Freescale MC9S12X 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

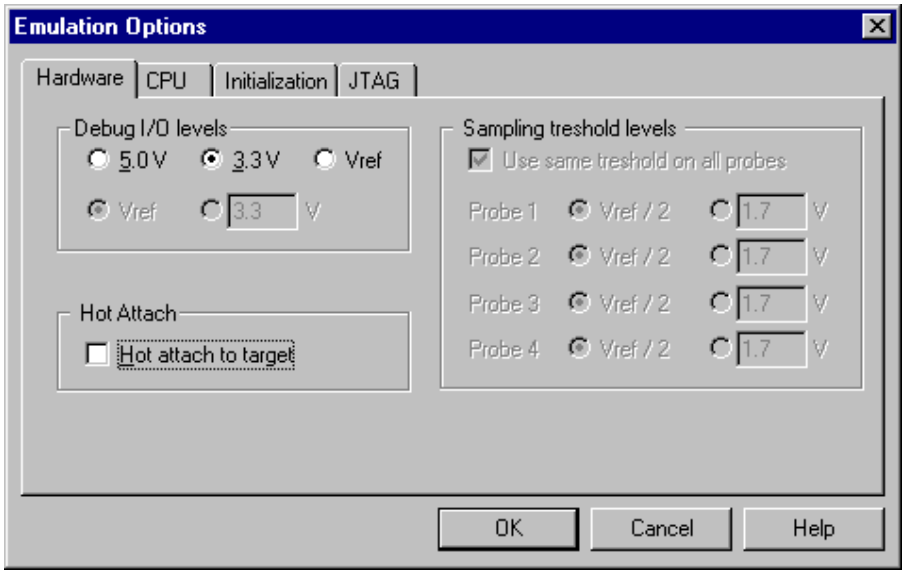
When the user's program is stopped, the CPU enters and operates in the BDM mode. Note that internal peripheral timers, previously active in the user's program, remain active after the BDM mode is entered. However, interrupts are disabled when BDM mode is entered.

Debug Features:

- Four hardware breakpoints
- Unlimited software breakpoints
- Real-time access
- Access breakpoints
- Fast flash programming
- XGATE Debugging
- Hot Attach
- COP Support
- PLL Support
- Secure & Unsecure Flash
- On-Chip Trace

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!

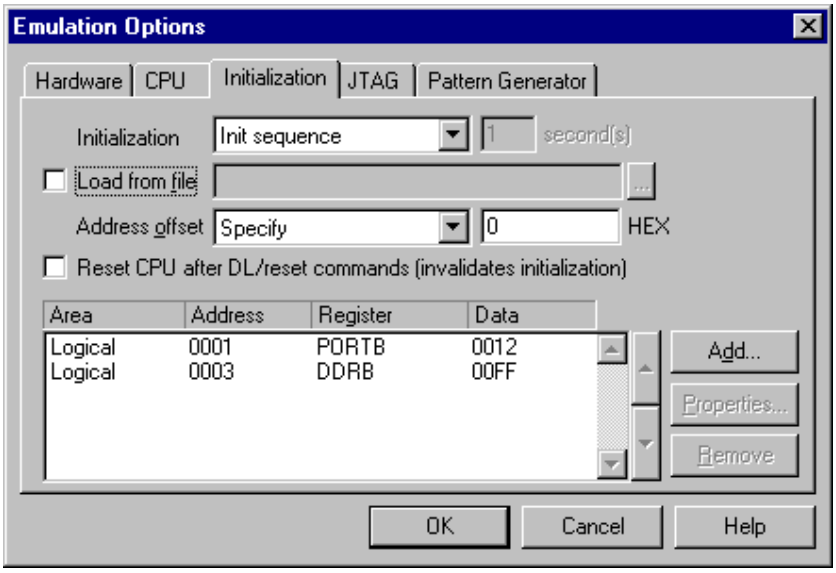
2.2 Initialization Sequence

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:

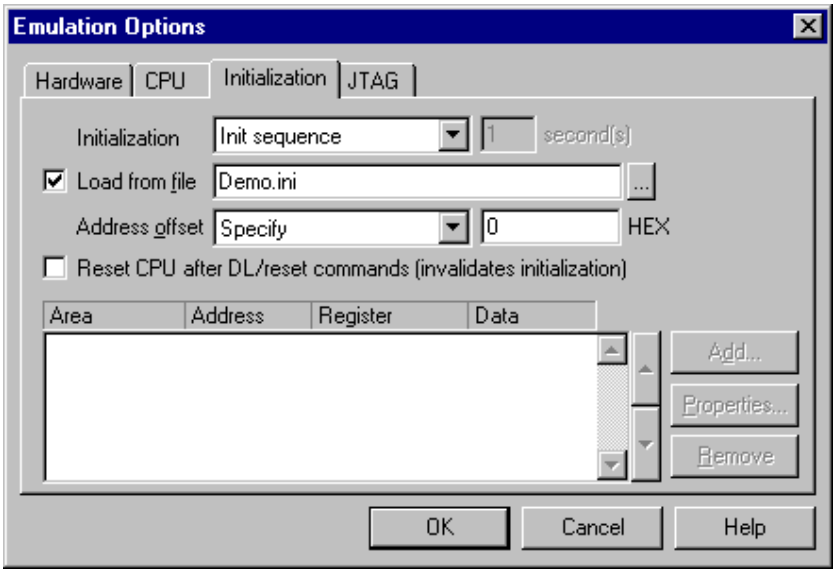
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

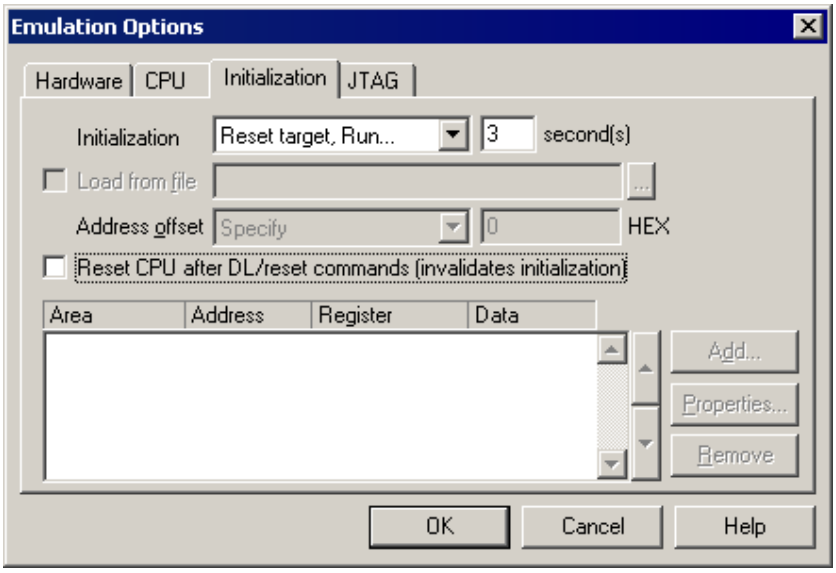
   Excerpt from the sample Demo.ini file:

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



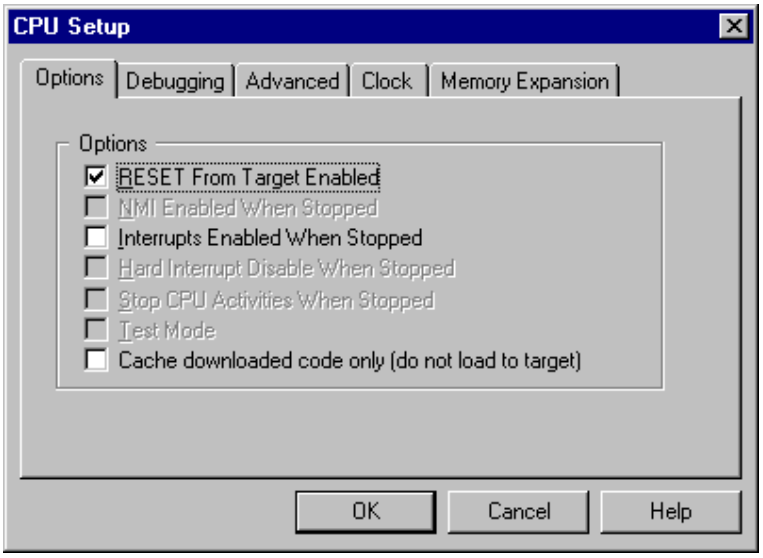
The advantage of the second method is that you can simply distribute your .ini file among different workspaces and users. Additionally, you can easily comment out some line while debugging the initialization sequence itself.

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.

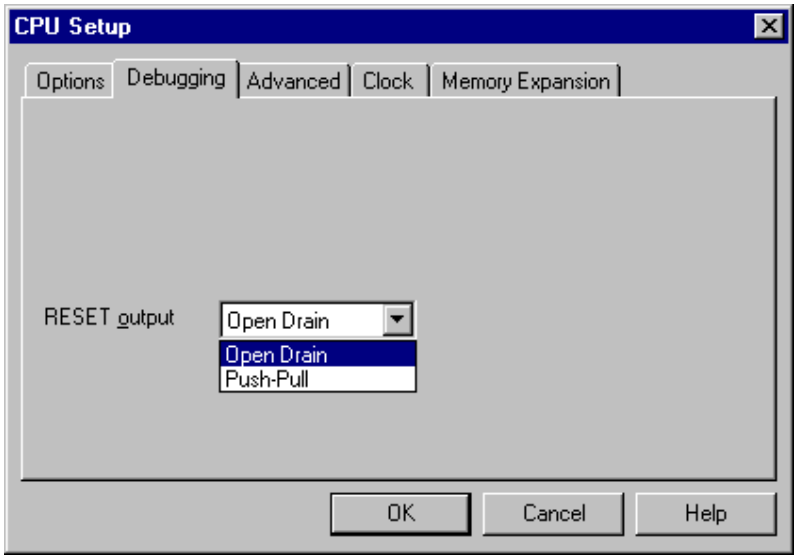
If this option is enabled, the Emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

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

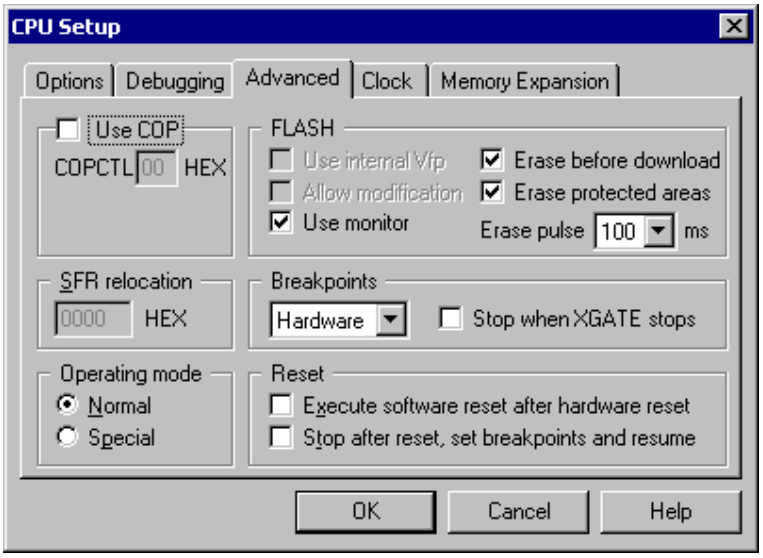
The reset line from the debugger can be driven as an open drain or as push-pull line. Typically, open drain reset signals are used, but in special cases, push-pull signals are preferred.

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



MC9S12X Advanced CPU Options

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-writeable 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.

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 CPU 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.

Using flash software breakpoints

A flash device has a limited number of programming cycles. Belonging flash sector is erased and programmed every time when a software breakpoint is set or removed. The debugger sets breakpoints hidden from the user also when a source step is executed. In worst case, a flash may become worn out due to intense and long lasting debugging using flash software breakpoints.

Stop when XGATE stops

When this option is checked S12X core is stopped whenever XGATE is stopped.

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.

Use COP

When using COP, the user must enter the COPCTL value used in the application in the 'Advanced' dialog. See 'COP Use' chapter for more details on COP use.

Use monitor

This option specifies the usage of a FLASH monitor residing in the internal RAM instead of standard flash programming through BDM port only. This option must be checked if software breakpoints are used.

Make sure you don't relocate the internal CPU RAM when programming the flash through the monitor. The software assumes default reset RAM location and allocates flash programming monitor adequately.

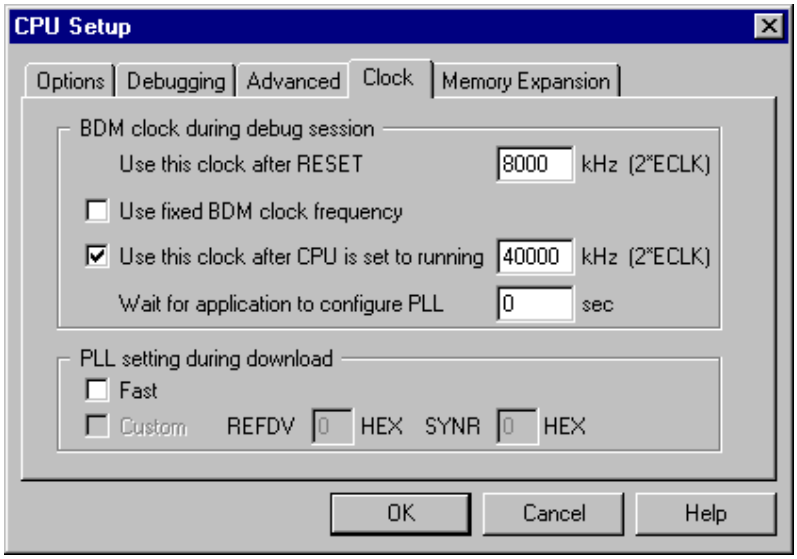
Erase before download

If this option is selected, the FLASH will be erased before every download.

Erase protected areas

If this option is selected, the protected areas in the FLASH will be erased too.

3.4 Clock options



Use this clock after RESET

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

Use fixed BDM clock frequency

When this option is checked, the CPU is configured not to change BDM clock when PLL is engaged.

Use this clock after CPU is set to running

If the PLL is engaged, BDM clock frequency is changed too. In order to keep the BDM communication synchronized with system clock, new clock (after the PLL is enabled) must be defined here, which will be used by the debugger after the CPU is set to running. See ‘PLL Use’ chapter for more details on how to use this option.

The user can alternatively use ‘Use fixed BDM clock frequency’ option which yields more predictable debug behaviour but results in debug performance not as good as when the BDM clock is equal to the PLL clock.

Keep this option unchecked if the PLL is not used.

Wait for application to configure PLL

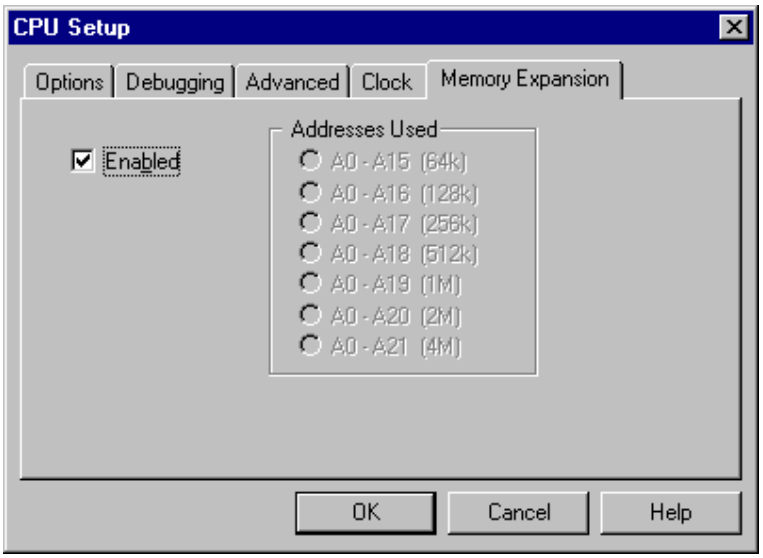
Specifies the time to wait for the application to configure PLL in seconds before the second (running) clock is applied.

PLL Setting during download

The debugger can preset PLL to maximum CPU frequency just before the debug download respectively flash programming starts. This will increase flash programming speed.

To speed up the BDM communication, the PLL settings can be set to Fast, if a PLL filter is present in the target. If there are problems with the automatic algorithm calculating the REFDV and SYNR values, those two values can be inserted manually by checking the Custom option. For more information about REFDV and SYNR, please refer to the CPU manual.

3.5 Memory Expansion Options



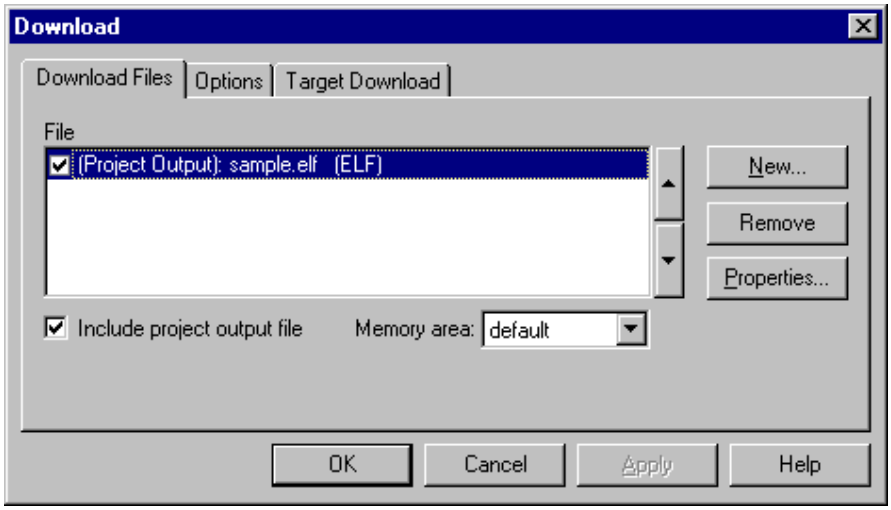
Enabled

Check this option in case of a banked application.

4 Download

A code residing in the internal flash is downloaded using standard debug download. There is no need to invoke the flash programming dialog, which was the case with HC12 and HCS12 families. The debugger takes care of the entire CPU configuration to program the file in the flash on Download debug command.

Add files to be loaded in the internal flash in the 'Download Files' tab. Download debug command programs all listed files matching with the flash memory area in the internal flash.



winIDEA 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. Uncheck the option, if it's a non-bank project.

Flash protection registers are located in a memory area from 0xFF00 to 0xFF0F. It's recommended not to load anything in this area to avoid startup problems with the download. When loading the data in this area, make sure it's correct; otherwise the download can fail due to protecting flash areas when programming wrong data in the 0xFF00-0xFF0F memory area.

Download in the internal flash can be sped up by using on-chip PLL. Programming the code into the internal flash is actually executed by the CPU, executing a special flash programming monitor in the internal CPU RAM. Thereby, flash programming monitor execution depends on the system CPU clock. If the 'Use fast PLL setting during download' option in the 'Hardware/Emulation Options/CPU Setup/Clock' tab is checked, the debugger presets PLL to the maximum CPU clock and then programs the flash. After the flash programming is completed, the debugger resets the CPU to put the CPU back in the reset state.

If presetting PLL to the maximum CPU frequency fails, the debugger programs the flash at CPU default frequency without warning the user. PLL may fail to lock at maximum CPU frequency due to the target PLL loop filter not being present or simply not designed for the maximum CPU frequency.

Standard debug download must be used also when programming internal EEPROM, internal RAM and internal data flash (S12XE). Note that it's user's concern to configure the CPU properly before the download. The debug download itself performs only a regular memory write accesses except for the internal program flash area, for which the debugger configures all the necessary registers and executes flash programming algorithm. For

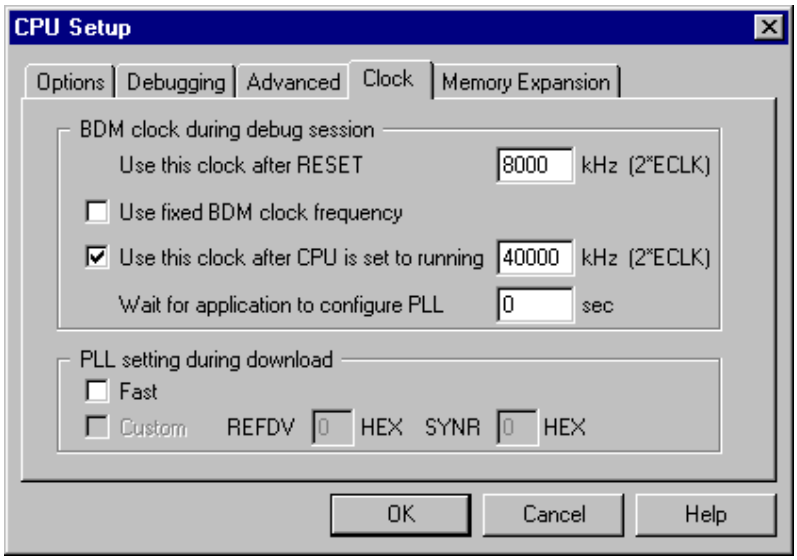
instance, for the internal EEPROM, an initialization sequence must be used through which the debugger configures ECLKDIV register (according to the CPU clock) before the download takes place. In case of problems related to the internal EEPROM, check the CPU configuration that EEPROM writes are not disabled.

5 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.

Troubleshooting

First, check the PLL filter Make sure it's implemented and has proper values. Note that PLL loop filter values depend on the reset and operating frequency.

6 COP Use

The debugger needs no watchdog awareness while the watchdog is disabled. Following explanation is irrelevant for applications not using COP.

When using COP, it's expected to be serviced properly by the application.

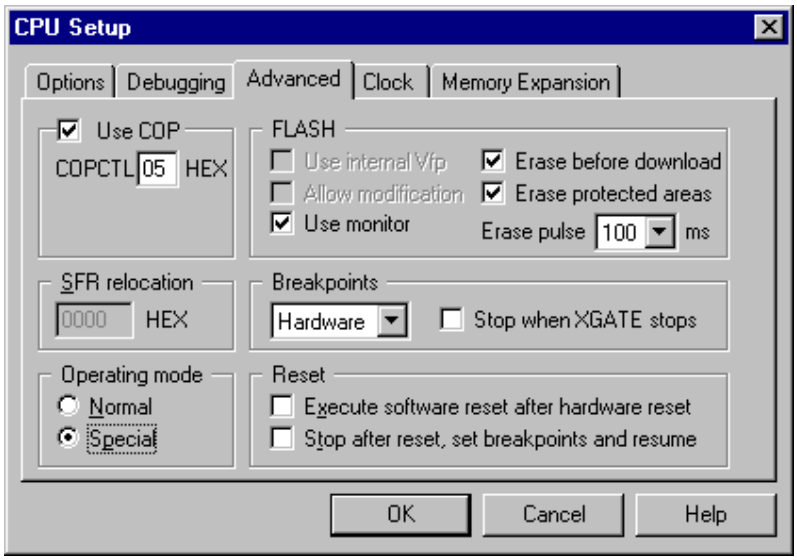
When the debugger stops the CPU for the first time after reset (system initialization), a specific CPU register (COPCTL) must be written properly to enable COP support during debugging. When the user's program is stopped during the debugging, the CPU enters BDM mode in which the COP timer must be stopped.

'RESET From Target Enabled' option in the 'CPU Setup/Options' tab must be checked when COP is used in the application. If it's not checked, the debugger cannot detect any reset source including COP.

- Normal operating mode

In normal mode, COP is disabled 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, which then enters in the active BDM mode. After the CPU enters in the active BDM mode for the first time, RSBCK bit in the COPCTL register is set first. The user must enter the COPCTL value used in the application in the 'Advanced' dialog.

Do note that the user's application must not write COPCTL register before the application is stopped for the first time by the debugger. This is necessary due to the reason that the COPCTL is write once register. Therefore, at least within 500us after reset, the application must not write to the COPCTL register or it must write the same value as the debugger (RSBCK=1).



- Special operating mode

In special mode, COP is disabled after reset and the CPU is stopped immediately.

If the application uses COP, the ‘Use COP’ option must be checked and COPCTL value from the application must be written in the COPCTL field.

After reset, the debugger writes a value defined in the COPCTL field to the COPCTL register and additionally sets RSBCK bit in the same register. Setting RSBCK bit stops the COP and RTI counters whenever the CPU is in Active BDM mode and thus assures proper operation of COP even when the application is being debugged.

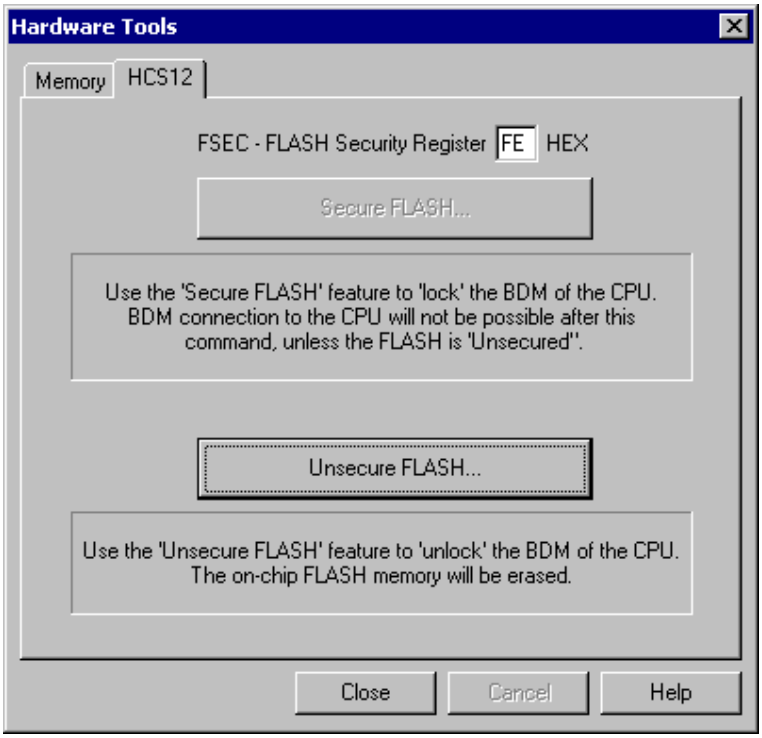
7 Secure & Unsecure FLASH

S12X family features mechanism, which locks BDM if the flash is secured. This mechanism was built into the CPU to prevent memory read over the BDM by unauthorized person and thus protecting the code (intellectual property) in the internal flash.

Secured flash can be unsecured by BDM debugger, which executes special unlocking sequence. Note that FLASH is erased when it’s unsecured. There is no way to read the code from the internal flash after it was secured.

Flash is unsecured by pressing the 'Unsecure FLASH' button in the ‘Hardware/Hardware Tools/HCS12’ tab.

The BDM/FLASH can be locked by pressing the 'Secure FLASH' button. The value of FSEC – FLASH Secure register must be specified. After the FLASH is secured, BDM is locked and unavailable after next CPU reset.



S12X Secure/Unsecure FLASH

8 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.

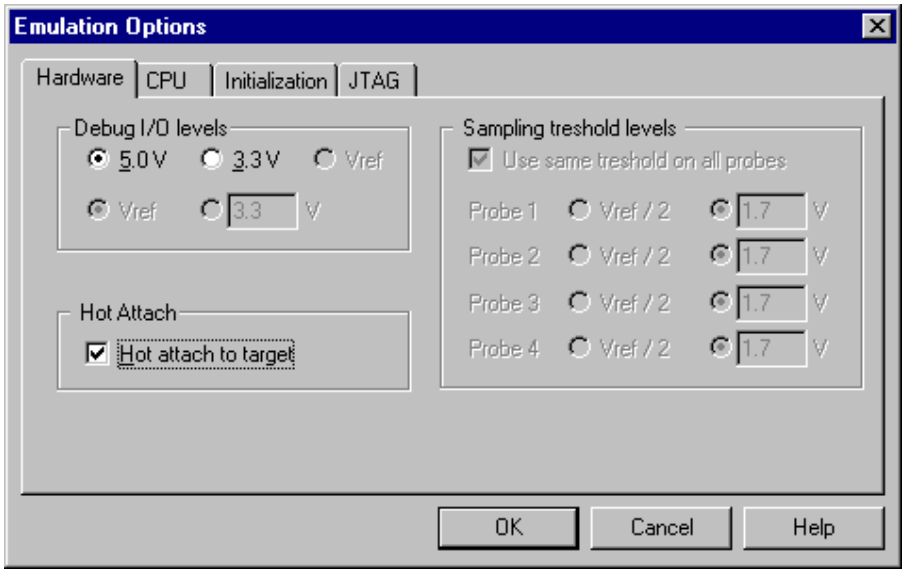
Please refer to the Software User's Guide for more information on Real-Time watches.

9 Hot Attach

MC9S12X 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.

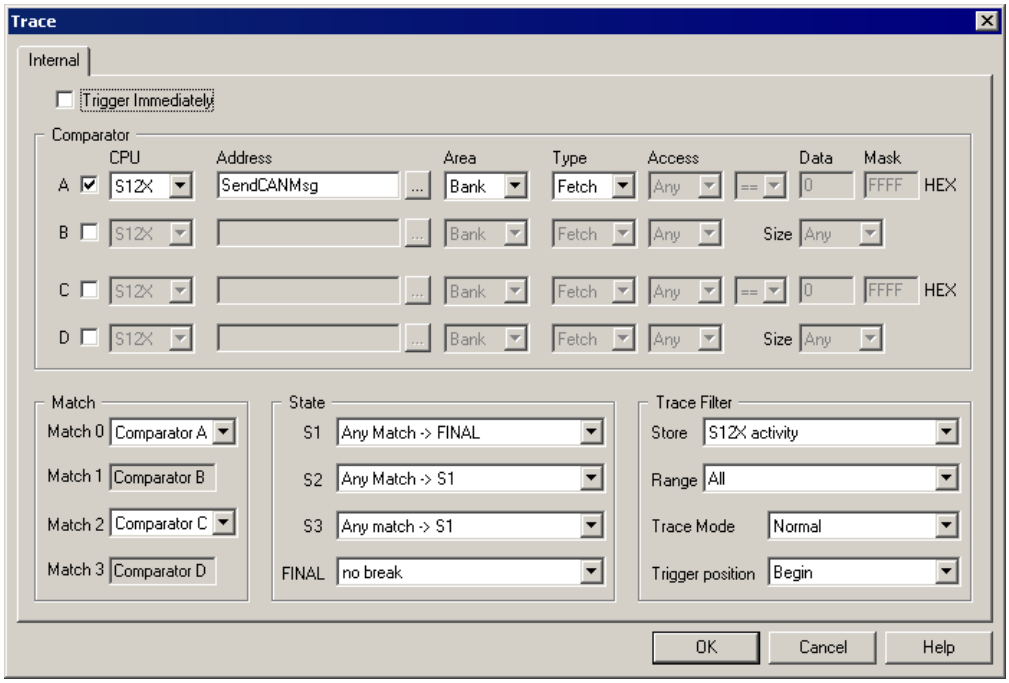
Select 'Detach' debug command in the 'Debug' menu to disconnect from the target application. If the CPU was stopped before detach, it will be set to running.

When running with Hot Attach, make sure that your program sets the RSBCK bit of the COPCTL register if the application is using COP. This way, the COP counter is disabled when the CPU is stopped.

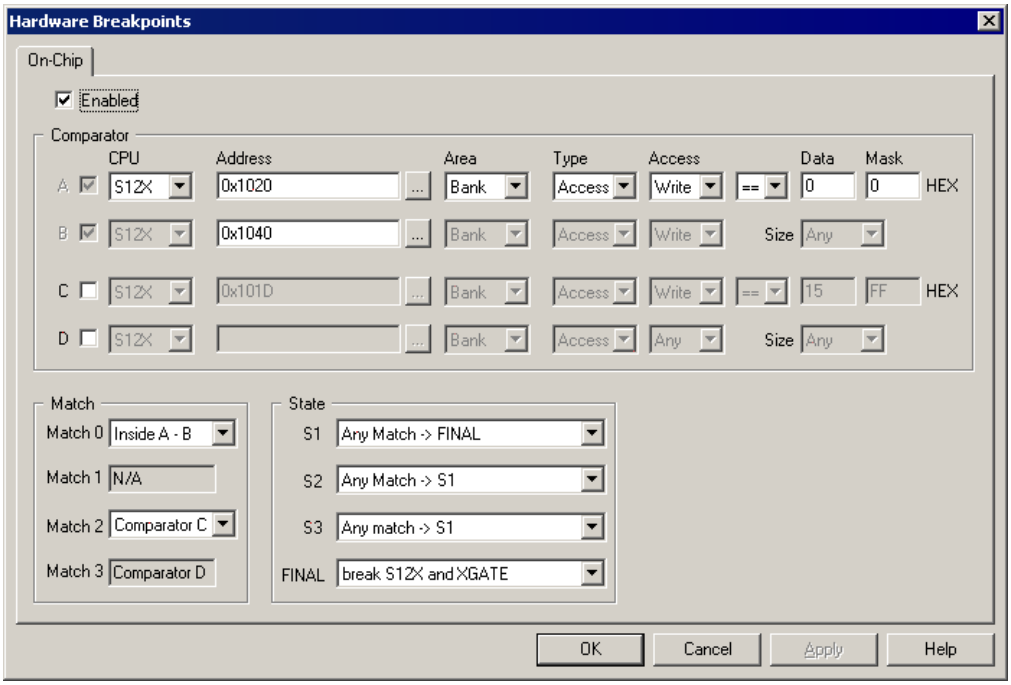
Note: Hot Attach function cannot be used for any flash programming or code download!

10 Access Breakpoints and On-Chip Trace

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



On-Chip Trace Trigger configuration



Access Breakpoints configuration

Four hardware comparator combinations are available, including a match and state machine. For each comparator, the appropriate core (S12X or XGATE) can be selected and the address, area, type, access type, data and mask can be specified.

10.1 On-Chip Trace

The trace buffer operating as FIFO is a 64 lines deep by 64-bits wide RAM array, which can record X12 and XGATE program changes.

Trace uses a branch-trace mechanism. Then it's up to debugger to interpolate the program trace for sequential instructions from a local image of program memory contents. In this way, the debugger can reconstruct the full program flow. Trace can record 128 program changes when recording only X12 or only XGATE activity and 64 program changes can be recorded when recording both, X12 and XGATE activity.

Reconstructed program normally contains over 1000 code instructions, which may sound a lot but it's nothing comparing to the trace available on high-end in-circuit emulator.

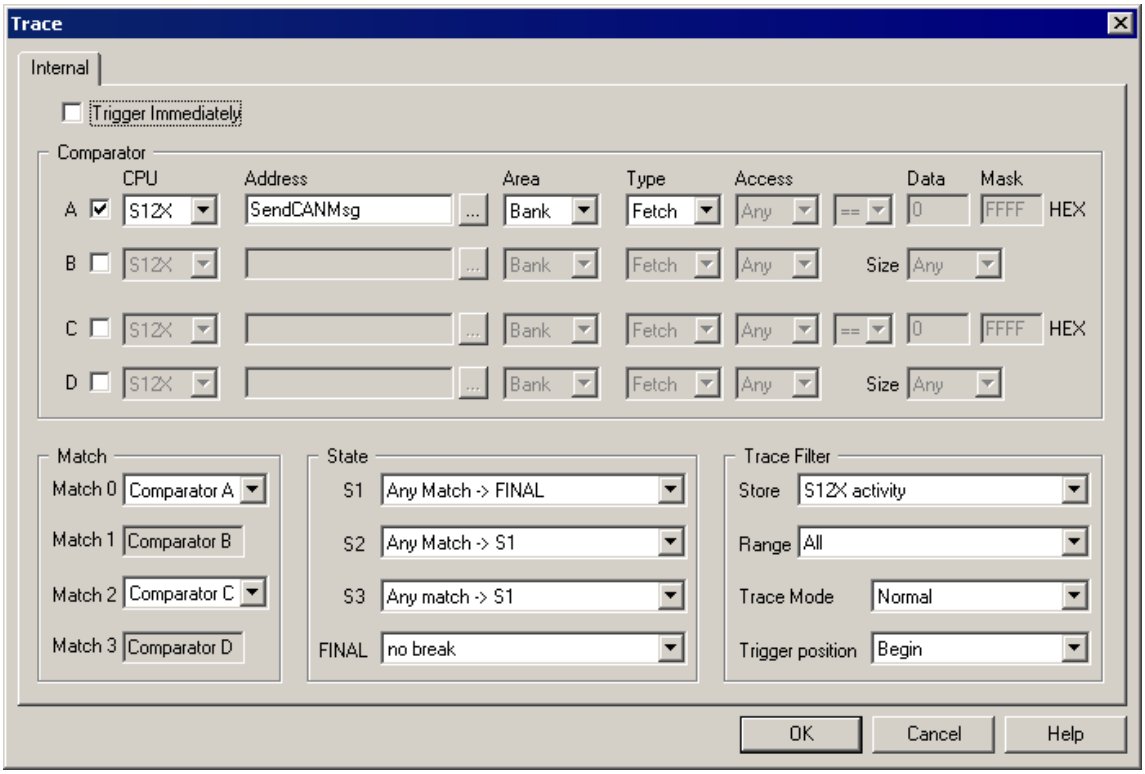
Trace can record in three modes:

- Normal mode records all code fetches when the program fetches
- Loop mode records all program fetches except if there are two or more consecutive
- Detail mode records 64 data accesses. Each data access is identified by address, data, and access type (RD/WR). It cannot be found out which code caused each data access since no code fetches are recorded.

Pretty complex events (state machine) can be configured for the trigger event. Depending on whether the user needs to look forwards or backwards, the trigger can be located at the beginning, at the end or in the middle of the trace buffer.

Frame 0 in the trace points to the first program change after the trigger event. Thereby, the trigger event itself is forward from the frame 0. Moreover, the trigger event itself may even not be visible in the trace record. For instance, if a trigger is set on a data access and trace records code fetches (normal mode), no data accesses and thus no trigger event are recorded.

Trace configuration dialog entirely follows the trace description in the CPU datasheet and there should be no indistinctness on how to set up a trigger after reading the CPU datasheet.



Trace configured to trigger on `SendCANMsg` execution and to record S12X activity only (no XGATE).

Trace Filter

The settings in this section are used to filter out unwanted results.

Store

- S12X activity – stores only S12X activity
- S12X and XGATE activity – stores S12X and XGATE activities

Range

Specifies the range of data to be saved.

- All – all data is saved
- From 0x000000 to Comparator D – all data from the beginning to the address specified in Comparator D is saved
- From Comparator C to 0x7FFFFF– all data from the address specified in Comparator C to 0x7FFFFF is saved
- From Comparator C to Comparator D – all data between the addresses specified in Comparator C and Comparator D is saved

Trace Mode

Defines the Normal capture (default), the Loop 1 or Detail capture, as per Freescale specifications.

In Normal mode, change of flow (COF) addresses will be stored.

Loop 1 mode, similarly to normal mode also stores only COF address information to the trace buffer, it however allows filtering out of redundant information. The intent of Loop 1 mode is to prevent the trace buffer from being filled entirely with duplicate information from a tight looping construct such as delays using dbne instruction or polling loops using BRSET/BRCLR instructions. It will prevent duplicate entries in the trace buffer resulting from repeated bit-conditional branches.

In the Detail mode, address and data for all memory and register accesses is stored in the trace buffer. In the case of XGATE tracing this means that initialization of the R1 register during a vector fetch is not traced. This mode is intended to supply additional information on indexed, indirect addressing modes were storing only the destination address would not provide all information required for a user to determine where his code was in error.

When tracing CPU activity in detail mode, all cycles except those when the CPU is either free or opcode fetch cycle. In this mode the XGATE program counter is also traced to provide a snapshot of the XGATE activity.

Trigger Position

- Begin

  Trace buffer starts recording after the trigger event and continues until 64 locations are filled. Trigger event itself is not visible in the trace record unless the trigger was set on the branch instruction. No program execution before the first branch instruction, being recorded after the trigger event, can be seen in the trace window.

- Middle

  As soon as the trace is started, the trace buffer starts collecting the data. When the trigger condition is met, another 32 lines will be traced before ending the trace session, irrespective of the number of lines stored before the trigger occurred, then the trace buffer is disarmed and no more data is stored. T

  The trace buffer requires to be filled with 32 locations before the trigger in order to read the collected information. For example, if the trace buffer doesn't collect 32 branches before the trigger, no trace results can be displayed. User should make sure that enough code (generating 32 changes of flow) is executed before the trigger in order to use Middle trigger position.

- End

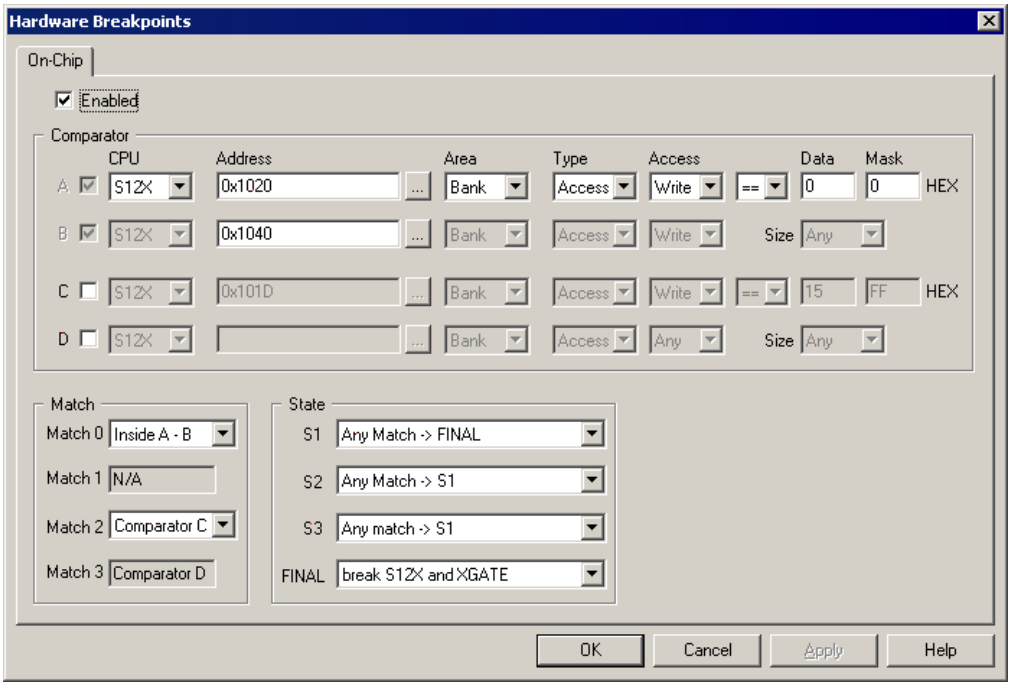
  Trace buffer stops recording with the first branch before the trigger event. Additionally, the trace buffer requires to be filled with all 64 locations before the trigger in order to read the collected information. For example, if the trace buffer doesn't collect 64 branches before the trigger, no trace results can be displayed. User should make sure that enough code (generating 64 branch addresses) is executed before the trigger in order to use End trigger position. Note also that the trigger event itself is not visible in the trace record unless the trigger was set on the branch instruction. No program execution can be seen in the trace window after the branch instruction, being lastly filled in the trace buffer before the trigger event.

10.2 Examples

Following examples demonstrate access breakpoints use. Note that the same samples apply for the trace trigger too. Trace Trigger and Access Breakpoints dialog are basically the same since the same on-chip debug resources can be used to break the CPU (access breakpoint) or trigger the trace.

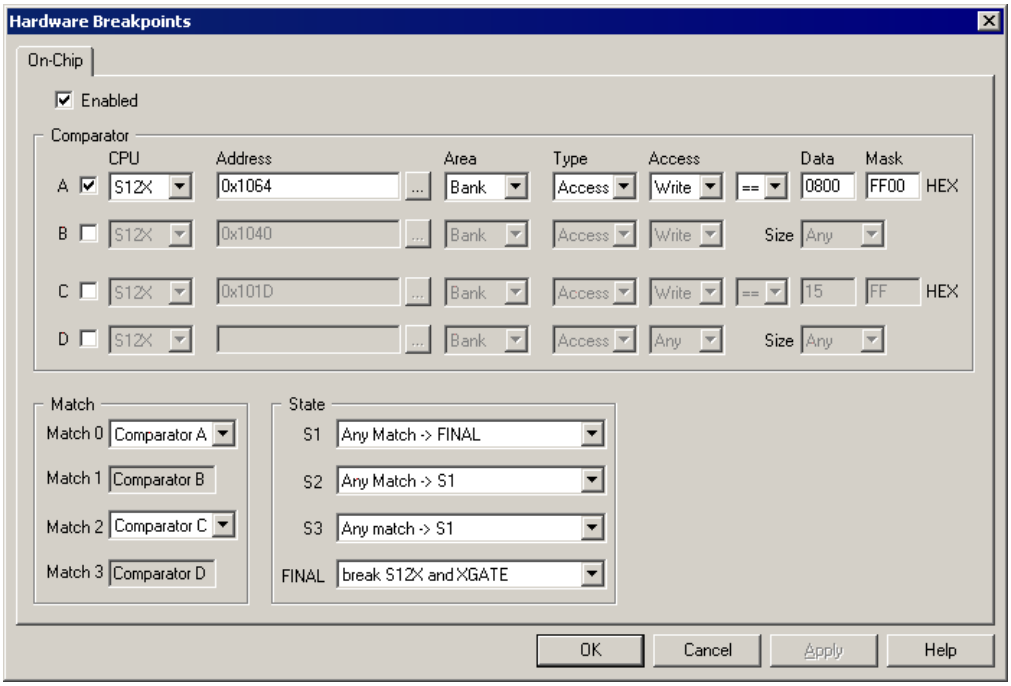
Example 1: The application writes to the 0x1020-0x1040 memory range although it shouldn't.

Following Access Breakpoints configuration will stop the application on first CPU write into the memory range from 0x1020 to 0x1040. Setting Mask to 0 ignores specific Data value.



Example 2: The application gets an unexpected 0x08 byte written at address 0x1064.

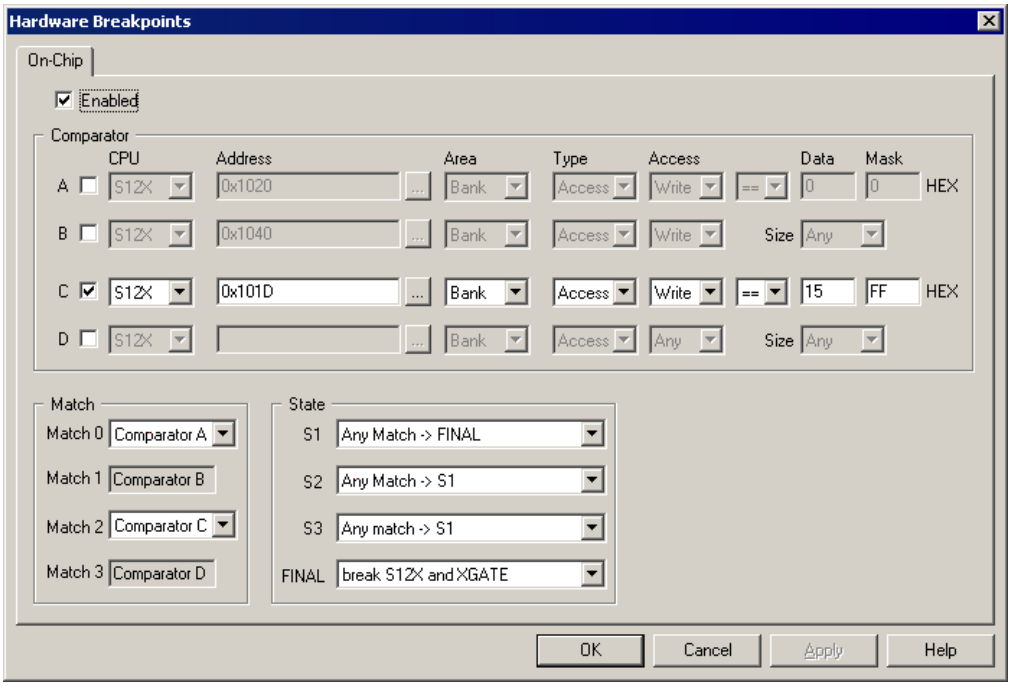
Following Access Breakpoints configuration will stop the application on first CPU byte write of value 0x08 to the address 0x1064.



Example 3: I'm getting an unexpected 0x15 byte written at address 0x101D.

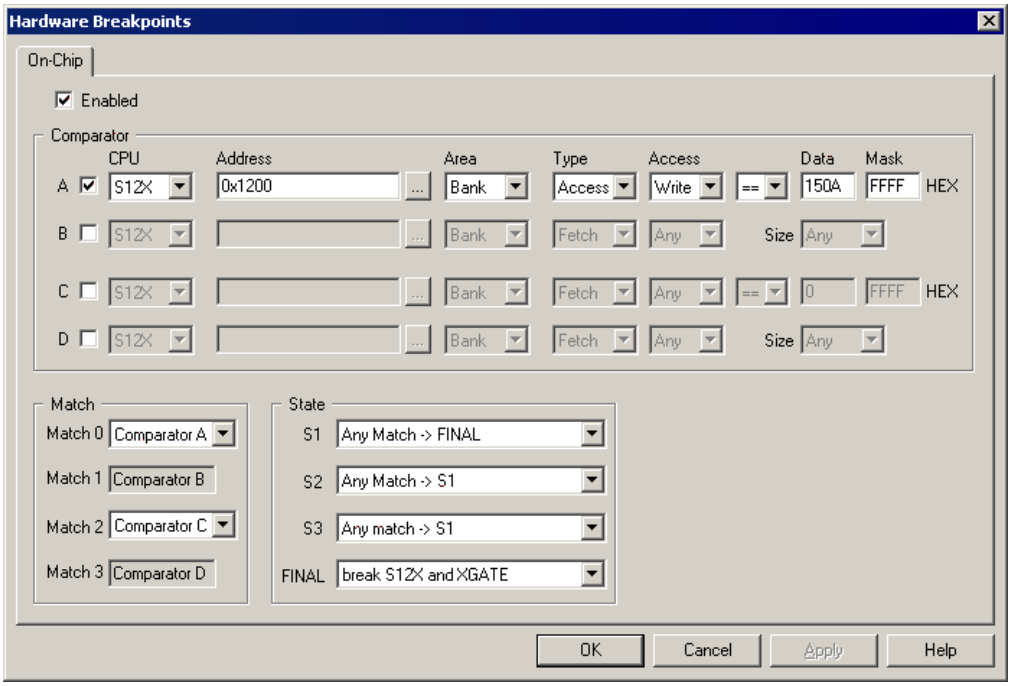
Example 3 differs from the example 2 in address alignment. Note that the comparator configuration is different when setting a breakpoint (or trace trigger) on odd or even address.

Following Access Breakpoints configuration will stop the application on first CPU byte write of value 0x15 to the address 0x101D.



Example 4: I'm getting an unexpected 0x150A word written at address 0x1200.

Following Access Breakpoints configuration will stop the application on first CPU word write of value 0x150A to the address 0x1200.

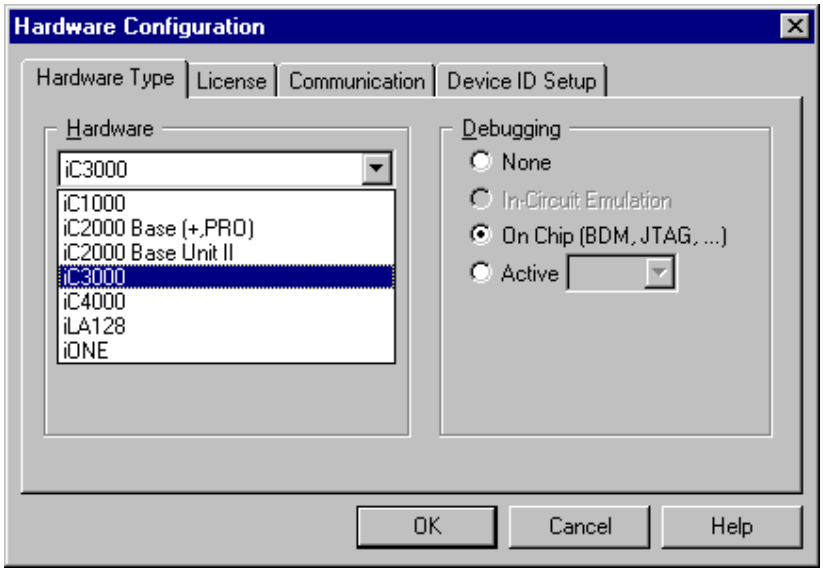


11 Getting Started

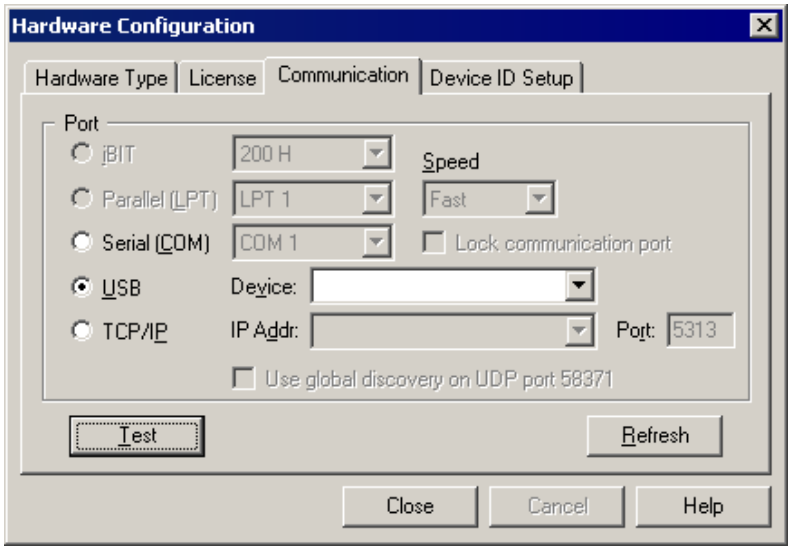
Quick start instructions help the user to start debugging a single-chip application in a very short time.

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…'

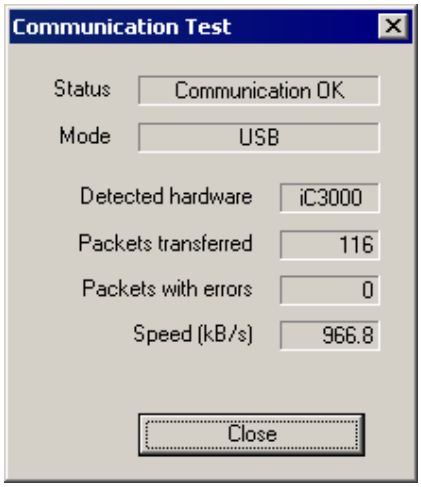


- Select 'Communication' tab within the same dialog and select the communication type that is going to be used.

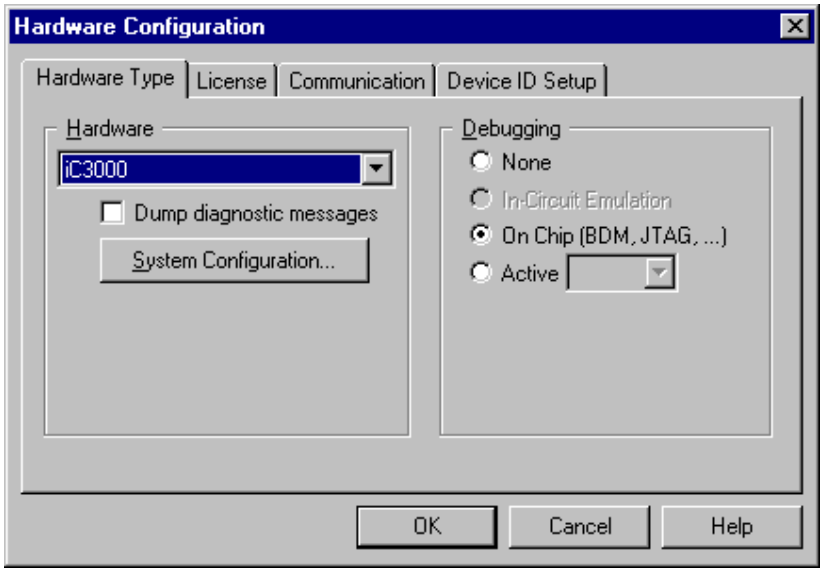


Refer to 'Setting up Communication' document (delivered beside the emulator) for more details on how to configure each of the supported communications.

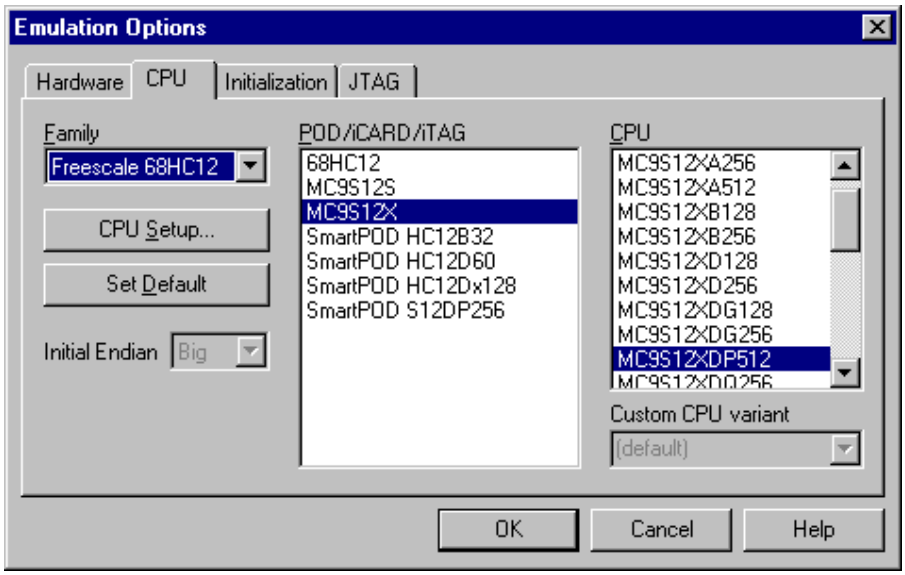
- Execute communication test by pressing ‘Test’ button. There must be no packets with errors reported during the communication test.



- 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.

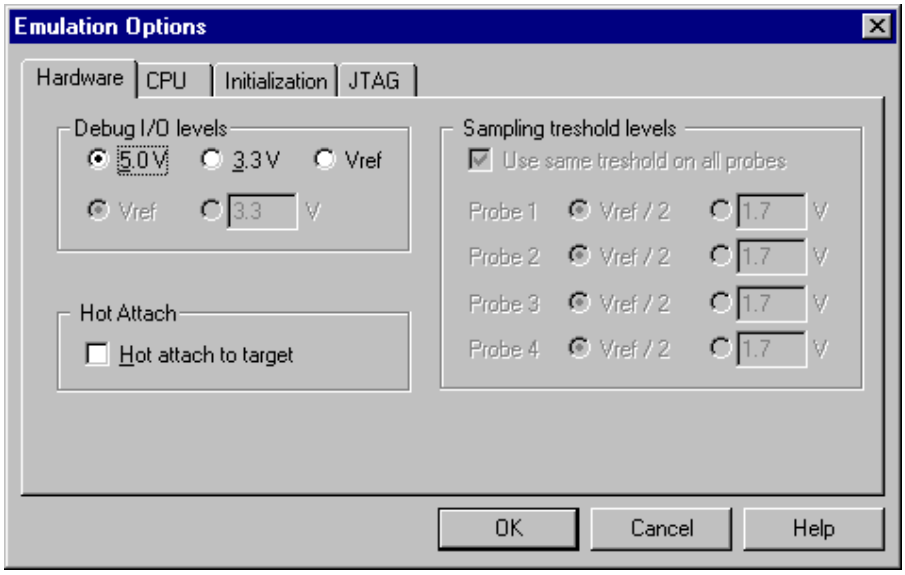


- Make sure that the 'Hot attach to target' option is unchecked in the 'Hardware/Emulation Options/Hardware' tab.

- 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

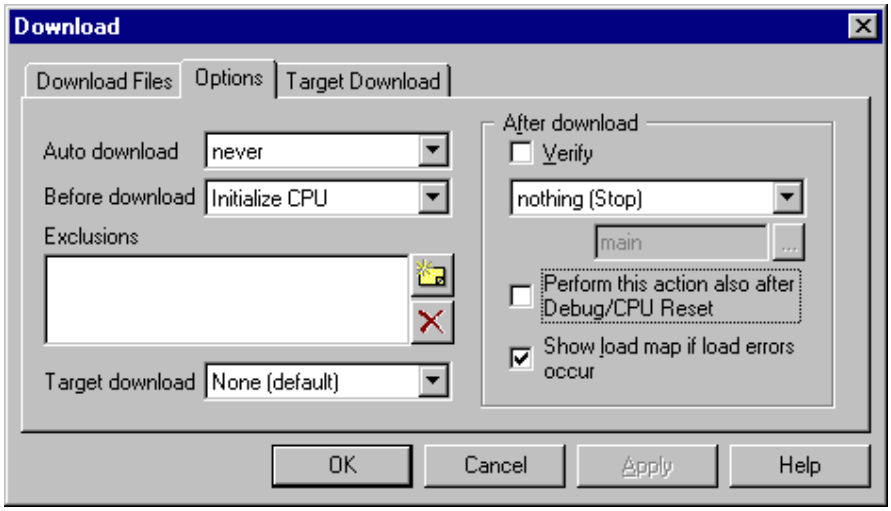
There should be no other reset sources, which could disturb BDM communication. Remove all capacitors and other reset logic from the reset line and try again. Make sure that the 'RESET from target enabled' option is checked in the 'CPU Setup/Options' tab.

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

Download

With old HCS12 CPUs, it was necessary to invoke the 'FLASH Programming Setup' dialog (FLASH/Setup….) to program the flash. In case of S12X the 'FLASH Programming Setup' dialog is not required. Flash programming process is hidden from the user and executed during standard debug download.

- Add the file that you'd like to download in the flash in 'Debug/Files for Download/Debug Files' tab.
- Check whether 'After download Stop' option is selected in the Debug/Files for download/Options' tab. Check 'Verify' option'



- Next, check whether 'Use monitor' and 'Erase before download' options in the 'CPU Setup/Advanced' dialog are set.

- Execute 'Download' debug command.

The debugger should not report any verify errors on verify after the download and the CPU should stop at the address where the reset vector points.

Troubleshooting

If download fails, verify

- If proper CPU is selected in winIDEA

- 'Hot Attach' option is unchecked in the 'Hardware/Emulation Options/Hardware' tab

If verify errors occur, check

- if errors occur in the flash protected areas only. Try to check the 'Erase protected areas' in 'CPU Setup/Advanced' dialog and see if it makes any difference.

- winIDEA 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. Wrong addressing can result in download errors.

- 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. Uncheck the option if it's a non-bank project. Wrong setting can result in download errors.

- Flash protection registers are located in a memory area from 0xFF00 to 0xFF0F. It's recommended not to load anything in this area to avoid startup problems with the download. When loading the data in this area, make sure it's correct; otherwise the download can fail due to protecting flash areas when programming wrong data in the 0xFF00-0xFF0F memory area.

Debugging X12

- Check 'Stop when XGATE stops' option in the 'CPU Setup/Advanced' dialog.

- Execute ‘Download’ debug command.

The program should be stopped. Set a breakpoint on main and run the application. It should hit the breakpoint. Next, try to step in the disassembly window first and then in the source window. Next, run the application and then set a breakpoint in the program, which is executed. The program should stop at the breakpoint.

Troubleshooting

If any of the above test fails, please

- Read thoroughly ‘PLL Use’ chapter.
- Read thoroughly ‘COP Use’ chapter.
- Double-check the BDM connection. Try to press the BDM connector in the target with fingers to establish better connection and to eliminate possible source of the problem.

It’s assumed that X12 program can be debugged at this point with no problems. Let’s call this X12 winIDEA session.

Debugging XGATE

It’s assumed that the application uses XGATE. Select Debug -> Core -> XGATE and new winIDEA will open. Select ‘Connect’ debug command from the ‘Debug’ menu in XGATE winIDEA session. If X12 program is stopped at this moment, XGATE session should display IDLE. Valid XGATE statuses are IDLE and RUN for active XGATE.

Finally, run the X12 program. X12 winIDEA should display run status and XGATE winIDEA should display run status whenever there is some activity on XGATE. Since status is updated approximately three times per second and XGATE routine may end very quickly, it may happen that the XGATE status will be idle all the time. It’s recommended to add the XGCHID register in real-time watches. Read ‘Real-time access’ section for more details on how to update SFRs in real- time watches. During this test, 100ms should be set for real time access update period.

When XGATE is active, XGCHID register has different value than zero. Each X12 interrupt source has unique XGATE channel ID and XGCHID register shows the interrupt source serviced by XGATE.

12 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.

- Question: What is the common value of the pull-up for the BGND line? Is it good to add a capacitor? Answer: It is not allowed to add capacitor to the bidirectional open-drain line. There must be no capacitor on the BGND and the RESET line. Target pull-up on the BGND line should have a value between 4k7 and 10k ohms because the debug iCARD has 1k ohms pull-up already.