
Technical Notes

Freescal 68HC12 Family In-Circuit Emulation

Contents

Contents.....	1
1 Introduction	2
1.1 Differences from a standard environment	2
1.2 Common Guidelines.....	2
1.3 Port Replacement Information	2
2 Emulation Options.....	3
2.1 Hardware Options	3
2.2 CPU Configuration	4
2.3 Power Source and Clock	5
2.4 Initialization Sequence	6
2.5 Synchronisation of Two or More Emulators	8
2.6 Pattern Generator	9
3 Setting CPU options	11
3.1 CPU Options	11
3.2 Advanced In-Circuit Emulation Options.....	13
3.3 Advanced Active Emulation Options.....	15
3.4 Memory Expansion	17
3.4.1 Memory Expansion (68HC12A4).....	18
3.4.2 Memory Expansion (68HC12DG128).....	19
3.5 Memory Mapping	20
4 Debugging Interrupt Routines	21
5 Memory Access	22
6 Emulation Notes	22
6.1 Memory spaces on 68HC12 CPUs.....	22
6.2 Hardware breakpoints	23
6.3 Clock Setup	23
6.4 Reserved CPU Resources.....	23
6.5 PLL	23
6.6 COP.....	23
6.7 STOP Instruction.....	24
6.8 Internal CPU FLASH.....	24
6.9 'Clear on read' register bits	24
6.10 Interrupts Enabled While Stopped.....	24
6.11 Internal EEPROM or RAM download	24
6.12 Things to remember.....	24
6.12.1 Troubleshooting Execution Breakpoints	25

1 Introduction

Debug Features

- Unlimited breakpoints
- Access breakpoint
- Real-time access
- Trace
- Execution profiler
- Execution coverage

1.1 Differences from a standard environment

The In-Circuit Emulator and the Active Emulator can emulate a processor or a micro-controller. Beside the CPU, additional logic is integrated on the POD. The amount of additional logic depends on the emulated CPU and the type of emulation. A buffer on a data bus is always used (minimal logic) and when rebuilding ports on the POD, maximum logic is used. As soon as a POD is inserted in the target instead of the CPU, electrical and timing characteristics are changed. Different electrical and timing characteristics of used elements on the POD and prolonged lines from the target to the CPU on the POD contribute to different target (the whole system) characteristics. Consequently, signal cross-talks and reflections can occur, capacitance changes, etc.

Beside that, pull-up and pull-down resistors are added to some signals. Pull-up/pull-down resistors are required to define the inactive state of signals like reset and interrupt inputs, while the POD is not connected to the target. Because of this, the POD can operate as standalone without the target.

1.2 Common Guidelines

Here are some general guidelines that you should follow.

- Use external (target) Vcc/GND if possible (to prevent GND bouncing),
- Make an additional GND connection from POD to the target if the Emulator behaves strangely,
- Use the reset output line on the POD to reset the target whenever Emulator resets the CPU,
- Make sure the appropriate CPU is used on the POD. Please refer to the POD Hardware reference received with your POD.
- No on-chip or external watchdog timers can be used during emulation (unless explicitly permitted). Disable them all.
- When interrupts in background are enabled, take note that the interrupt routine must return in 25 ms, otherwise the Emulator will assume that the program is hung.

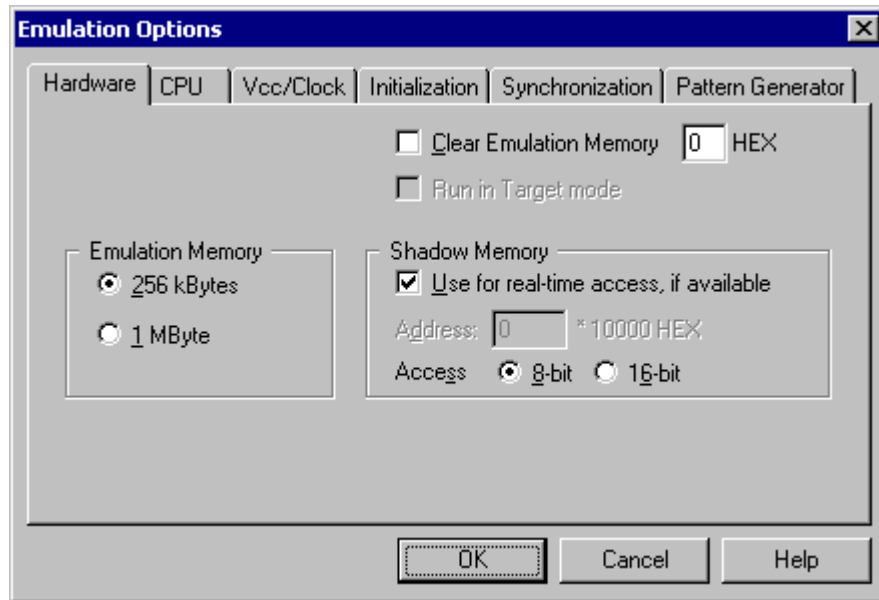
1.3 Port Replacement Information

In general, when emulating the single chip mode, some ports have to be rebuilt on the POD because original ports are used for emulation – typically ports used as address and data bus in extended mode. Special devices, so called port replacement units, provided already by the CPU vendor or other standard integrated circuits are used to rebuild "lost" ports. Rebuilt ports are logically compatible with original CPU's ports, but electrical characteristics may differ. If a special device (the port replacement unit (PRU), available from the CPU manufacturer) is available, electrical characteristics don't differ much and usually the user doesn't have to pay attention. The differences may become relevant when standard integrated circuits are used and operating close to electrical limits, e.g. when input voltage level is close to specified maximum voltage for low input level ("0") or

specified minimum voltage for high input level (“1”) or if, for example, the target is built in the way that the maximum port input current must be considered.

2 Emulation Options

2.1 Hardware Options



In-Circuit Emulator Options dialog, Hardware page

Emulation Memory

Defines the size of emulation memory available on the In-Circuit emulation module.

Note: You must specify the memory size correctly, otherwise the Emulator will not initialize.

Clear Emulation Memory

This option allows you to force clearing (with the specified value) of emulation memory after the emulation unit is initialized.

Clearing emulation memory takes about 2 seconds per megabyte, so use it only when you want to make sure that previous emulation memory contents don't affect the current debug session.

Shadow Memory

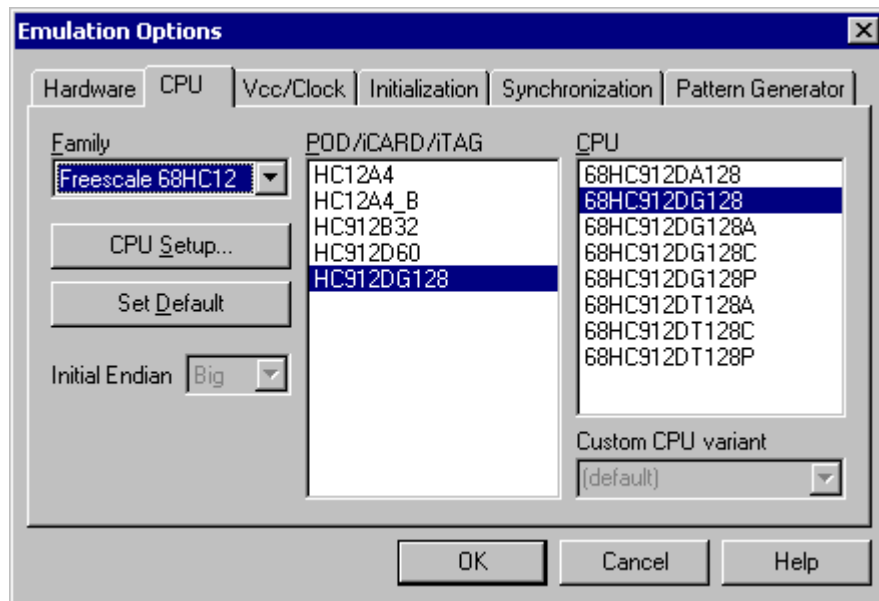
On-board shadow memory is provided that allows reading of memory without stopping the CPU or stalling it even for a single cycle. If you wish to use this memory for real-time access, check the 'Use for real-time access if available' option.

If you leave the option unchecked, the 'regular' real-time readout (if available – see "Real-Time Memory Access" on page 22) will be used for real-time access.

The 'Access' setting specifies whether 8-bit or 16-bit real-time access is used.

2.2 CPU Configuration

With In-Circuit emulation besides the CPU family and CPU type the emulation POD must be specified (some CPU's can be emulated with different PODs).



In-Circuit Emulator Options dialog, CPU Configuration page

CPU Setup

Opens the CPU Setup dialog. In this dialog, parameters like memory mapping, bank switching and advanced operation options are configured. The dialog will look different for each CPU reflecting the options available for it.

Set Default

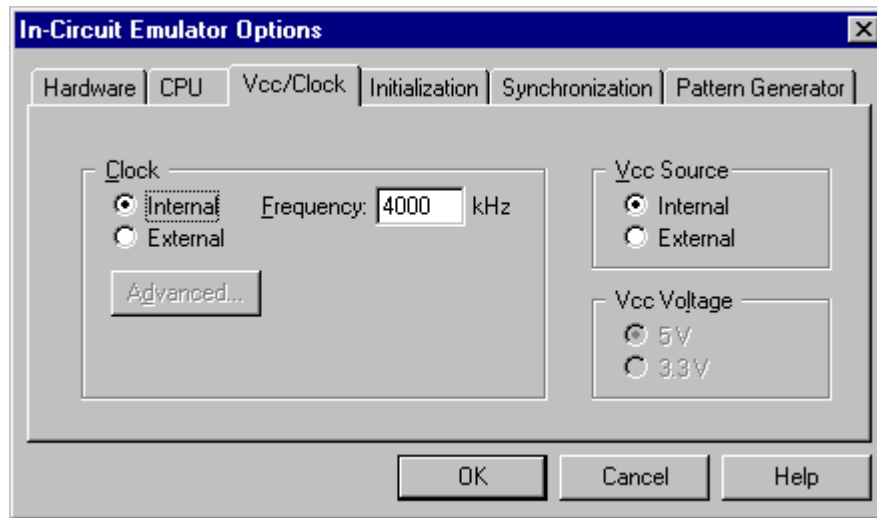
This button will set default options for currently selected CPU. These include:

- Vcc and clock source and frequency
- Advanced CPU specific options
- Memory configuration (debug areas, banks, memory mapping)

Note: Default options are also set when the Family or a POD is changed.

2.3 Power Source and Clock

The Vcc/Clock Setup page determines the CPU's power and clock source.



In-Circuit Emulator Options dialog, Vcc/Clock Setup page

Note: When either of these settings is set to External, the corresponding line is routed directly to the CPU from the target system.

Clock Source

Clock source can be either used internal from the emulator or external from the target. It is recommended to use the internal clock when possible. When using the clock from the target, it may happen that the emulator cannot initialize any more.

It is dissuaded to use a crystal in the target as a clock source during the emulation. It is recommended that the oscillator be used instead. Normally, a crystal and two capacitors are connected to the CPU's clock inputs in the target application as stated in the CPU datasheets. A length of clock paths is critical and must be taken into consideration when designing the target. During the emulation, the distance between the crystal in the target and the CPU (on the POD) is furthermore increased; therefore the impedance may change in a manner that the crystal doesn't oscillate anymore. In such case, a standalone crystal circuit, oscillating already without the CPU must be built or an oscillator must be used.

When the clock source is set to Internal, the clock is provided by the emulator and you may control its frequency in steps of 1 kHz. Depending on the Emulator and its oscillator version, you will be able to use clock from 1MHz to 33 MHz (on iC181) or up to 100MHz on iC2000 emulation units.

Note: The clock frequency is the frequency of the signal on the CPU's clock input pin. Any internal manipulation of it (division or multiplication) depends entirely on the emulated CPU.

If the clock source is set to external, the clock is provided by the target system. In certain applications, for instance, a 32.786 kHz clock is used. Since the minimal clock the Emulator can generate is 1MHz, an external clock source must be used and the clock source set to external.

Vcc Source

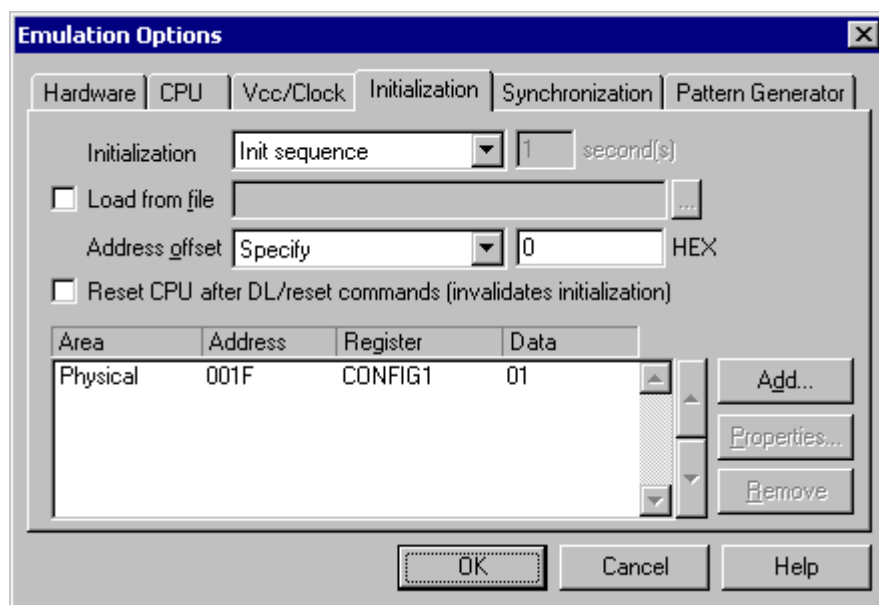
Determines whether Emulator or the target system provides power supply for the CPU.

2.4 Initialization Sequence

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

The initialization sequence can be set up in two ways:

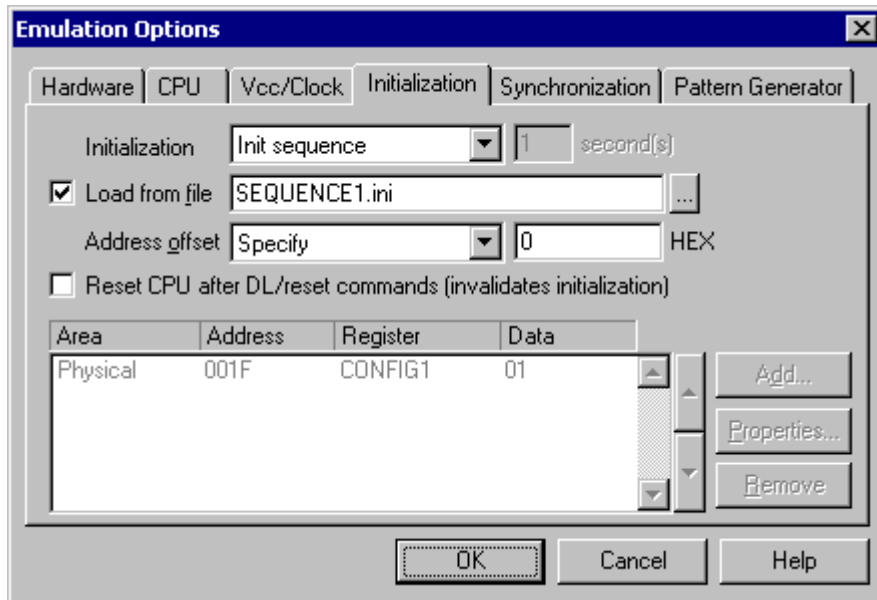
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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

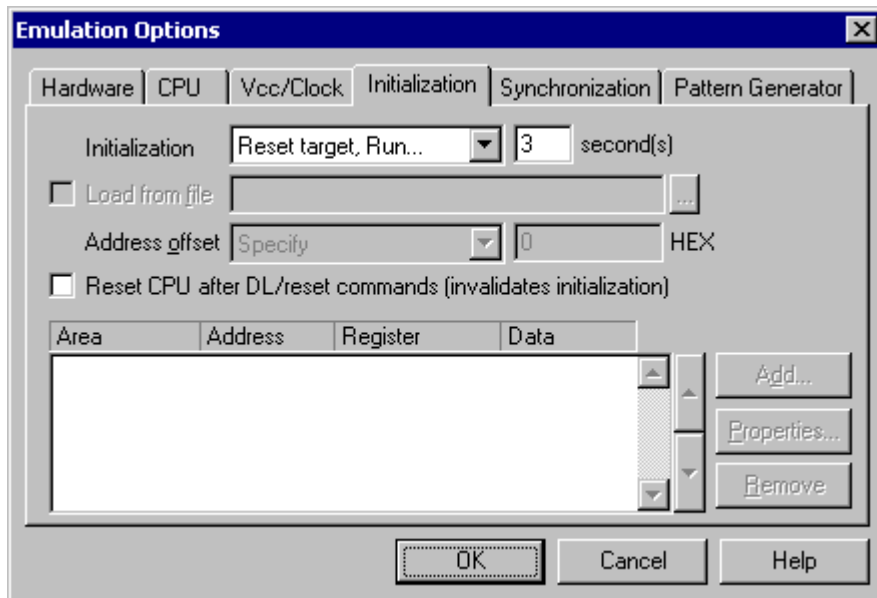
Excerpt from the sample SEQUENCE1.ini file:

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



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

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



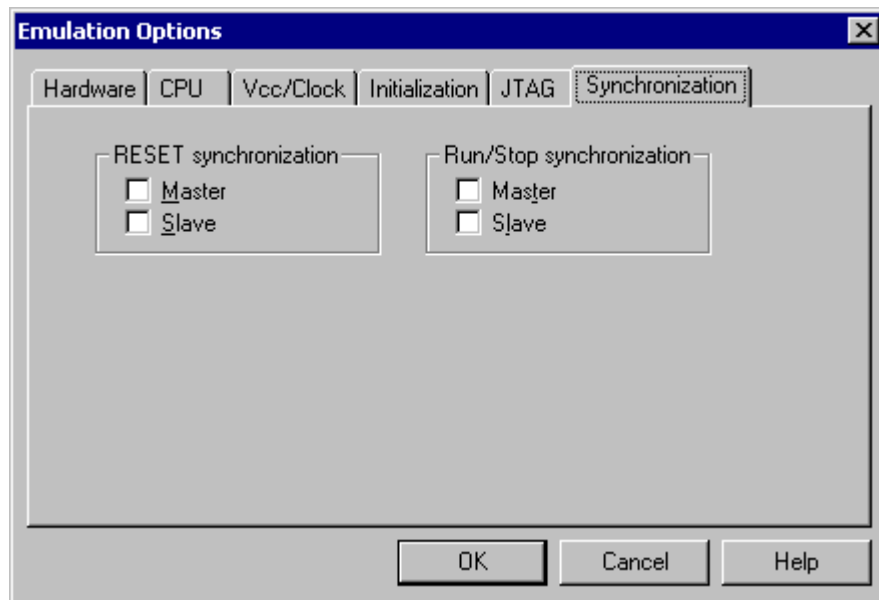
2.5 Synchronisation of Two or More Emulators

Synchronization of two or more Emulators is currently supported on all HC(S)12 PODs.

There are two special pins on the POD:

- SR - synchronous reset
- SS - synchronous stop

The corresponding pins of all Emulators must be connected together. This means that all SR pins must be connected together, and all SS pins must be connected together (but, of course, separated from each other).



Synchronization Options

The Emulator operation is specified in a special dialog, the 'Hardware/In-circuit emulation/Synchronization' tab. Both, SR and SS line can operate independent of each other as master or slave. The user must specify the operating mode in the 'Synchronization' tab.

Synchronous Reset (SR) line

- Master: when the CPU resets, the reset will also be broadcast on the SR line;
- Slave: only monitors the activity on the SR line.

Synchronous Stop (SS) line

- Master: when the program stops, this will be broadcast on the SS line;
- Slave: monitors the activity on the SS line and automatically stops or goes into running depending on the messages received on the line.

Both lines, synchronous reset and synchronous stop are open drain lines – it means that they can operate as input or output. If you define the Emulator as master, open drain line (SR, SS) operates as output and if you define the Emulator as slave it operates as input.

2.6 Pattern Generator

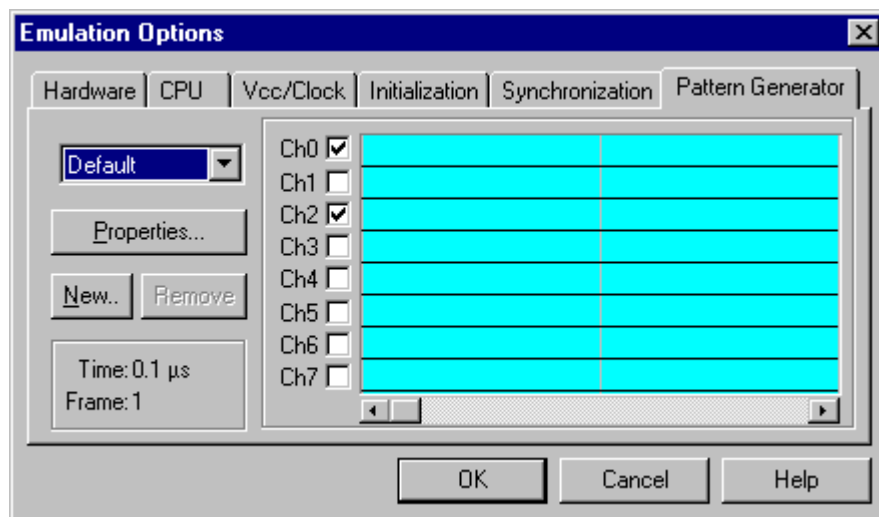
iC1000, iC2000 and iC4000 provide an 8-channel waveform programmable pattern generator capable of continuous or single shot operation at up to 10MHz-clock rate with up to 512 samples.

Note: when using the iC4000 system, it has in certain configurations two Pattern Generators: one on the base module and one on the Power Emulator module. The Pattern Generator on the base module is active when the debugging type is set to 'Active Emulation' or 'BDM/JTAG Emulation'; the Pattern Generator on the Power Emulator Module is active when 'In-circuit Emulation' is selected.

You can configure any number of patterns using 'New...' and 'Remove' buttons. The currently selected pattern is displayed in the combo box as indicated in the above figure.

State of a disabled channel can be configured either to high or low. Every individual channel can be enabled or disabled by configuring the check box next to its name. When a channel is disabled you can still configure its state, which remains unchanged throughout its period.

Waveforms are configured easily by clicking and moving the mouse cursor on the desired channel and position.



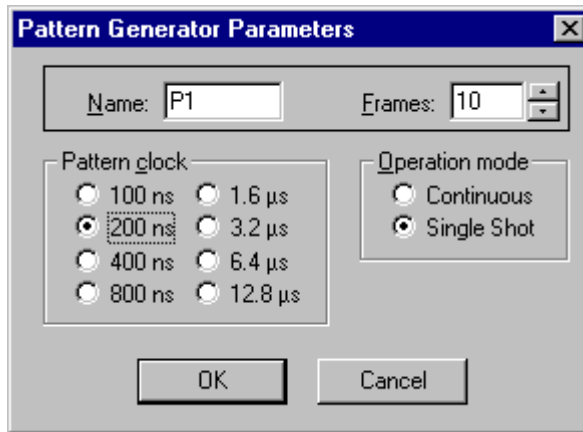
In-Circuit Emulator Options dialog, iC1000/iC2000/iC4000 Pattern Generator page

Properties

This button opens a dialog where parameters for the current pattern can be configured.

Pattern Generator Parameters

Parameters of a pattern are valid for all of its eight channels. This means that all channels are of the same length and all use the same clock.



iC1000/iC2000/iC4000 Pattern Generator Parameters dialog

Name

Defines the name of the current pattern

Frames

Defines number of frames used in the pattern. Frames multiplied by pattern clock define the period of the pattern. The number of frames is limited to 512.

Pattern clock

Defines the clock rate by, which the waveform progresses.

Operation mode

Defines whether the pattern is to run continuously or to execute only a single shot on demand. In any case, pattern operation is controlled from the Hardware menu by selecting the 'Run Pattern' command.

When continuous mode is selected, the 'Run Pattern' command will either stop pattern execution (at the last frame), or resume it.

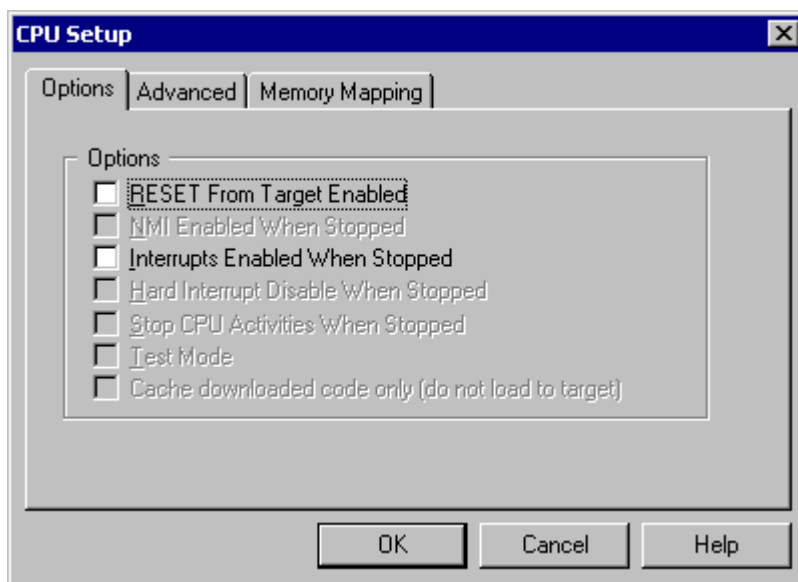
In single shot mode selecting the 'Run Pattern' executes a single pattern shot.

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

3 Setting CPU options

3.1 CPU Options

The CPU Setup, Options page provides some emulation settings, common to most CPU families and all emulation modes. Settings that are not valid for currently selected CPU or emulation mode are disabled. If none of these settings is valid, this page is not shown.



CPU Setup, Options page

RESET from Target Enabled

When checked, the target's RESET line can reset the CPU while the CPU is running.

“Interrupts Enabled When Stopped” checked

When this option is checked, the Interrupt Enable (I (interrupt) on Freescale CPUs) flag is never modified by the emulator. When the user's program is stopped the emulator doesn't influence the state of Interrupt Enable flag. During program stop any interrupts will always be serviced with the exception when BDM, JTAG or SDI is used. When the CPU enters the BDM mode, the CPU itself cannot service interrupts. Thereby they become pending interrupts and are serviced first after the user's program proceeds with execution.

Note: On all 8 bit CPUs the emulator allows interrupt nesting up to 15 levels in depth, representing no limitations in practice. Nesting will occur only if interrupt servicing is interrupted by another interrupt before the servicing is completed. While any nested interrupt is serviced by the CPU, the emulator has no access to the CPU therefore debug windows cannot be refreshed in the meantime.

To allow background interrupt execution on 8 bit CPUs, interrupt routines must meet the following conditions:

- All CPU registers must be preserved,
- Interrupt routines must return with the corresponding return-from-interrupt instruction (RETI, RFI, etc.). Do not assume that your compiler always gets it right. Interrupt routine exiting with jump or call instruction cannot be debugged.
- The return address must not be changed in the interrupt routine.

“Interrupts Enabled When Stopped” unchecked

After the user’s program is stopped (STOP), the emulator remembers the current Interrupt Enable flag status and disables interrupts. When the program is set back to run, the emulator restores the interrupts (Interrupt Enable flag) back and proceeds with program execution (RUN).

There is no problem when the ‘Run’ command is being used, but a problem can occur under certain conditions when a single step command is being used.

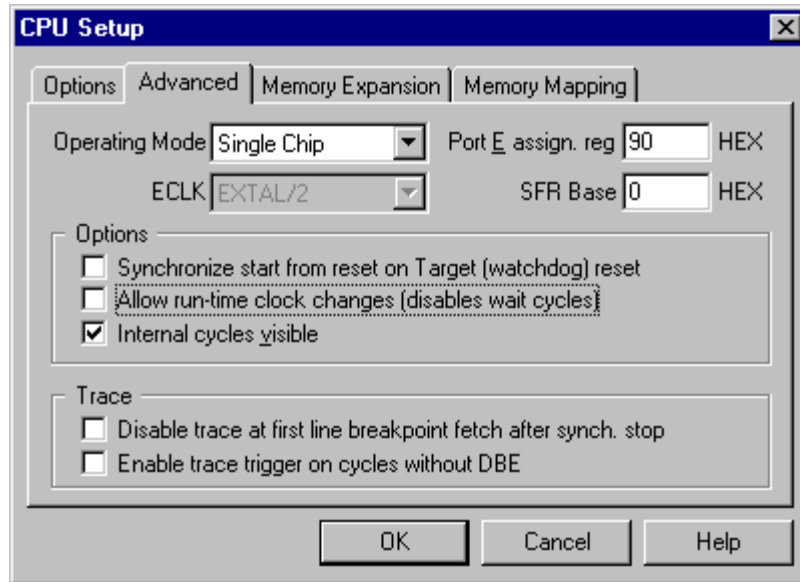
While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of Interrupt Enable flag and handles it correctly.

For example, interrupts are active and the program is stopped. The emulator remembers the Interrupt Enable flag state and disables interrupts. Now the user executes single steps in the disassembly window and, for example, once the SWI instruction (software interrupt) is stepped. At this moment, the CPU pushes the content of the CCR register to the stack, where the Interrupt Enable flag is stored and jumps to the address where the interrupt vector points to. Before the user’s program was stopped (from running), the interrupts were active (Interrupt Enable flag) and after the program was stopped, they were disabled (Interrupt Enable flag) by the emulator. Therefore an incorrect Interrupt Enable flag value (CCR) is now pushed to the stack. Since the emulator can detect such an instruction it modifies the stack with the proper Interrupt Enable value. If this would not be done, the program execution would be changed after RETI instruction in the software interrupt routine is executed. Interrupts in the user’s program would now be disabled and not enabled as before while the program was running.

When using step in the source window the above-mentioned problem becomes relevant and the user should never forget it. The source step is actually executed with RUN command with prior setting of breakpoint on the required source line. If SWI (software interrupt) occurs during one source step the CCR with disabled interrupts will be pushed to the stack and after returning from software interrupt routine (RETI) the same value is popped up from the stack. When the user re-runs his program, interrupts are disabled and not enabled, as before the user’s program was stopped.

During the source step the emulator cannot detect instructions that changes the state of Interrupt Enable flag as it is the case with single step in the disassembly window.

3.2 Advanced In-Circuit Emulation Options



Freescale 68HC12 In-Circuit Emulation Advanced Options

Operating Mode

- Single Chip - use when emulating single chip applications. In this mode ports A, B and E (and K on DG128) are simulated
- Expanded Narrow - use in extended mode applications with 8-bit data bus. In this mode port E is simulated.
- Expanded Wide - use in extended mode applications with 16-bit data bus. In this mode port E is simulated.

Note: the expanded modes are not available when the HC12A4_B POD is used.

Port E Assignment Register (PEAR)

Specify the value that you use in your program. Any configuration of this register in your program is ignored.

Note: Always specify the correct value of this register, even if you do not configure it in your program.

The Emulator requires ECLK, RW and DBE signals for operation and enables them all.

This setting defines the signals that are required for proper target operation.

Note: Reading the PEAR register in the program can return a value different than the one specified here.

When using the HC12A4_B POD in Expanded Narrow mode, the PEAR register must be set to xx10x1xx, when using Expanded Wide mode, the PEAR register must be set to xx1011xx. These signals are necessary for emulation, others can be user defined.

ECLK

When emulating the HC12BD32 CPU the CPU's ECLK can be defined through the DIVBYP pin. The ECLK can be set to EXTAL/2 and EXTAL/4.

SFR Base

The SFR Base setting is required only in case of using memory expansion. Currently PODs HC12DG128 and HC12A4_B provide memory expansion. The user should specify the starting memory address of the RAM area where user's SFRs are located.

Note: If the memory expansion is not being used and if using the ActivePOD, the Emulator autodetects SFR relocation.

Allow run-time clock changes

This setting sets the number of wait cycles to 0.

Check this option when using the PLL in your project. The reason is due to a failure in the CPU design. It is advised to use the latest available mask of the MCU. The problem is that when you switch on the PLL, the MCU clock is changed and thus the BDM communication can fall out. BDM communication must be synchronized with the MCU's clock all the time.

Internal cycles visible

If this option is disabled, R/W signal becomes inactive to the target application during accesses to the internal CPU memory (EEPROM, internal RAM).

Additionally, the timing of the R/W signal is changed – delayed approx. 25 ns after ECLK.

Visibility of internal cycles can be disabled only on the 68HC12DG128 POD when EEPROM and internal RAM are not relocated from their default reset locations.

Synchronize Start from Reset On

The target reset signal resets the CPU immediately. However, when the target reset becomes inactive, the CPU reset is overdue for few hundred milliseconds by the emulator. If external watchdog is active, the CPU restart must be synchronized with the external watchdog, therefore "Reset from target enabled" option in the Advanced dialog must be checked. The watchdog timer event allows reset synchronization on the rising edge of external watchdog (target) reset. Note that the external watchdog must be a periodic signal (while forcing the CPU to a reset state). After the CPU starts, the external watchdog must be refreshed by the application, which ensures the target reset line not to be active.

Trace

The '**Disable Trace at first line breakpoint fetch after synchronous stop**' option will cause the trace to stop recording on the first line breakpoint fetch after a stop request is received on synchronization lines.

The '**Enable trace trigger on cycles without DBE**' will, when checked, trigger on bus cycles even when there is no DBE present. This is the case with internal memory accesses.

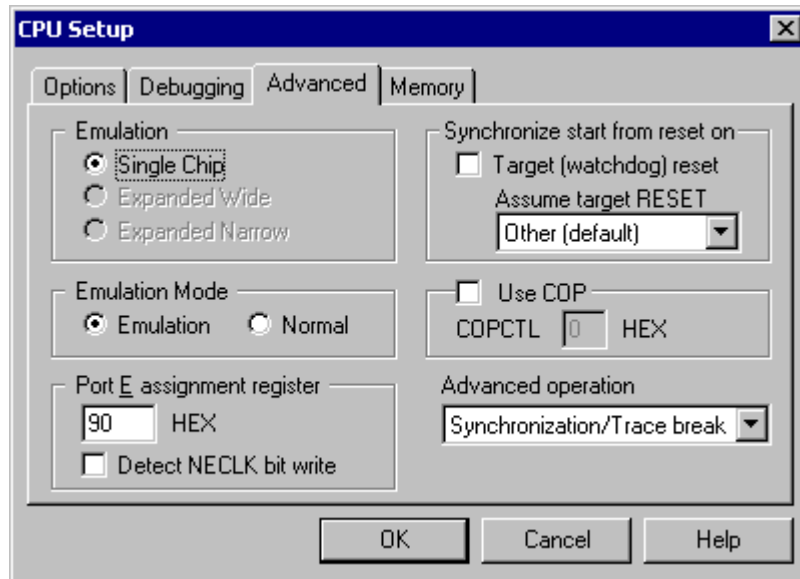
Note: cycles without DBE are very common in regular CPU operation (typically after a write operation). To preserve power the CPU does not put addresses on the multiplexed address/data bus. The result of this behavior is a 'data to address bus propagation'. The Emulator will do its best to filter such cycles and, if possible, not trigger on such propagations. In certain (rare) situations however, the propagated data can still be interpreted as address and cause incorrect triggering.

There are limitations when the MCU accesses internal resources (internal RAM). If the MCU works in 8-bit mode no internal RD and WR cycles are visible. In this case you cannot set any trigger/qualifier on internal accesses.

If the MCU works in 16-bit mode all internal RD and WR cycles are visible but the data bus is not active during internal RD cycle. You can set:

- trigger/qualifier on write cycle as any combination of address, data and control bus or
- trigger/qualifier on read cycle as combination of address and control bus.

3.3 Advanced Active Emulation Options



Freescale 68HC12 Active Emulation Advanced Options

Port E Assignment Register (PEAR)

Specify the value that you use in your program. Any configuration of this register in your program is ignored.

Note: Always specify the correct value of this register, even if you do not configure it in your program, in this case specify the default value.

The Emulator requires ECLK, RW and DBE signals for operation and enables them all.

This setting defines the signals that are required for proper target operation.

Note: Reading the PEAR register in the program can return a value different than the one specified here.

When using ActivePODs, only single-chip emulation is possible and therefore only the functions that can be set on original the CPU in this mode are available.

Supported functions:

For 68HC921DT128: ECLK output on PE4 and CAL (Calibration output) on PE7.

Detect NECLK Bit Write

NECLK bit selects port PE4 operation, which can operate either as IO port or ECLK output. When the CPU is running in Emulation expanded wide mode and the 'Detect NECLK Bit Write' option is checked, the emulator detects writes to the PEAR register and runtime selects PE4 operation. Note that this option cannot be used when Normal expanded wide mode is used for emulation.

Emulation Mode

The emulation mode used can be specified. Two emulation modes are available and both have a limitation:

Emulation Expanded Wide Mode

Because of a CPU flaw, in this mode the write to the IRQE bit in the INTCR register is impossible.

Normal expanded Wide mode

The PEAR register can be written any time and thus writable by the user's application. The application misbehaves when ECLK is turned off by the application since the emulator requires ECLK for its operation. The ECLK signal that goes from the emulator to the target is rebuilt on the POD and set according to the settings in the 'CPU Setup/Advanced' dialog (PEAR value). The POD emulates single chip mode (to the target) according to the CPU specification despite that the CPU on the POD operates in the expanded mode. When emulation mode is used, the register is write once register and first write is executed by the emulator, so the application cannot modify it later any more. The user needs to enter the PEAR value (that his application uses) in the 'CPU Setup' dialog. It is recommended to use the Emulation expanded wide mode always, except when the IRQE bit in the INTCR register is used.

Synchronize Start from Reset On

The target reset signal resets the CPU immediately. However, when the target reset becomes inactive, the CPU reset is overdue for few hundred milliseconds by the emulator. If external watchdog is active, the CPU restart must be synchronized with the external watchdog, therefore "Reset from target enabled" option in the Advanced dialog must be checked. The watchdog timer event allows reset synchronization on the rising edge of external watchdog (target) reset. Note that the external watchdog must be a periodic signal (while forcing the CPU to a reset state). After the CPU starts, the external watchdog must be refreshed by the application, which ensures the target reset line not to be active.

Use COP

The CPU has an internal watchdog, that must be refreshed periodically, or the CPU resets. The COP can be disabled in STOP (while the CPU is stopped by BDM), which is necessary for debugging. Since the register, which turns off COP is a write-once register, the whole register must be written.

Using COP with HC12 PODs

Use COP is a global option with, which the COP usage is selected. If this option is selected, the option to insert RTICTL is available. If COP is enabled, the software enables COP with a write to the COPCTL register (address 0x0016) and enters the value written in the RTICTL option to the RTICTL field. It makes sure automatically that RSBCK (the bit that disables COP when the user's program stopped respectively when the CPU is in the BDM mode) is always active.

Target (Watchdog) Reset and Assume Target Reset

If COP is enabled, the 'Target RESET Enabled' function must be enabled. At COP reset the CPU forces a reset that the emulator only recognizes as target reset.

The HC12 CPUs have three sorts of resets with each having its own individual vector. These are:

- External reset - power on. Vector 0xFFFFE-0xFFFF
- Clock monitor fail reset. Vector 0xFFFFC-0xFFFFD
- COP failure reset. Vector 0xFFFFA-0xFFFFB

In most cases all vectors point to the same location, but this does not have to be so.

To test the situation when the program starts after reset with another vector, the software has three 'Assume Target RESET' options:

- Other (default) (Vector 0xFFFFE-0xFFFF)
- Clock monitor fail (Vector 0xFFFFC-0xFFFFD)
- COP failure (Vector 0xFFFFA-0xFFFFB)

Depending on the selected option, the CPU starts at every target reset on the location, to, which the appropriate vector points.

At every emulator reset (through the software) the CPU always starts on the location, to, which the 0xFFFFE-0xFFFF vector is pointing.

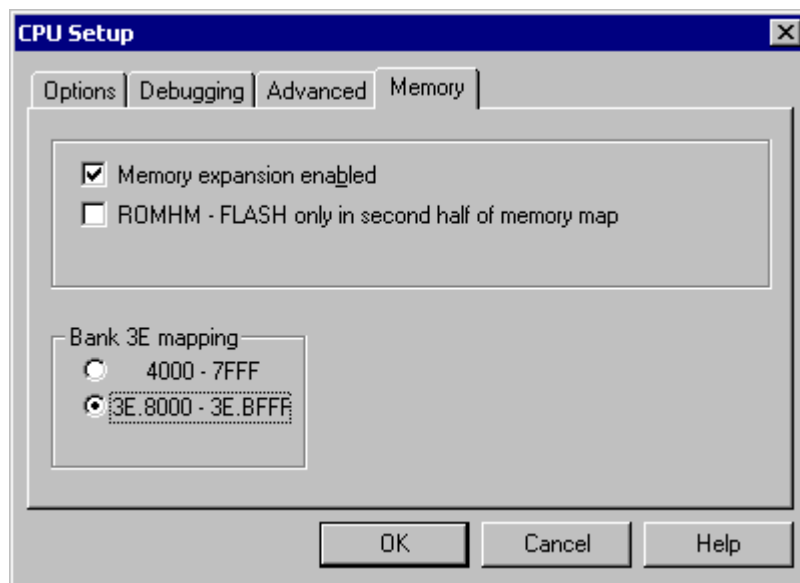
Advanced operation

If software breakpoints are used, the program memory can be used in two ways:

- **Task debugging.** This operation mode enables task breakpoints. These are breakpoints, available only in a certain task, which can be set in the 'Debug/Operating System' dialog.
- **Synchronization/Trace break.** Normally, there is a delay when the stop command is used. This mode minimizes the time needed to stop the CPU and is used for quickly stopping the CPU.

3.4 Memory Expansion

This page configures the mode of the on-chip memory expansion unit. It is shown only for CPUs with integrated memory expansion unit.



Memory Expansion

Memory Expansion Enabled

If Memory Expansion is being used, this option must be checked. According to this selection, the mapping on the POD is automatically configured.

Bank 3E and Bank 6 mapping

At the 68HC912DT128 CPU, a bank 6 (0x68000-0x6BFFF) is a mirror picture of logic area 0x4000-0x7FFF. This area can be used as bank 6 or as logical 0x4000, depending on how the project is linked; the debug information depends on linking. The trace does not know automatically how to transform the adequate address: into logical 0x4000 or into bank 0x68000. With this option proper source level displaying in the trace window is enabled.

ROMHM

This option sets the ROMHM bit in the MISC register. With this bit turned on, access to the Flash Memory in the area of 0x4000-0x7FFF is turned off.

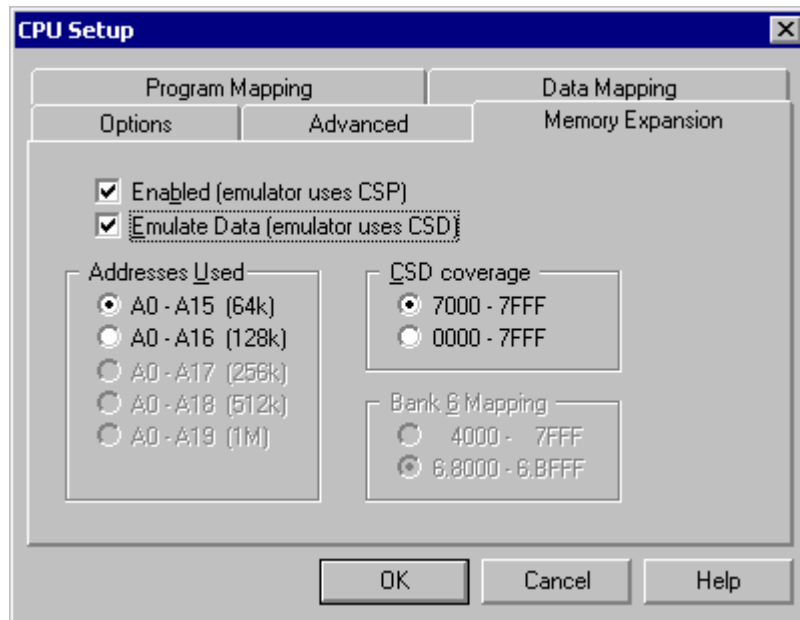
0 - The 16K byte of fixed FLASH EEPROM in location 0x4000-0x7FFF can be accessed

1 - disables direct access to 16K byte FLASH EEPROM from 0x4000-0x7fff in the memory map. The physical location of this 16K byte FLASH can still be accessed through the Program Page window.

The option must be checked, if direct access to FLASH 0x4000-0x7FFF is disabled. The ROMHM must have the same value as the user uses in his application.

3.4.1 Memory Expansion (68HC12A4)

This page configures the mode of the on-chip memory expansion unit. It is shown only for CPUs with integrated memory expansion unit.



68HC12 A4 CPU Memory Expansion Options

Note: The CSD unlike CSP0 and CSP1 signals is not enabled after CPU reset. If Data memory is used, the target program must enable the CSD signal (same as when running without Emulator).

Enabled (Emulator uses CSP)

Check if your application is using memory expansion features.

Emulate Data (Emulator uses CSD)

Check if in a banked application you wish to emulate data memory. If you choose to do so, half of the available emulation memory will be reserved for data memory (CSD addressed), and the other half for program memory (CSP0 addresses).

If you have a working target system, you can choose not to emulate data (it will be used from the target system) and thus gain more memory for program banks.

If the Data area is not emulated, the in-circuit Emulator cannot see the memory accesses. In this case the in-circuit Emulator cannot stop at the breakpoint if the breakpoint is set in the data area. Also the trigger/qualifier on the data area does not work. Therefore the 'Emulate Data' checkbox must be checked.

Note: if Data is not emulated, the Emulator cannot set memory access breakpoints, trace trigger or qualifier conditions on data area.

Addresses Used

This option specifies the top most address used by the application.

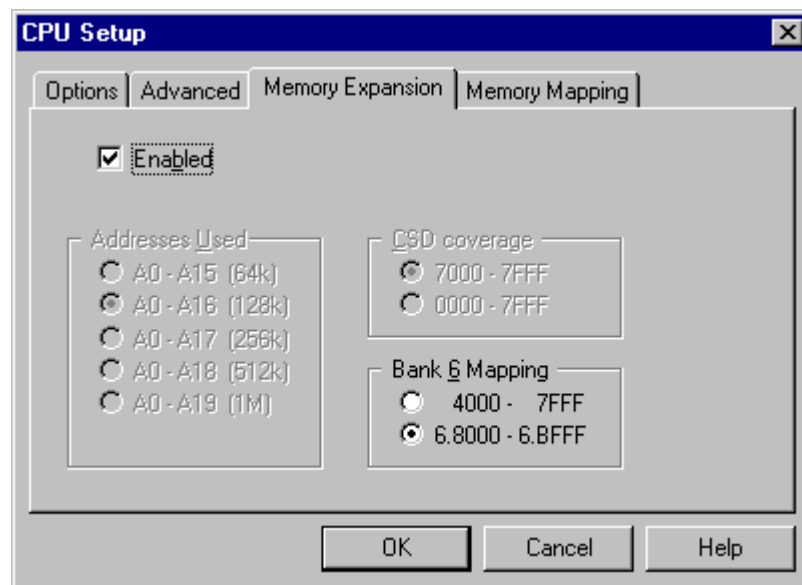
Note: This setting must match the value written (in the target program) to MXAR register.

CSD Coverage

The MCU can generate CSD when it addresses logical address 0x7000-0x7FFF (default) or 0x0000-0x7FFF. The user can configure CSD active area in the dedicated register and respectively he has to check the chosen area. Different linear address is generated from the same logical address if CSD is active in the 0x7000-0x7FFF area or in the 0x0000-0x7FFF area. Thus this option was implemented for proper displaying of address bus in the trace window.

3.4.2 Memory Expansion (68HC12DG128)

This page configures the mode of the on-chip memory expansion unit. It is shown only for CPUs with integrated memory expansion unit.



68HC12 DG128 CPU Family Memory Expansion Options

Enabled

Check if your application is using memory expansion features.

Note: The Hardware breakpoint dialog is not available at the 68HC12 DG128 if memory expansion is used. The reason for this is because the data area is not covered by the emulation memory.

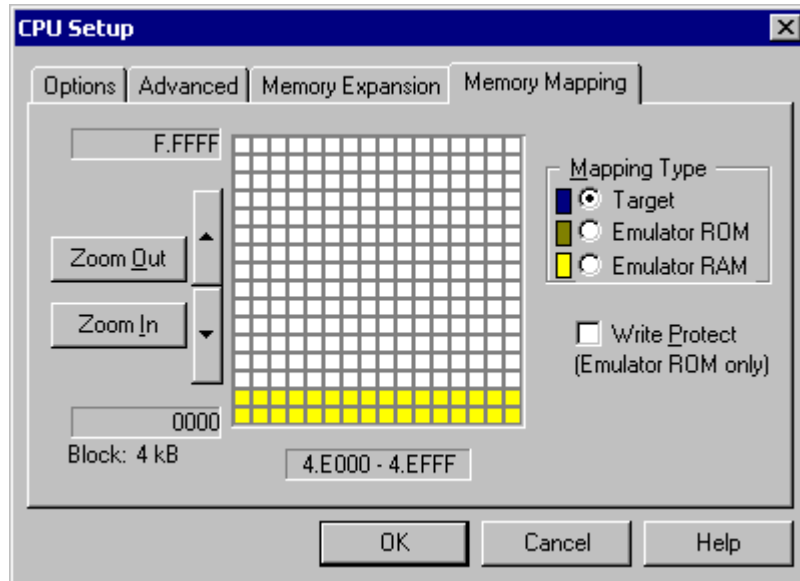
Bank 6 mapping

At the 68HC12 DG128 CPU, a bank 6 (0x68000-0x6BFFF) is a mirror picture of logic area 0x4000-0x7FFF. This area can be used as bank 6 or as logical 0x4000, depending on how the project is linked; the debug information depends on linking. The trace does not know automatically how to transform the adequate address: into logical 0x4000 or into bank 0x68000. With this option proper source level displaying in the trace window is enabled.

Note: This option is available for 68HC12 DG128 CPU only.

3.5 Memory Mapping

The mapping page displays currently configured memory mapping. Memory mapping on the HC12 is configured from the memory device's aspect i.e. a linear addressing configuration scheme is used for banked applications too.



CPU Setup dialog, Mapping page

Gray blocks in mapping configuration area indicate memory ranges that are either outside the CPU's range (bank systems) or aren't covered by emulation memory.

Colored blocks define current mapping of the covered area:

- dark blue for target
- brown for Emulator ROM - CPU can read from it but not write to it.
- yellow for Emulator RAM - read and write access
- cyan for blocks with mixed mapping - use zoom to view where exactly such blocks map.

To change the mapping type of a block, select desired Mapping Type and click on the block that you wish to map to the select type.

Note: Clicking on a block with mixed mapping, clears all underlying mapping configuration and sets mapping for the entire block to selected mapping.

To configure and view mapping at higher resolution:

- click the 'Zoom In' button
- position the mouse cursor over the block that you wish to zoom in; the mouse cursor will change to indicate zoom mode.
- click on the block.

You can configure mapping options with 4k resolution on iC181 and 2 bytes resolution on iC1000, iC2000 and iC4000, while the ActivePOD does not provide memory mapping since this is a single-chip CPU. The mapping configuration area always shows a grid of 256 blocks. In the bottom left corner the current block size is displayed and current ranges are visible to the left of the mapping configuration area. You can zoom in and out and scroll the current range to reach the desired address and resolution.

In general you should configure your mapping as follows:

- where read only devices containing target program are located, set mapping to Emulator ROM. This allows you to download the program quickly without programming EPROMs, while preventing the program from overwriting itself.
- areas occupied by on-chip or off-chip, memory addressable peripherals must always be mapped to target. Otherwise the CPU will not be able to write to them.
- areas occupied by RAM devices can be mapped either to target or to Emulator RAM. You will want to have them mapped to Emulator if the target system is not being used, or when using advanced debugging features like real-time watches. Otherwise map them to target.

Write Protect

Prevents the memory, mapped to the Emulator ROM, from being written to. If this option is checked, a write to the Emulator ROM area results in an error message.

Note: This option is available only for the Emulator ROM type of memory.

Data Mapping

Data Mapping is available only on CPUs that can separate program and data accesses with external chip select signals (like the A4) and these chip-selects are used (banked applications).

On DG128, the configurable mapping applies to 128kB program memory, but the internal data memory is always used on the CPU and cannot be mapped to the Emulator.

Note: On single-chip applications, make sure that the memory where the on-chip special function registers are located, is mapped to target, if memory expansion is not used.

If memory expansion is used, the logic addresses between 0x0000 and 0x3FFF are automatically mapped to target, since this is the SFR and internal RAM area.

4 Debugging Interrupt Routines

An interrupt routine can only be debugged when the interrupt source for this routine has been disabled; otherwise you will keep reentering the routine and thus run out of system stack.

For example, there is an interrupt routine with 10 source lines. Let's say that interrupt routine is called periodically by free running timer is an interrupt source. A breakpoint is set on the first source line in the interrupt routine. Program execution stops at the breakpoint. Now source step is executed. Source step is actually executed using RUN command with prior setting of breakpoint on adequate source line. In this particular case, while source step is executed, the CPU executes the code and before source step finishes, new interrupt call occurs. New values are pushed on to the stack and the CPU stops on breakpoint again. If you repeat source steps in such interrupt routine new values are pushed to the stack and you can easily run out of stack.

An interrupt source can be disabled in two ways:

- Disable the interrupt process in the stopped mode. The stopped mode is entered whenever CPU is stopped, and the emulator remains in stopped mode until the Run command is executed. (During Step, Step over, etc. commands, the stopped mode persists).
- Do not place a breakpoint on any instruction in the interrupt routine where interrupts are not yet disabled.
Also, you must not step over any instruction that re-enables the current interrupt, but run the program before the instruction is executed.

5 Memory Access

HC12 development tools feature standard monitor memory access, which require user program to be stopped and real-time memory access, which allows reading and writing the memory while the application is running.

Real-Time Memory Access

All HC12 development systems feature real-time memory access. The debugger can access CPU memory without disturbing the program being executed. Using hardware commands and on-chip BDM firmware, the debugger can access required resources in real time. In general, on-chip BDM firmware uses CPU dead cycles, when real-time read or write memory access is required. In worst case, some CPU cycles might be stolen.

Note: Real-time access is available only for memory, which can be accessed through single memory read/write access.

Watch window, memory window and SFRs window can be updated in real time.

Monitor Access

When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

Since all accesses are performed using the CPU, all memory available to the CPU can be accessed. The drawback to this method is that memory cannot be accessed while the CPU is running. Stopping the CPU, accessing memory and running the CPU is an option, which, however, affects the real time execution considerably.

The time the CPU is stopped for is relative and cannot be exactly determined. The software has full control over it. It stops the CPU, updates all required windows and sets the CPU back to running. Therefore the time depends on the communication type used, PC's frequency, CPU's clock, number of updated memory locations (memory window, SFR window, watches, variables window), etc.

6 Emulation Notes

6.1 Memory spaces on 68HC12 CPUs

Following memory spaces are available:

Logical

This is a 16 bit (64kB) addressable space as seen from the CPU core before any MMU translation.

Linear

This is a 22 bit (4MB) addressable space as seen from outside of the CPU (like by an EPROM or RAM device). The (logical) address issued by the CPU core can pass through memory expansion scheme where it is translated into a 4MB linear address space. Note that this address is not sufficient to fully describe a location, since additional information can be passed in chip-selects (CSD, CSP, etc.).

In case where a fully descriptive address is required (program download, breakpoints, trace, etc.) program area (CSP) is assumed.

Banked

This is a 24 bit addressing convention with, which a location can be fully described. The upper 8 bits hold the bank number (the value of an appropriate page register) and the lower 16 bits hold the logical (16 bit) address.

The Emulator can identify the matching chip-select and calculate the linear address by knowing the layout of bank areas of the current CPU.

Since this is the only fully descriptive method, this is the preferred notation.

Note: In case no memory expansion is used, all address spaces are identical.

6.2 Hardware breakpoints

There are a few combinations, in, which the Hardware breakpoints dialog is not available, because the data area is not covered by the emulation memory.

The Hardware breakpoint dialog is not available at the 68HC12 DG128 if memory expansion is used.

Also, the Hardware breakpoint dialog is not available at the 68HC12 A4 if data banks are not emulated.

6.3 Clock Setup

Even when an external clock source is used, you must specify its frequency to allow SDI synchronization (used by Emulator) after CPU is released from reset.

6.4 Reserved CPU Resources

No CPU resources are used.

6.5 PLL

The on-chip PLL is supported by the emulator. When using a PowerPOD, the 'Allow Run-Time Clock Changes' must be checked. The ActivePODs support it by default without any additional settings necessary.

Synchronization

RESET	Emulator	Target
Entry (RUN→RESET)	Immediate	Immediate
Exit (RESET→RUN)	150 E cycles	Immediate

Note: If you are using two Emulators with the HC12 running on the same clock, they will start synchronously. Also the RESET output line on the POD is delayed until the CPU starts. You can use this line to synchronize a third CPU.

Run/Stop	Emulator
Entry (RUN→WAITING)	On next source line
Exit (WAITING→RUN)	0-12 μ s

6.6 COP

Using HC12 Power PODs, the internal COP must be disabled while debugging. When using HC12 Active PODs, COP can be used while debugging. COP is supported by ActivePOD.

When experiencing some problems with the emulator when using COP, please check, whether you have 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 resets the CPU when the interrupt occurs.

6.7 STOP Instruction

STOP instruction is completely supported by the emulator. After the STOP instruction is being executed, the CPU is stopped and the debugger displays HALTED status. Note that the debug windows cannot be updated while HALTED status is displayed. When the CPU is awoken either by interrupt or target reset, the emulation/execution proceeds normally.

6.8 Internal CPU FLASH

Note that internal FLASH is disabled during the emulation and cannot be used in any way. It should also not be enabled in the user program in any situation, since this would disable the emulation.

6.9 'Clear on read' register bits

Be careful, when the CPU has special function 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 register or the associated memory location in the memory/watch window during final test. Otherwise, it may happen that 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 a flag in the application. Using the emulator, the application works correctly since the user uses a SFR window, which clears the flag when the window is updated.

6.10 Interrupts Enabled While Stopped

If the 'Interrupts Enabled While Stopped' option is selected in the CPU Setup/Options menu, the interrupts will not be processed in the background when the user program is stopped, since this is not supported by the CPU. If an interrupt occurs when the program is stopped, it becomes pending and will be executed once the program is run again or a single step is performed. If this option is not checked, the interrupts are disabled during stop.

6.11 Internal EEPROM or RAM download

The internal EEPROM or the internal RAM must be downloaded via the Target download.

6.12 Things to remember

- Ports that are simulated (additionally serial 100 Ohm safeguard resistors are used) may have slightly different AC/DC characteristics.
- The reduced drive of the I/O lines register (RDRIV) is ignored, because ports are simulated.
- In single chip mode, the pull-up control register (PUCR) is ignored. All pull-ups are always active.
- The 68HC12A4 CPU has no STOP indication. As a consequence the trace buffer will hold extra samples indicating CPU stop activity when used in continuous mode.
- A, B and E port registers (Port Registers and Port Data Direction Registers) must always be mapped to target when using a PowerPOD and B32, D60 or DG128 CPUs without memory expansion.
- A, B, E and K port registers (Port Registers and Port Data Direction Registers) must always be mapped to target when using DG128 without memory expansion. Beside these registers you can use memory mapping only in flash/code/program area.

- The ROMON bit in the MISC register must not be set to 1. By doing so the internal FLASH would be enabled. Internal FLASH is overlaid with emulator's memory thus internal FLASH must be disabled.

6.12.1 Troubleshooting Execution Breakpoints

When reading memory, on the breakpoint address a breakpoint instruction is inserted. Since the Emulator does not know whether this is an execution address or a data address (for example reading data when calculating checksum at startup), it always inserts a breakpoint instruction, which results in wrong data read.

Workaround:

- When using execution breakpoints on a CPU with no fetch signal, the breakpoint must always be set to the first byte/word of the instruction. It is recommended that it be set in the source.

In a development stage, such safety checks (like checksum calculation) should not be executed or all breakpoints should be deleted before calculating checksum.

Notes:

Notes:

Disclaimer: iSYSTEM assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information herein.

© iSYSTEM. All rights reserved.