM mode. After the CPU enters in the active BDM mode for the first time, bits RSWAI and RSBCK of the RTICTL register are set first. They are set according to the options 'Disable COP when Stopped' and 'Disable COP when in Wait' being set in the 'Advanced' dialog.



Advanced tab in the CPU Setup dialog

Do note that the user's application must not write RSWAI and RSBCK bits before the application is stopped for the first time, either by the debugger or by the user. This is necessary due to the reason that RSWAI and RSBCK bits are written once. Therefore, within 500us after reset, the application must not write RSWAI and RSBCK bits or it must write the same values as the debugger (RSBCK=1, RSWAI=0/1).

- Special mode

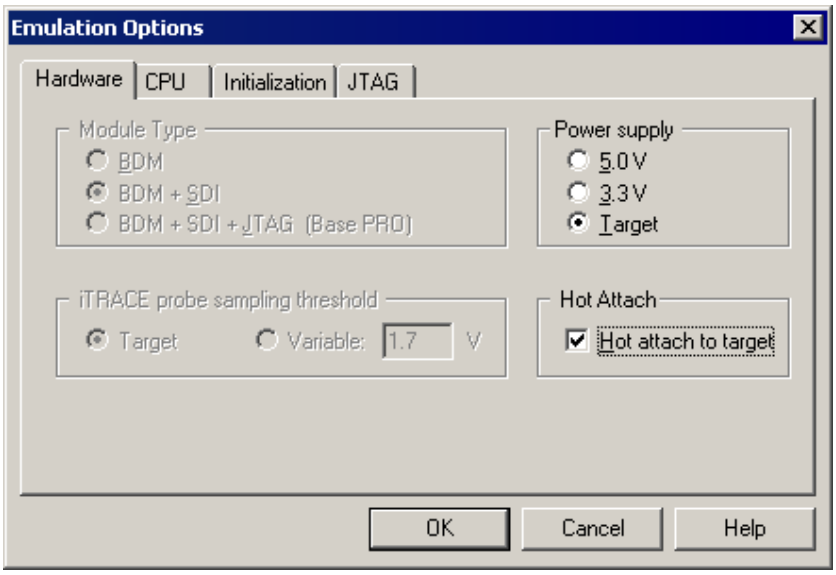
In special mode, COP is disabled after reset and the CPU can be stopped immediately after reset by the debugger. After the CPU enters in the active BDM mode for the first time, bits RSWAI and RSBCK of the RTICTL register are set first. They are set according to the options being set in the 'Advanced' dialog. These options must be set according to the user's application. Since RSWAI and RSBCK bits are write any time in special mode, the application must write the same values as the debugger (RSBCK=1, RSWAI=0/1) when the RTICTL register is written by the application.

9 Hot Attach

HC12 BDM allows attachment to a running target system without affecting its operation. Such operation is called Hot Attach.

It's assumed that there is a running target with no debugger connected. To hot attach:

- Check the 'Hot attach to target' option in the 'Hardware/Emulation Options/Hardware' tab.
- Execute Download debug command.
- Connect the BDM cable to the target system
- Select the 'Attach' debug command in the 'Debug' menu to attach to the target system.



Now, the debugger should display run status and the application can be stopped and debugged if necessary.

Select 'Detach' debug command in the 'Debug' menu to disconnect from the target application. If the CPU was stopped before detach, it will be set to running.

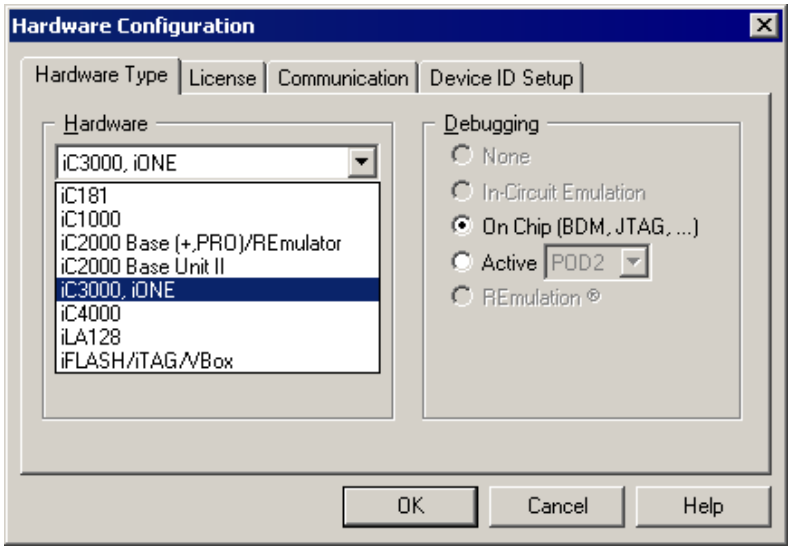
Note: Hot Attach function cannot be used for any flash programming or code download!

10 Getting Started

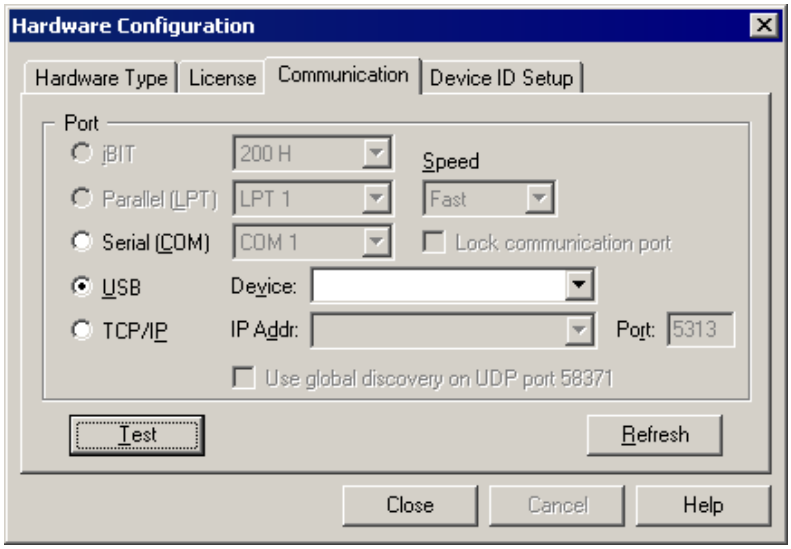
Quick start instructions help the user to start debugging a single-chip application in a very short time.

Setup and Initialization

- Connect the emulator to the PC where winIDEA is installed.
- Supply the power to the emulator and switch it on.
- Select Hardware by selecting 'Hardware/Hardware…'

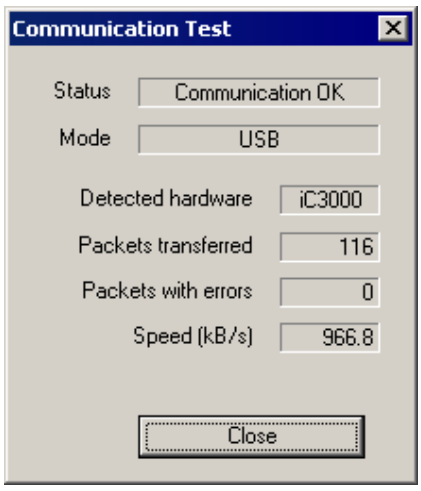


- Select 'Communication' tab within the same dialog and select the communication type that is going to be used.

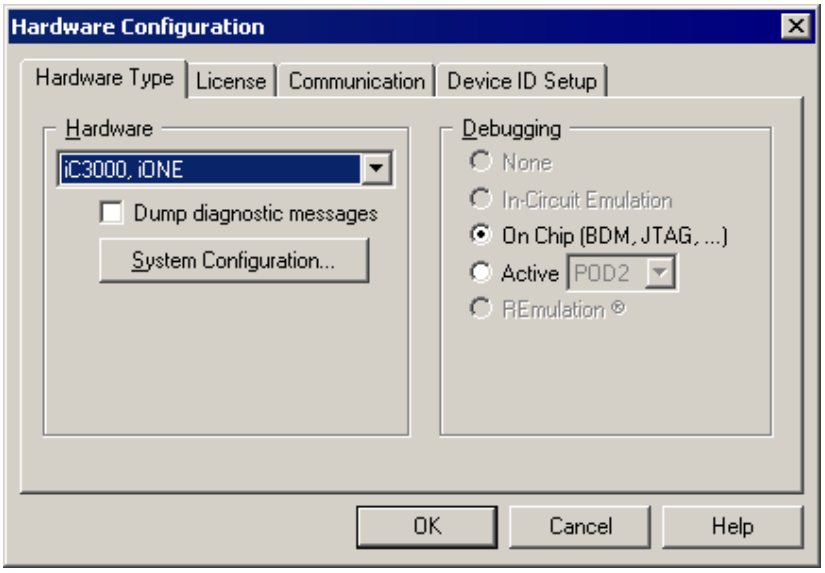


Refer to 'Setting up Communication' document (delivered beside the emulator) for more details on how to configure each of the supported communications.

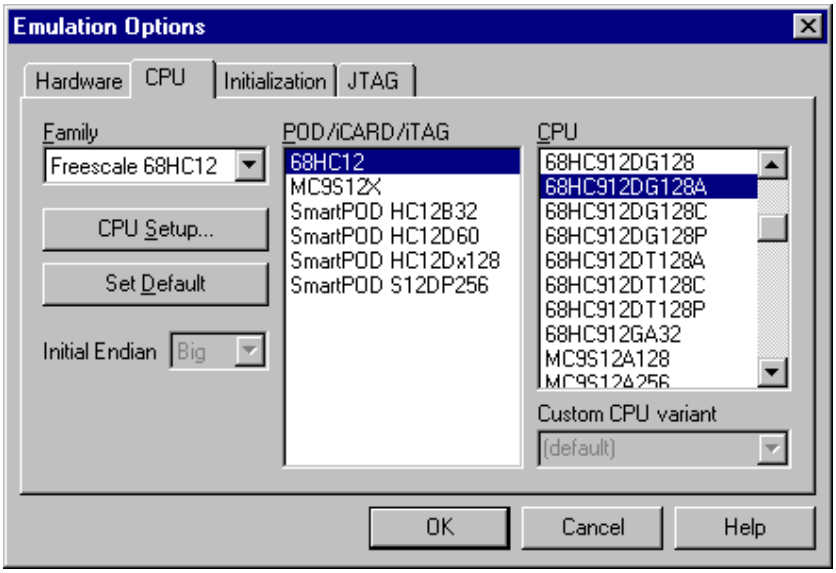
- Execute communication test by pressing ‘Test’ button. There must be no packets with errors reported during the communication test.



- Next, switch off the emulator and proceed with the configuration.
- Select ‘On-Chip (BDM, JTAG…)’ debugging type.



- Select the CPU family, the iCARD and the CPU in the ‘Hardware/Emulation Options’ dialog.

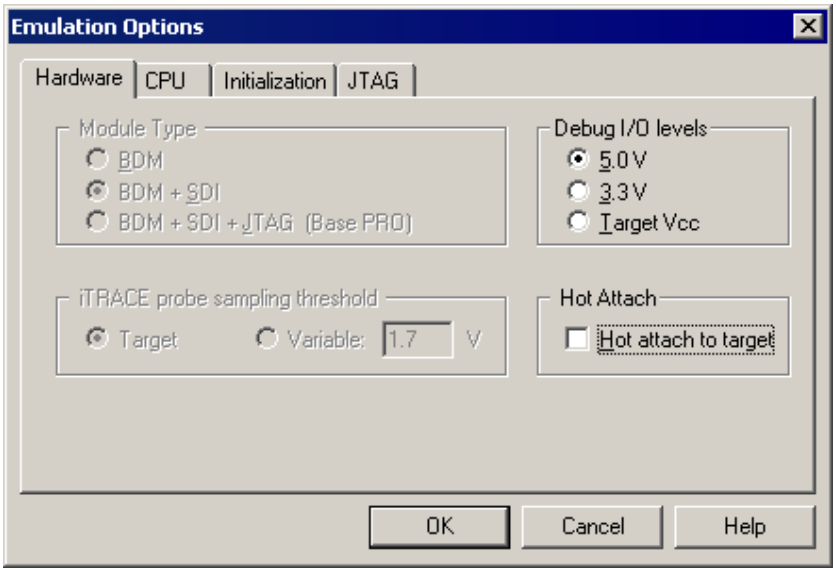


- Make sure that the 'Hot attach to target' option is unchecked in the 'Hardware/Emulation Options/Hardware' tab.

- Set CPU clock frequency in the 'Hardware/Emulation Options/CPU/CPU Setup/Clock' tab that your target uses. Note that the BDM protocol operates synchronously to the CPU clock.

- Select 'Special Mode' in the 'Advanced' tab for the operating mode.

- Select the Debug I/O levels in the 'Hardware/Emulation Options' dialog. The development tool can drive debug BDM signals at 5, 3.3V or target Vcc level. It's recommended to keep the default '5V' setting since all existing supported CPUs have 5V tolerant BDM debug signals.



- Normally, these are the minimum settings required by the emulator to be able to connect to the target CPU.

- Verify if the BDM connector in the target matches with the pinout defined by the CPU vendor. The required connector pinout can be also found in the hardware reference document delivered beside the debug iCARD.

- Connect the emulator to the target.
- First power on the emulator and then the target! On power off, first the target needs to be switched off and then the emulator. Otherwise damage to the hardware may occur!
- Close all debug windows in winIDEA except for the disassembly window.
- Execute debug CPU Reset command.

WinIDEA should display STOP status and disassembly window should display the code around the address where the program counter points to. If that's the case, the debugger is operational. You can inspect special function registers in the SFR window or read and modify (RAM) memory in the memory window.

Troubleshooting

If the debug CPU Reset command fails, verify:

- BDM cable, target BDM connector pinout and connection with the target
- clock frequency in your target and a frequency specified in winIDEA
- physical clock signal

  Measure EXTAL clock signal with oscilloscope. It may happen that the crystal doesn't oscillate.
- reset line of your target

There should be no other reset sources, which could disturb BDM communication. Remove all capacitors and other reset logic from the reset line and try again. Make sure that the 'RESET from target enabled' option is checked in the 'CPU Setup/Options' tab.

Next step is to download the program or more precisely to program the CPU internal flash.

FLASH Programming

- Add any file you would like to program in the FLASH. This file can be added as a debug download file used in this particular case or as FLASH programming file.

Note: It is recommended that all files contain bank addresses in case of bank application. Likewise, 'linear' addresses can be forced too. Logical addresses are not allowed.

Debug download file

Add a file in the Debug/Files for Download/Debug Files tab and check 'Use Debug download files' in the FLASH/Setup/Target tab.

FLASH programming file

Likewise, a file may be added in the 'FLASH/Setup/Download Files' tab and the 'Use Debug download files' option should be unchecked in the FLASH/Setup/Target tab.

- Select 'Entire chip' in the FLASH/Setup/Scope tab.
- Provide required programming voltage on Vfp pin of the CPU. Pay attention on proper voltage level or the FLASH programming may fail.

Note that newer HC12 derivatives no longer require a special Vfp.

When using iC3000 or iC4000 and if the Vfp voltage is required, it can be acquired from a special pin present on the front side of the HC12 iCARD and on pin 5 of the BDM connector. When using iC2000 BDM module revision H or newer, the Vfp voltage is also present on pin 5 of the BDM connector and must be connected to the CPU's Vfp pin. When using internal Vfp of the emulator, it must be enabled beforehand in the Hardware/BDM Emulation/CPU Setup dialog.

- Press 'Load – Erase – Program' button in the FLASH/Program dialog.

You should get a confirmation after each phase is successfully passed. The FLASH should be now programmed.

Troubleshooting

If you have problems with erasing the FLASH of HC12D60, try to change the erase pulse duration in the Hardware/BDM Emulation/CPU Setup dialog.

If FLASH programming fails, verify if:

- the proper CPU is selected
- the 'Hot Attach' option is unchecked
- the programming voltage is present and sufficient

Try to program another CPU too. It is also possible that the FLASH memory was already programmed more times than specified and has become unusable.

Debugging

- Now execute the Download debug command. Additionally, high-level debug info and code to the internal EEPROM and RAM are loaded if included in the download file.

Note: It is recommended that all files contain bank addresses in case of a bank application. Likewise, 'linear' addresses can be forced too.

The system should initialize and the debugger's status should be STOP. The program counter should point to the location where your reset vector points. If so, the FLASH programming was successful. If you don't use PLL and relocate SFRs in your project, you can start with debugging. All debug commands like Run, Step, Step Over, etc. can be used on assembler and high-level sources.

Have in mind that CPU operates in the special single-chip mode in our case. This mode must always be used if the FLASH is not programmed yet. After programming the FLASH, you can use normal mode.

Troubleshooting

If 'MUST INIT' state occurs after a while and/or it looks like a sporadic behavior you should:

- Read thoroughly 'PLL Use' chapter when PLL is used.
- Read thoroughly 'COP Use' chapter when COP is used.
- Read 'SFR Relocation' chapter when the application reallocates SFRs
- Double check the BDM connection. Press the BDM connector on the target with fingers to establish better connection and to eliminate a possible source of a problem.

11 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 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.

Be careful when the CPU has register bits that are cleared on read access. Do note that when such register (memory location) is accessed either by memory/watch window or SFR window, the flags are cleared and the application may behave different when using the emulator or the target CPU. It is recommended not to display such registers or the associated memory location in the memory/watch window during final test. Otherwise, it may happen that the target application doesn't work due to the bug in the code even though it works with the emulator. For instance, the user makes a mistake and does not clear the flag in the application. Using the emulator, the application works correctly since the user uses the SFR window, which clears the flag when the window is updated.

STOP Instruction – Not supported

When stop instruction is used to force the CPU in the standby mode, the BDM ceases to work. When STOP instruction is executed, the CPU enters standby mode and all system clocks are stopped. Consequently, on-chip BDM becomes inactive too and the BDM communication falls out.

STOP instruction is controlled by S bit in the CCR register. The STOP instruction is disabled and operates like the NOP instruction, when the S control bit is set.