Freescale 68HC11 Family In-Circuit Emulation

Contents

1 In-Circuit and Active Emulation 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.

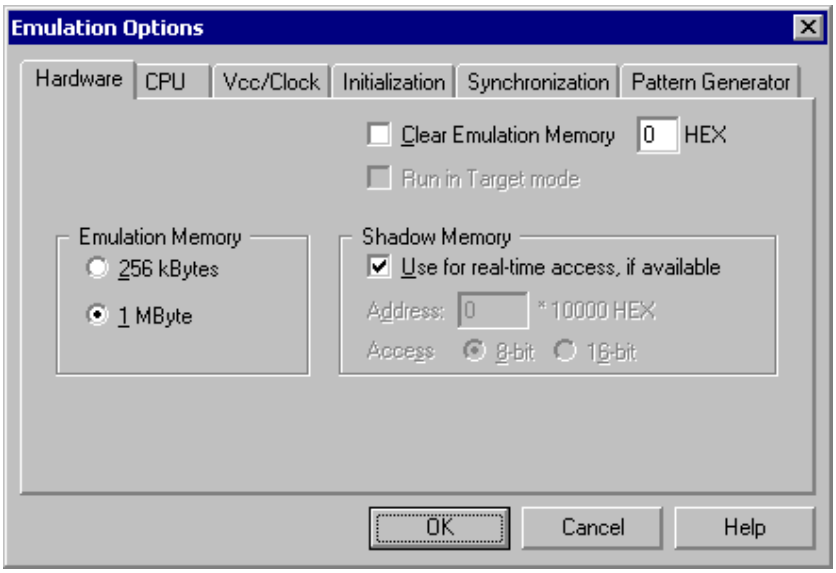
1.3 Port Replacement Information

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 rebuilt "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.

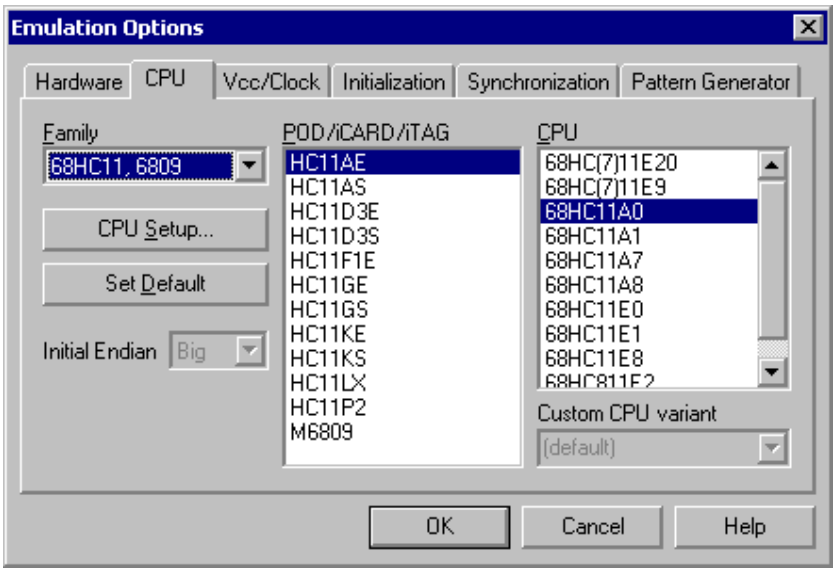
Shadow Memory

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 19) will be used for real-time access.

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

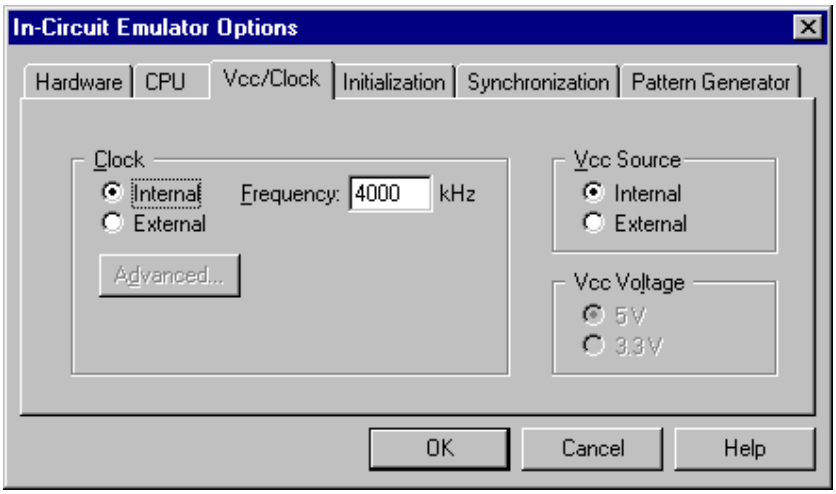
This button will set default options for currently selected CPU. These include:

- 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 1kHz. 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 32.786kHz 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

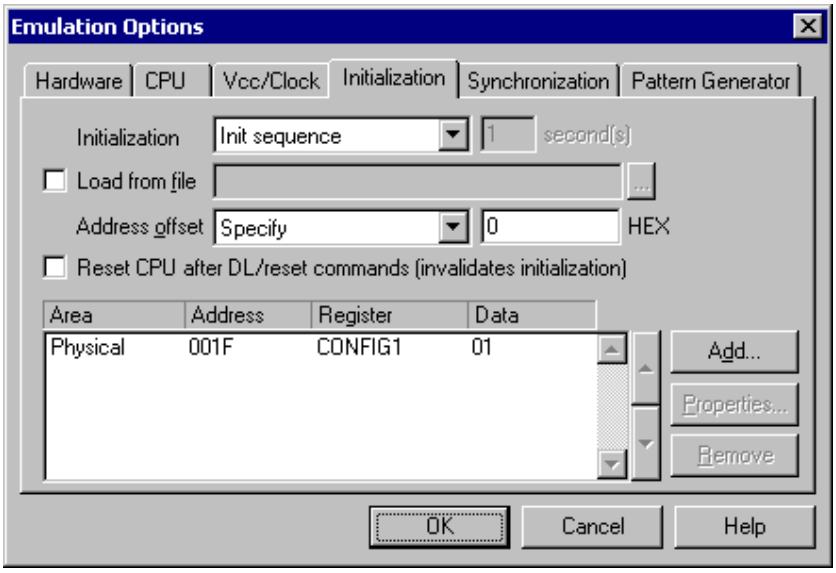
Determines whether Emulator or the target system provides power supply for the CPU.

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:

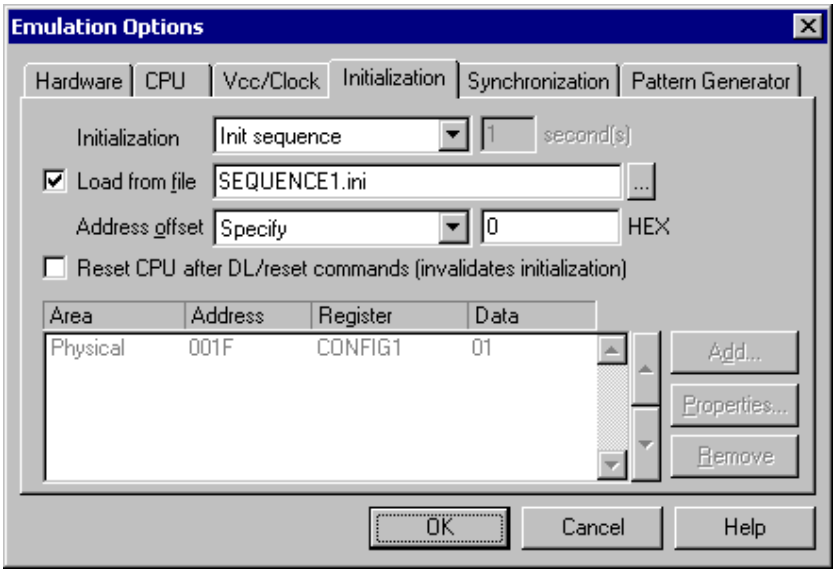
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

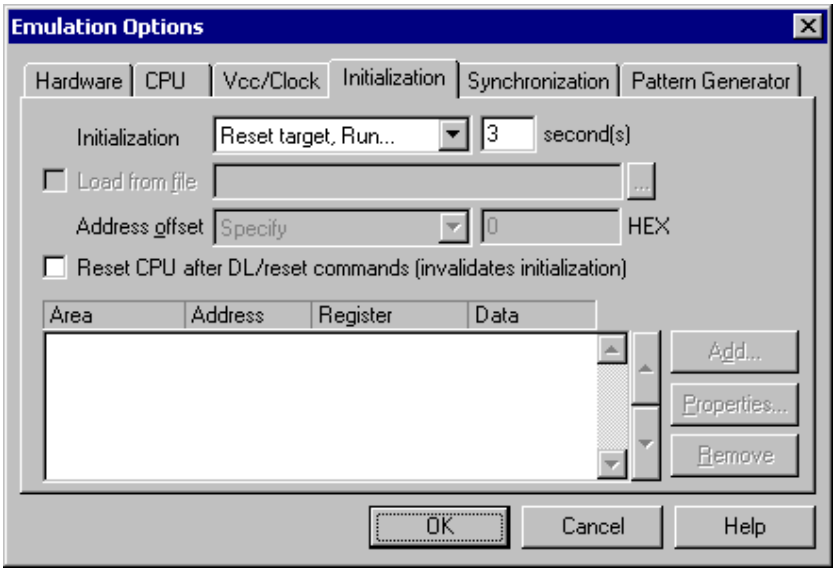
   Excerpt from the sample SEQUENCE1.ini file:

   ```
   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 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.

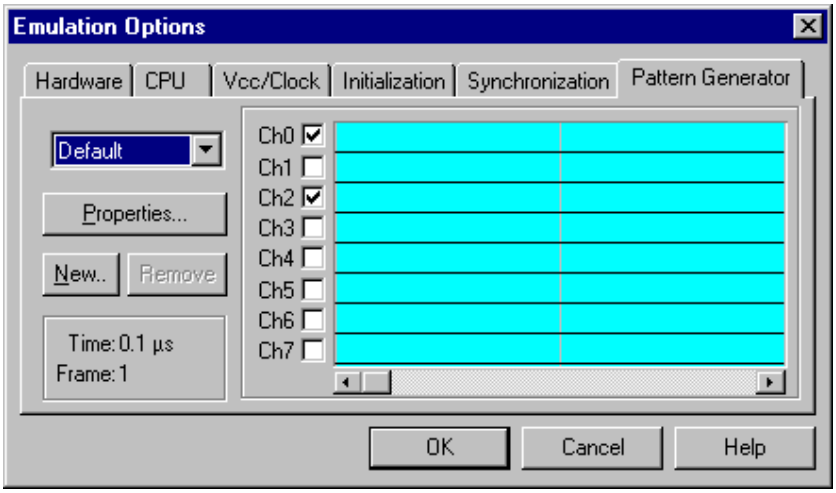
Note: when using the iC4000 system, it has in certain configurations two Pattern Generators: one on the base module and one on the Power Emulator module. The Pattern Generator on the base module is active when the debugging type is set to 'Active Emulation' or 'BDM/JTAG Emulation', the Pattern Generator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



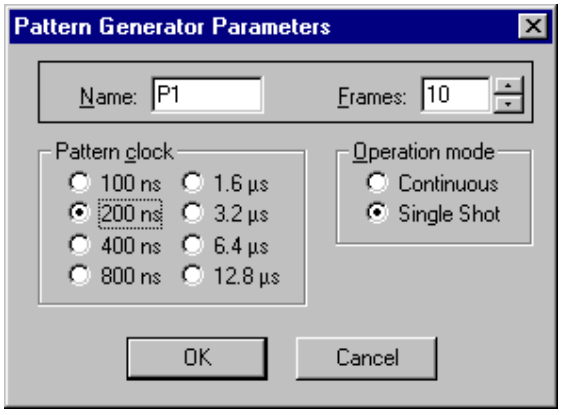
In-Circuit Emulator Options dialog, iC1000/iC2000/iC4000 Pattern Generator page

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.

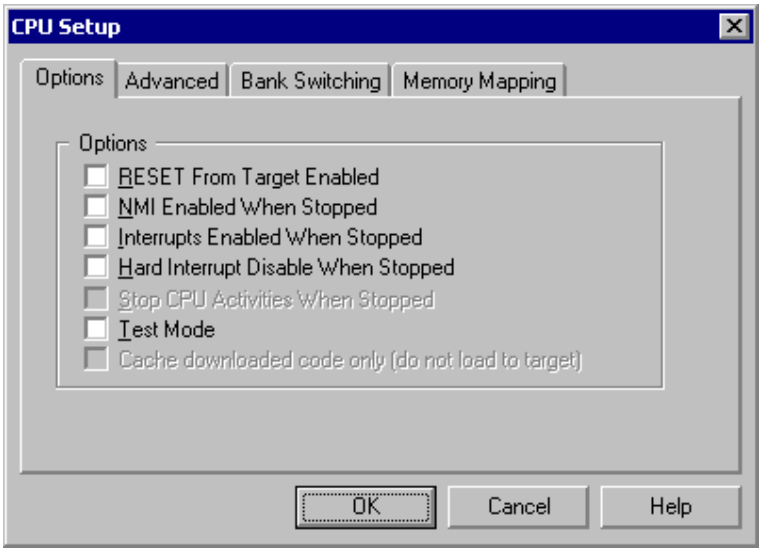
Note: Pattern generator operation can be controlled by an external device through the TRIG/CLKEN pin on the pattern generator connector. Refer to the Hardware User's Manual for more information.

---

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.

NMI Enabled When Stopped

Allows background servicing of NMI interrupts while the main program is stopped.

Otherwise any non-maskable interrupt is disabled using Emulator hardware.

"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.

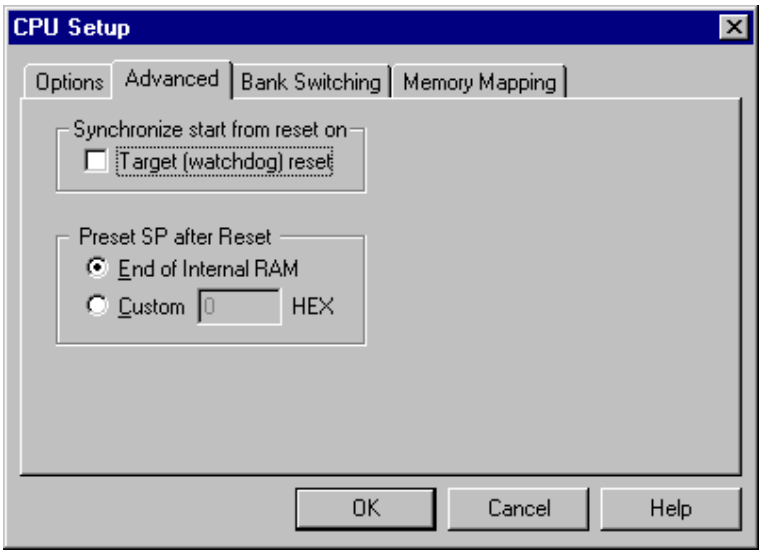
Hard Interrupt Disable When Stopped

When this option is checked interrupts will be enabled immediately after program execution resumes. Otherwise, the CPU must execute a couple of instructions before returning to the program to determine whether interrupts were enabled when the CPU was stopped. These extra instruction executions can prevent task preemption when an interrupt is already pending.

Test Mode

Should be checked when the 68HC11 chip is started in Test Mode.

3.2 Advanced Options



68HC11 CPU Family Advanced Options

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.

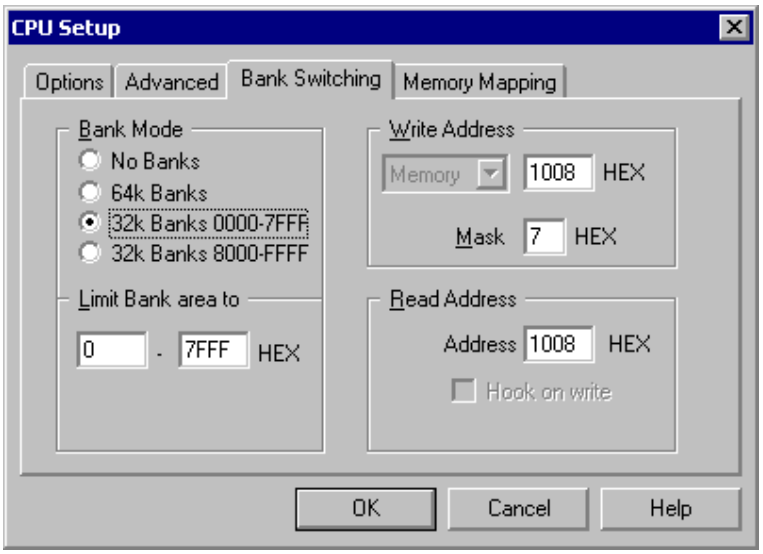
Preset SP after Reset

The emulator presets the reset SP to a value defined in the dialog. It can be modified later by the user's program. By default the SP points to the end of the internal RAM.

A custom value should be entered when the code is downloaded in the area where the reset stack pointer is located. The SP location used in the application should be entered.

3.3 Bank Switching

The Banks pages determine custom bank switching parameters. There will be one Banks page visible for every memory area that can be externally addressed by the CPU.



CPU Setup dialog, Bank Switching page

Bank switching is supported on extended mode PODs.

Note: for more information on bank switching, please refer to chapter 5, Bank Switching Support.

Depending on Emulator RAM capacity a different number of banks will be available. The table below lists number of available banks:

	256k Emulator	1M Emulator
64 KB banks	Root + Bank 1-3	Root + Banks 1-15
32 KB	Root + Banks 1-7	Root + Banks 1-31

3.3.1 Compiler Support

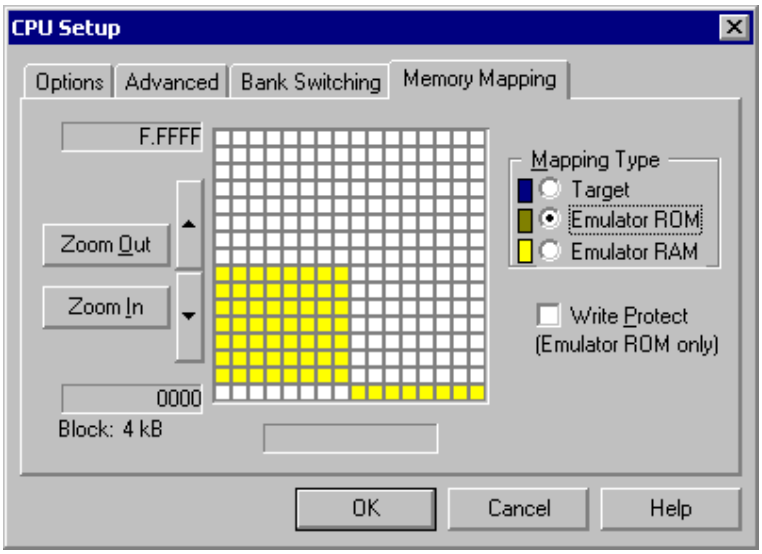
Bank switching is supported, you must however configure startup files according to the banking scheme that you will be using. Refer to your compiler manual for further information.

Since the Emulator does not support Bank 0, you must not put any program modules in Bank 0. To do this, simply omit all references to Bank 0 in the linker command file.

Call stack is not supported when using custom bank switching.

3.4 Memory Mapping

The mapping page displays currently configured memory mapping.



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.

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.

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 gets it right always. Interrupt routine exiting with jump or call instruction cannot be debugged.
- The return address must not be changed in the interrupt routine.

5 Bank Switching Support

Bank switching is an extension of the CPU's addressable memory. It is used mainly on CPUs where programs have grown larger than 64k.

User programs switch banks through an on-chip port or memory mapped latch, which in turn provides chip select or additional address lines to the target system's memory devices.

The CPU still operates with 16-bit addresses although up to 16 banks of 64KB each can be used. This memory is treated as linearly addressable. Using the bank number as the upper 4 bits and the CPU's 16-bit address as the lower 16-bits forms extended 20-bit addresses. Address 2345h in bank 1 is displayed as:

1 2345

Things to remember

- Remember to set the ports that switch banks to output. Otherwise neither Emulator nor standalone operation can access banks.
- Banks will be properly visible after the ports that switch banks are configured as outputs. Before that the Emulator cannot preset the bank.
- Banks cannot be switched manually in the disassembly window. Only addresses within the current bank can be preset.

5.1 Hardware Configuration

In-Circuit emulation PODs that support bank switching, provide input lines to, which you must connect signals that drive the target's chip select logic. These inputs are marked BS0 through BS3. On 8051 family PODs, additional inputs for XDATA bank switching, marked BX0 through BX3, are provided.

These signals are used to allow the Emulator to recognize, which bank is currently active. This way breakpoints can be set across the entire address space covered by banks.

Example:

HC11 CPU's CODE banks are switched through port P1, bits 3 through 5 - yielding 7 banks and common area.

In such case:

- connect port P1 bit 3 line to input BS0
- connect port P1 bit 4 line to input BS1
- connect port P1 bit 5 line to input BS2

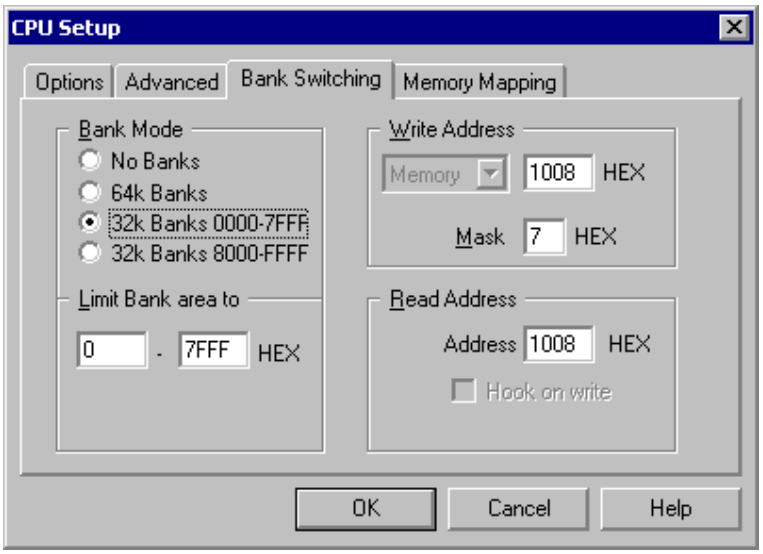
Check whether the necessary signals are already available on the POD.

Note: All bank switch PODs provide default port lines on the POD. On 8051 family, for example, port P1 bits 0 through 3 are located next to BS0 - BS3 lines. If you use these lines in your application, you simply bridge them with jumpers.

If you are using 64k bank switching, you must bridge PODs BSC pin to GND (use a jumper).

5.2 Software Configuration

To allow Emulator access to banked systems (memory access and breakpoints in full address space not just in the current bank), it must be made aware of the type of the bank switching system, the addresses of the ports that drive it, etc.



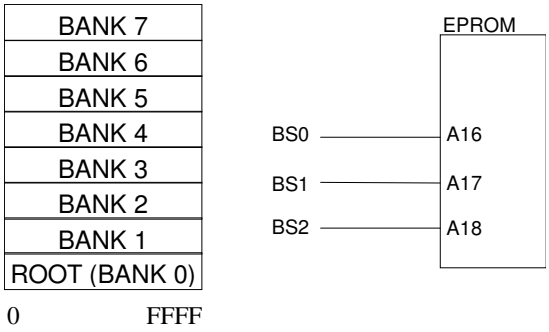
CPU Setup dialog, Bank Switching page

When the selected POD supports bank switching, a bank configuration page in the CPU setup dialog will be available for every CPU memory area where banks are supported.

Bank Mode

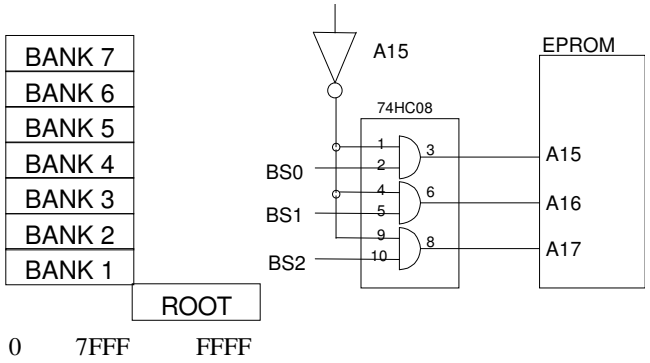
The bank mode determines bank configuration in the memory area. This can be:

- No Banks - no bank switching is used
- 64k Banks - 64k bank switching is used



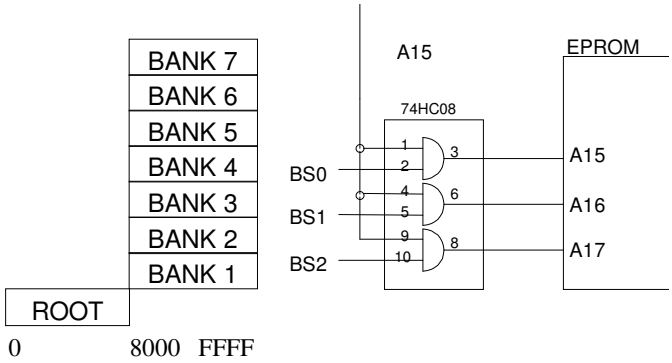
Target chip select logic for ROOT+7 64k banks configuration

- 32k Banks 0000-7FFF - lower 32k of CPU's address space is used for banks



Target chip select logic for ROOT+7 32k banks 0000-7FFF configuration

- 32k Banks 8000-FFFF - upper 32k of CPU's address space is used for banks



Target chip select logic for ROOT+7 32k banks 8000-FFFF configuration

In above figures signals BS0, BS1 and BS2 are outputs from the port or latch where the CPU writes to switch banks.

Note: Bank 0 is not supported on 32k bank switch systems.

The 'Limit Bank Area To' setting is used to additionally limit the address range where bank switching is used. This setting should be used when the bank area is smaller than implied by bank mode.

Example:

ROOT is located on addresses 0-3FFFh, banks are switched from 4000h-FFFF, target chip select logic is designed for 64k bank switching.

In such case:

- select 64k banks mode
- limit bank area to 4000h - FFFFh

Note: If you do not wish to limit the bank area, set the limit values to 0 and FFFF respectively.

Write and Read Address

To allow the Emulator to access the entire banked address space of the target, you must specify the address or register, which is used by the program to switch banks. This is the address of a memory-mapped latch or an on-chip port that drives the chip select unit.

In the field preceding the write address, specify the memory area where this port is mapped (depending on CPU family, this will usually be DATA, XDATA, I/O or MEMORY, in case of the Z80 family, the read address can be either MEMORY or I/O area).

The mask field defines the number of bits and their offset within the byte that is written to the write address.

Example:

HC11 CPU banks are switched through port PD, bits 0 through 2 - yielding 7 banks.

In such case:

- set write address to 1008 (address of port PD)
- set mask to 7 (bits 0,1,2 set to one, others to zero)

Note: Bits used to switch banks must be consecutive. You cannot use bits 3,4 and 6.

The Read address is the location where the last value written to the write address is cached. This will be the same as the write address, unless the value cannot be read from the write address (memory mapped latch). In such case the compiler must be configured (it usually is automatically) to store the value written to the write address to a location where it can be read as well. For the above example:

- set the read address to 1008

6 Memory Access

HC11 development tools feature standard monitor memory access, which require user program to be stopped and real-time memory access based on shadow memory, which allows reading the memory while the application is running.

Real-Time Memory Access

Real-time memory access is available on PowerEmulator unit with shadow memory. Data area can only be read in real-time.

Real-time write memory access is not possible due to shadow memory use. Monitor access must be used to write to the memory.

There is an alternative solution to the shadow memory. The debugger can access debug memory almost in real time using debug monitor. Stop takes 1 instruction for 1 byte variable or 2 byte variable aligned on even address and n*20us for n byte variables. Note that CPU internal memory cannot be accessed using this method since it's limited to debug (ICE overlay) memory.

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.

7 Emulation Notes

7.1 Reserved CPU Resources

The following CPU resources are used:

- 2 bytes of stack are used when the CPU is stopped

7.2 Requirements

The signal MODA/LIR must be connected to the target via a pull-up or pull-down resistor. MODA/LIR is an input for the MCU before the reset is released and specifies MCUs operating mode. After reset this is an output from the MCU. Thus the signal must not be connected to the Vcc or GND directly.

7.3 Memory Mapping

On single-chip PODs, make sure that the memory where the on-chip special function registers are located, is mapped to target.

7.4 Reconfiguring On-Chip Peripherals

Some on-chip peripherals must be configured within the first 64 E cycles after CPU reset. To allow this, don't use the 'Reset' command, but the 'Reset and Run' command.
The 'Reset and Run' command resets the CPU and then lets it run, thus keeping the first 64 E cycles entirely to the user program.

To debug your program, you should place any breakpoints after the initialization code (usually on the entry to function 'main').

7.5 Watchdog timer

The internal watchdog must always be disabled when using a POD. To disable the watchdog timer (refer to the CPU datasheet - set NOCOP bit in the CONFIG register – see "Configuring the CONFIG Register" on page 21).

7.6 Internal ROM

Have in mind when exchanging CPU in the POD that internal ROM must be disabled at any inserted CPU in the POD by programming the ROMON bit in the CONFIG register - see "Configuring the CONFIG Register" on page 21).

7.7 Programming the on-chip EEPROM

According to the selected CPU, the Emulator will correctly handle its accesses (when writing to and refreshing memory windows, etc.) to the on-chip EEPROM (if present).

You must however pay attention to the configuration of the BPROT register (if available on the CPU used), which can prevent write access to the EEPROM. If you want to modify the contents of the EEPROM within your debugging session, you should disable the BPROT protection. Refer to the CPU manual for more information.

Note: You will have to do it within the first 64 E cycles.

7.8 Configuring the CONFIG Register

CONFIG Register in EEPROM

On such CPUs the CONFIG register can only be programmed in the TEST mode. Since the Emulator won't recognize the CONFIG register as an EEPROM register, any write to it must be implemented in the user program running in TEST mode.

Note: the register is in RAM area only on HC11D and HC11G derivates.

Note that the new value written to the CONFIG register will take effect after the next CPU reset.

Note: If you exchange the CPU on the POD and experience problems, make sure that the new CPU has the CONFIG register programmed as required (watchdog timer and PROM disabled).

You will find an example of how to program the CONFIG register on the distribution disk.

CONFIG Register in RAM

On such CPUs the CONFIG register must be written by the user program within first 64 E cycles.

You must use the 'Reset and Run' command to allow CPU configuration within first 64 E cycles after reset

7.9 Running in TEST Mode

To run the CPU in the TEST mode, you must enable the 'Test mode' option in the 'CPU Options' page.

Note: the TEST mode has RESET and interrupt vectors in different locations than 'extended' and 'single chip' modes. Therefore you must adjust the linking of the program vector table

Also note that the 64 E cycle limitation does not apply in TEST mode

8 Other Things to Pay Attention To

Don't tie MODA/LIR pin directly to GND or Vcc. Since it acts as input only during reset, and then turns to output, you will cause a short by tying it directly to Vcc. Always use a pull-up or a pull-down resistor.

LIR signal is required for proper emulation.

When using single chip PODs the B and C port registers (Port Registers and Port Data Direction Registers) must always be mapped to target, with POD HC11K4S also the ports F, G and H.

9 Single chip emulation

HC11exaf and HC11axaf adapters are used to emulate single chip mode. In this mode port B and port C are rebuilt on the adapter by a so-called Port Replacement Unit (PRU) designed by Freescale. Both adapters have some limitations, which originate from Freescale's PRU device.

When using port B and POD with HC11exaf adapter the user should write appropriate value into the port B data direction (DDRB) or to the adequate address (0x1006) when there is no DDRB register. Some MCUs supported by HC11exaf adapter have DDRB and some don't. For proper PRU operation the user must always write to DDRB (0x1006) prior to use port B as output.

Tip: To define port B as output you can add a single write of value 0xFF to address 0x1006 in the 'Initialization' tab in the 'Hardware/In-circuit emulation' dialog. This sequence will be executed always when the system will be reset for the first time or when the download will be executed. Of course the user can add this sequence in his user program instead to the 'Initialization' dialog.

Notes:

Notes: