Freescale 68HC05 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.

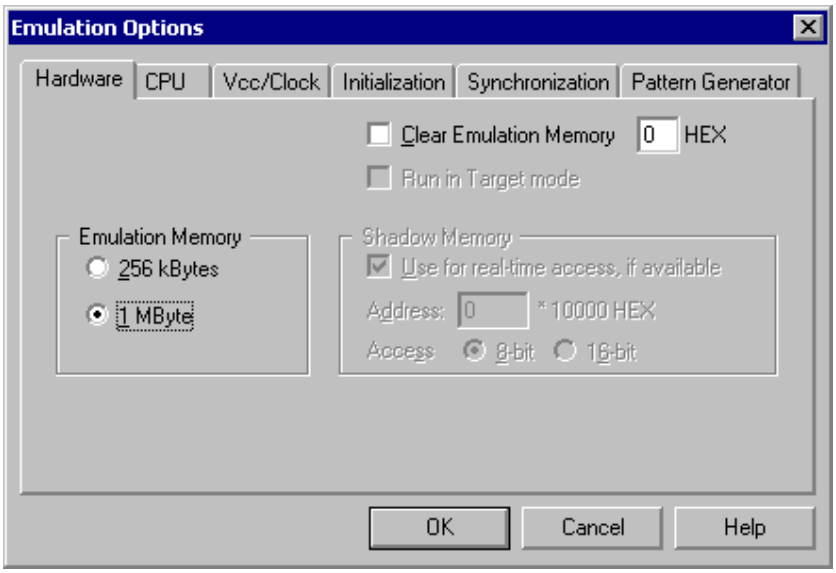
1.3 Port Replacement Information

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.

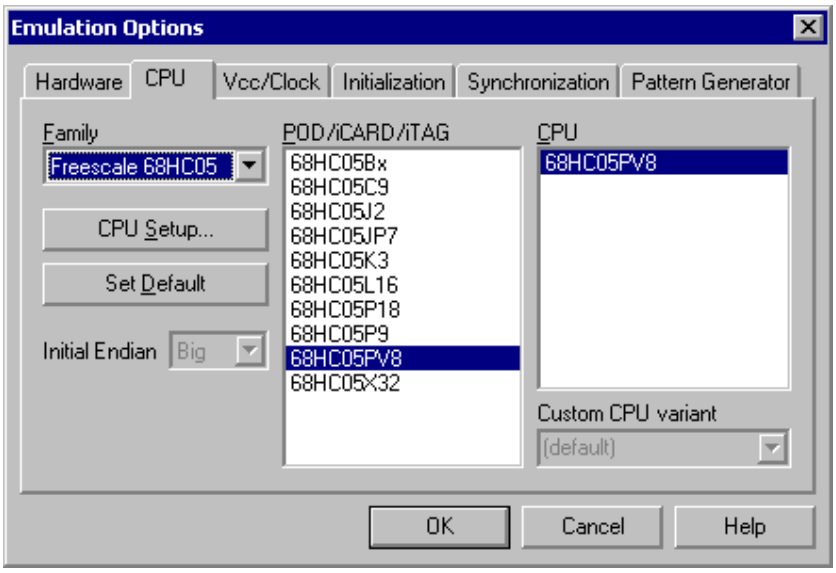
Clear Emulation Memory

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.

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

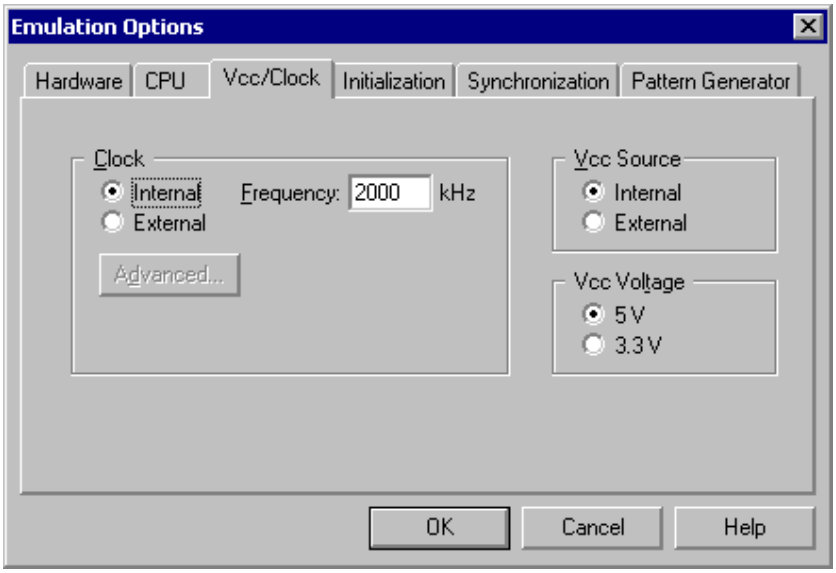
This button will set default options for currently selected CPU. These include:

- 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.

Vcc Voltage

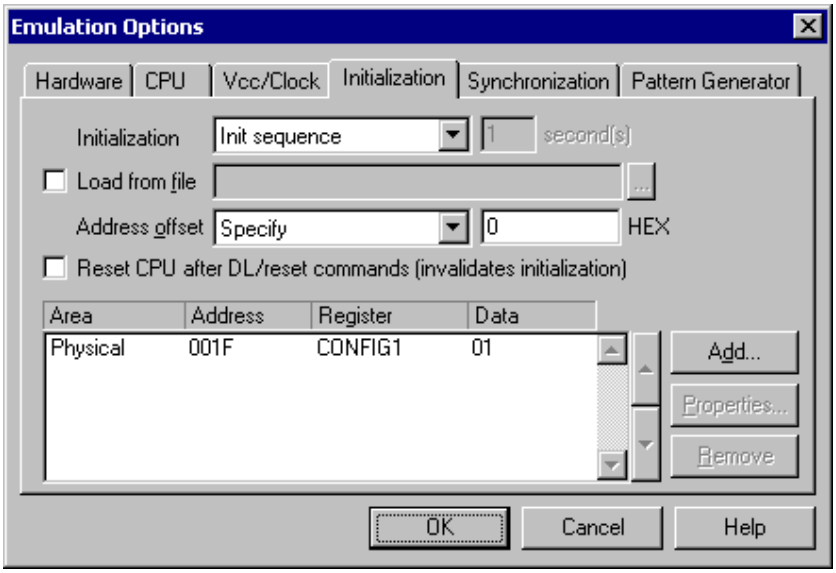
Determines (when applicable) the voltage used. Note that this setting is usually set by a jumper on the POD.

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:

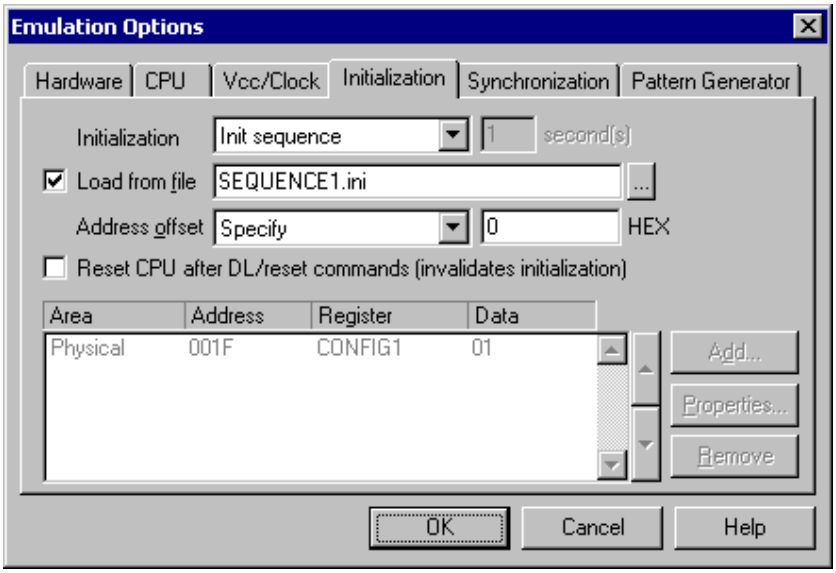
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

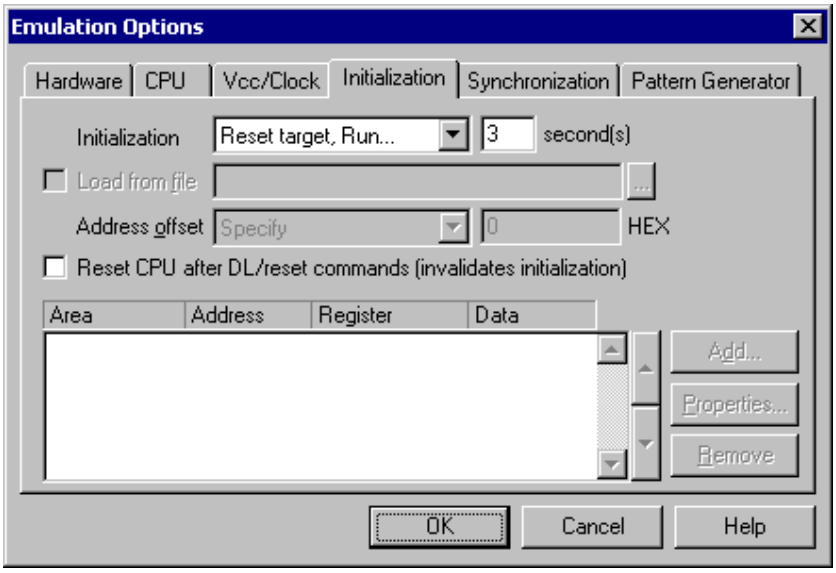
   Excerpt from the sample SEQUENCE1.ini file:

   ```
   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.

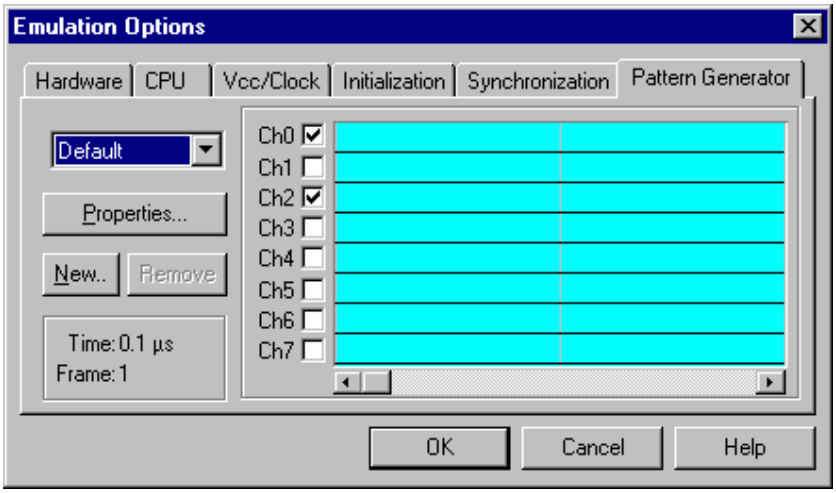
Note: when using the iC4000 system, it has in certain configurations two Pattern Generators: one on the base module and one on the Power Emulator module. The Pattern Generator on the base module is active when the debugging type is set to 'Active Emulation' or 'BDM/JTAG Emulation', the Pattern Generator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



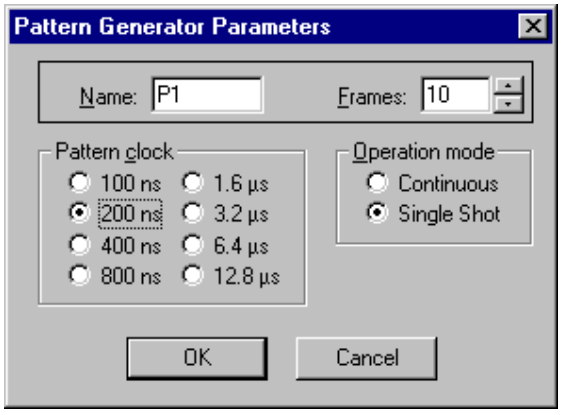
In-Circuit Emulator Options dialog, iC1000/iC2000/iC4000 Pattern Generator page

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.

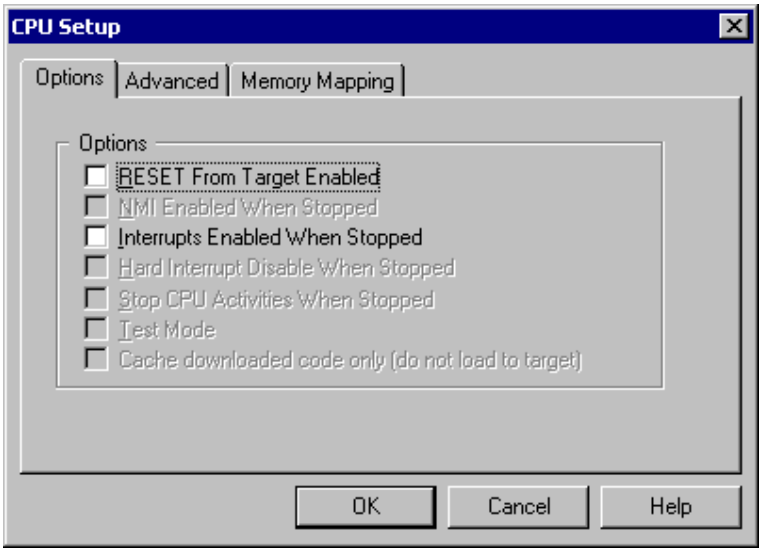
Note: Pattern generator operation can be controlled by an external device through the TRIG/CLKEN pin on the pattern generator connector. Refer to the Hardware User's Manual for more information.

---

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.

"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.

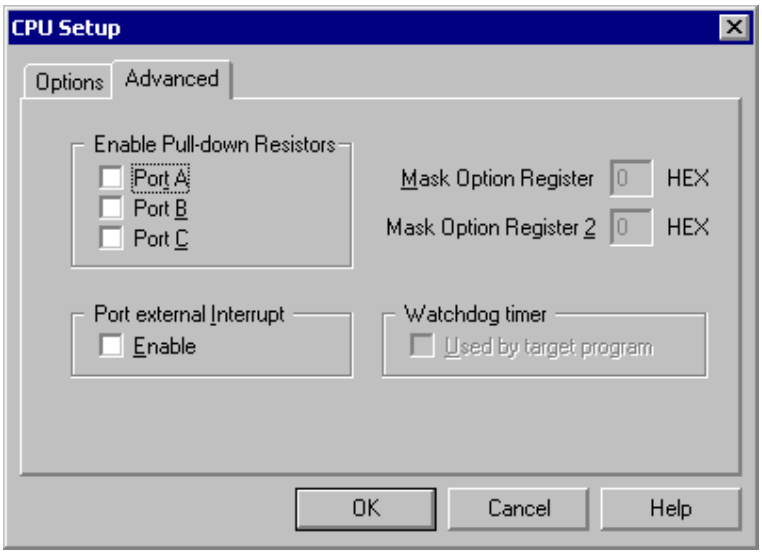
While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of Interrupt Enable flag and handles it correctly.

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 Options



68HC05 CPU Family Advanced Options

Enable Pull-down Resistors

Check the registers for, which you wish to enable pull-down resistors (100 kOhm).

Pull-down resistors operate on the whole port. They are not software programmable via the CPU's 'Pull-Down' registers, because they are reconstructed on the POD.

Port External Interrupt

Enable this option if any port is used as interrupt source. This option is available on CPUs that can use the port/keyboard interrupt.

Mask Option Register

Used on X32 and B16 PODs. Specify the value that you use in your program, but note:

- Pull-down configuration is ignored since it is configured separately.

Mask Option Register 2

Used on the P18 POD. Specify the value that you use in your program, but note:

- Pull-down configuration is ignored since it is configured separately.

Watchdog Timer Used

Check this option if you are enabling the watchdog timer in your program.

Note: The watchdog timer is supported only on P18 and X32 PODs revision C and higher.

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 Memory Access

HC05 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 respectively CPU internal RAM 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.

6 Emulation Notes

6.1 Reserved CPU Resources

The following CPU resources are used:

- 5 bytes of stack are used when the CPU is stopped and interrupts in stop mode are enabled. Otherwise no stack space is used.

6.2 Watchdog Timer Operation

The watchdog timer is supported on X32 and P18 POD. On other CPUs the Watchdog Timer is not supported and should be disabled while debugging.

If the watchdog timer is enabled implicitly through the mask option register, the Emulator will service the watchdog timer periodically when the CPU is stopped.

If the watchdog timer is enabled explicitly (in the program), make sure that the "Watchdog Timer Used" option is checked.

6.3 Wait Mode, Stop Mode

Both modes are supported. While the CPU is in one of the mode, it generates (random) dummy cycles, the debugger looses the control over it and displays HALTED status. The CPU can be brought out of the stop mode by the external IRQ and external RESET. Any interrupt or reset will cause the CPU to exit the wait mode.

6.4 Things to remember

- If the clock source is taken from target, only the oscillator can be used. Quartz or RC oscillators (2 or 3 pin), cannot be used. This does not apply when the clock is taken from Emulator.

- Port interrupts are reconstructed and connected to the IRQs (when the port interrupt is active, the IRQ is active too). This affects BIL and BIH instructions directly.

- Any write to the CPU’s test register (1Fh) can cause CPU malfunction (up to the next reset).

Notes:

Notes: