
Technical Notes

Intel 8051 Family In-Circuit Emulation

Contents

Contents.....	1
1 In-Circuit and Active Emulation 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 Pattern Generator	7
3 Setting CPU options	10
3.1 CPU Options	10
3.2 Standard 8051 Advanced Options.....	12
3.3 Philips H-8xC52 Advanced Options	13
3.4 Memory Mapping	15
3.5 Bank Switching	17
4 Debugging Interrupt Routines	17
5 Memory Access.....	18
6 Bank Switching Support.....	19
6.1 Hardware Configuration.....	20
6.2 Software Configuration	21
7 External XDATA - Custom Application	25
8 Reserved CPU Resources	25
9 Emulation Notes	26
9.1 POD operating mode.....	26
9.2 Writing and Debugging Interrupt Routines.....	26
9.3 Things to remember	26
10 Port Emulation Mode	28

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.

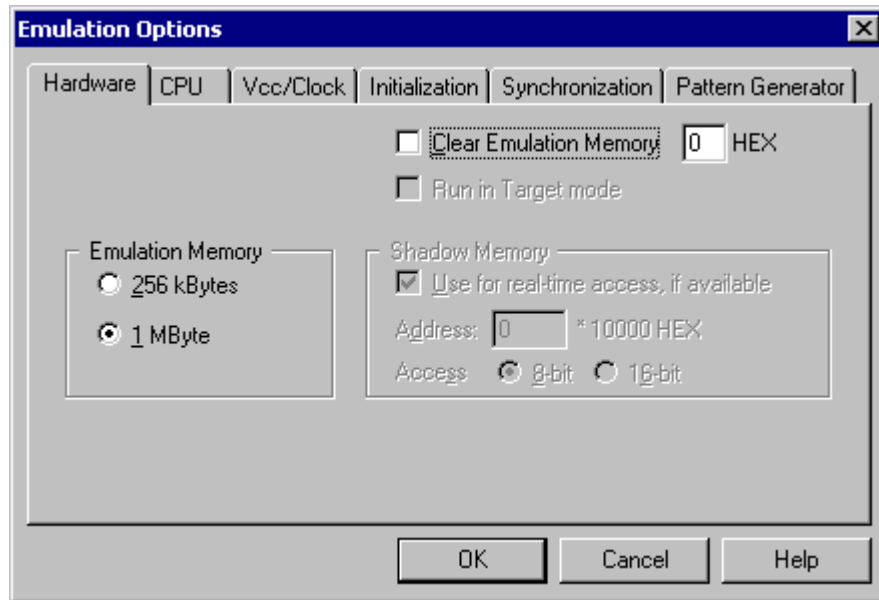
1.3 Port Replacement Information

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 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.

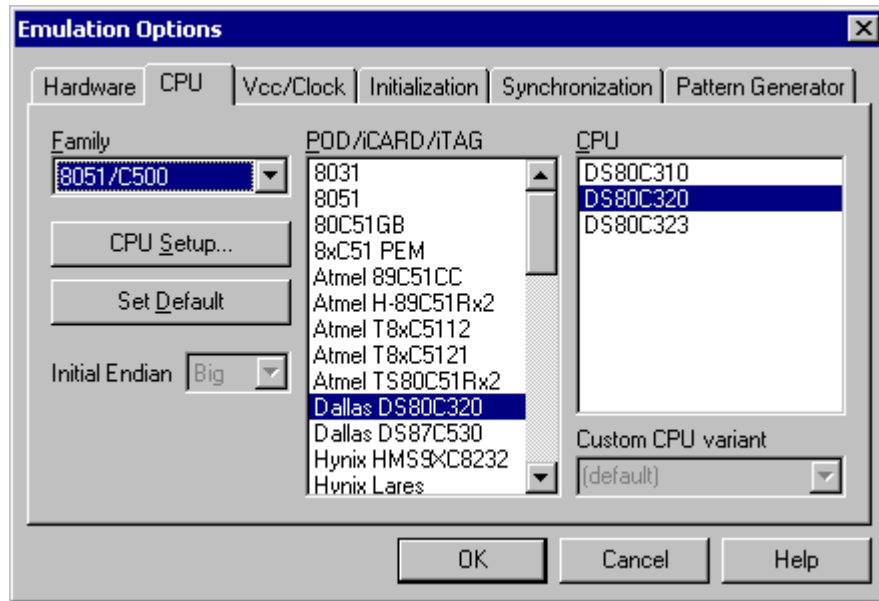
Clear Emulation Memory

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.

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

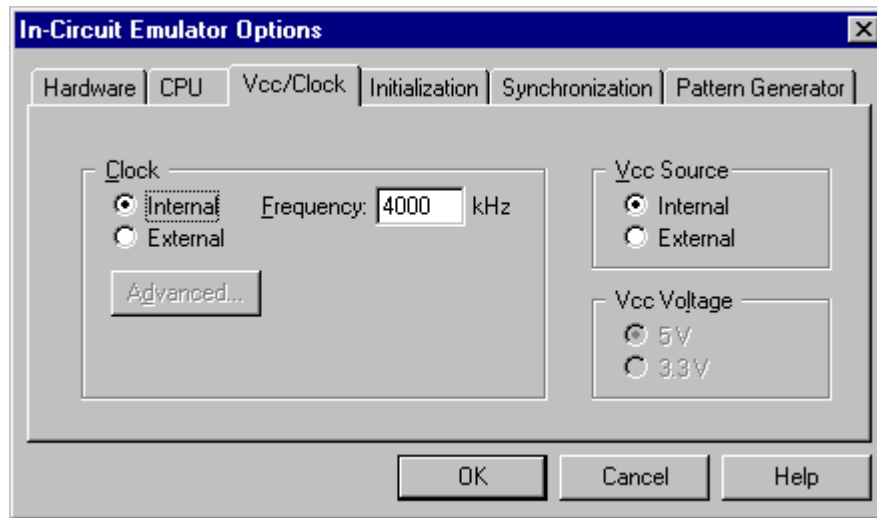
This button will set default options for currently selected CPU. These include:

- 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

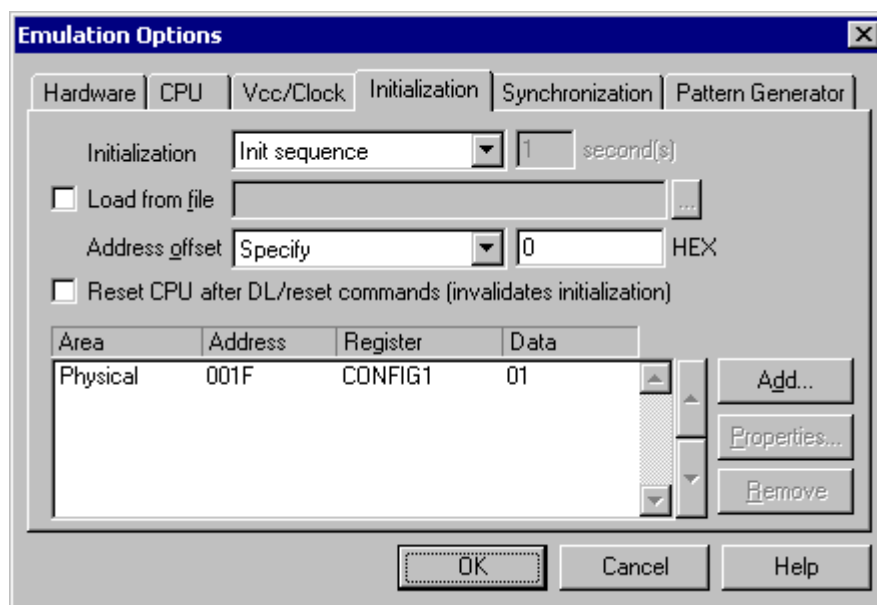
Determines whether Emulator or the target system provides power supply for the CPU.

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:

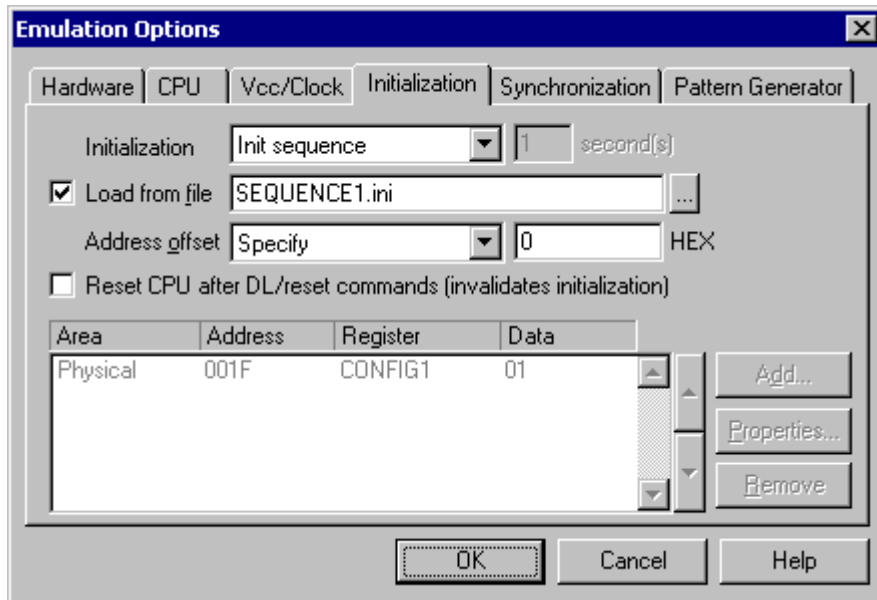
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

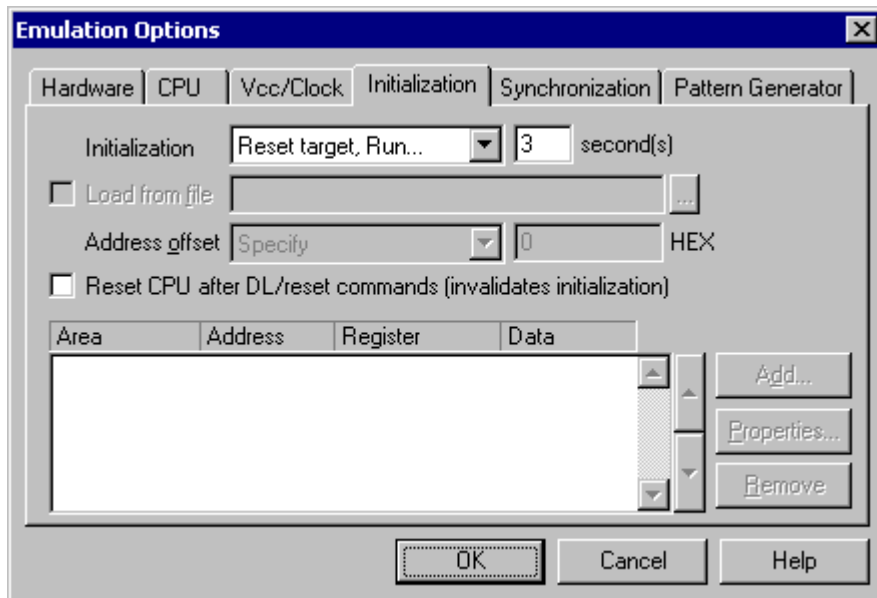
Excerpt from the sample SEQUENCE1.ini file:

```
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 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.

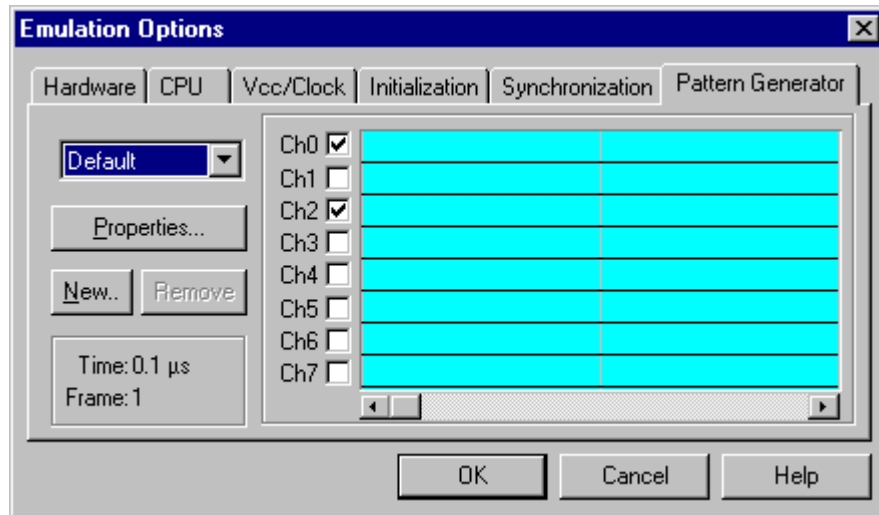
Note: when using the iC4000 system, it has in certain configurations two Pattern Generators: one on the base module and one on the Power Emulator module. The Pattern Generator on the base module is active when the debugging type is set to 'Active Emulation' or 'BDM/JTAG Emulation'; the Pattern Generator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



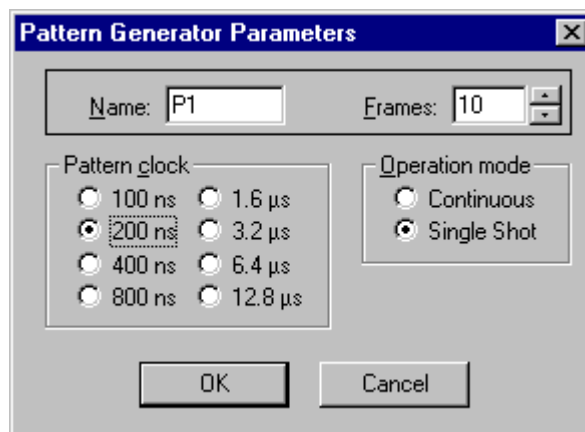
In-Circuit Emulator Options dialog, iC1000/iC2000/iC4000 Pattern Generator page

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.

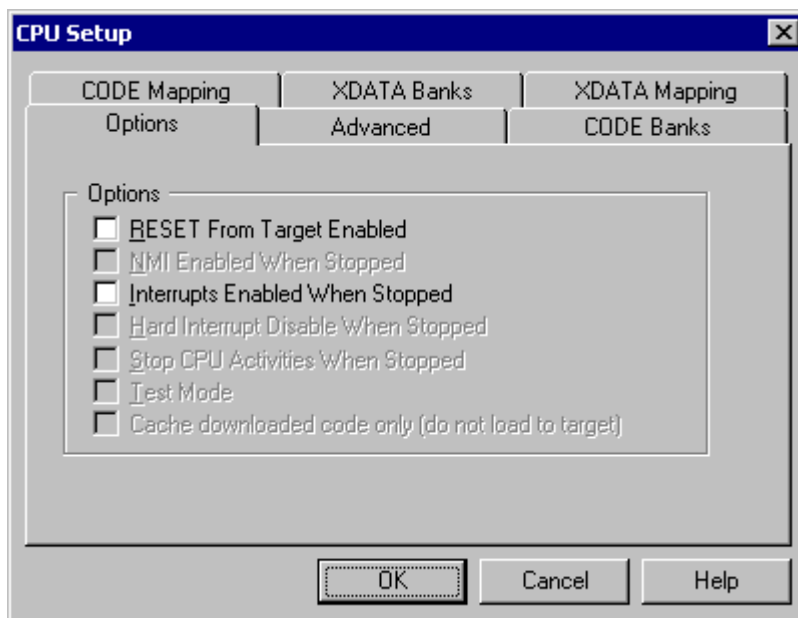
In single shot mode selecting the 'Run Pattern' executes a single pattern shot.

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.

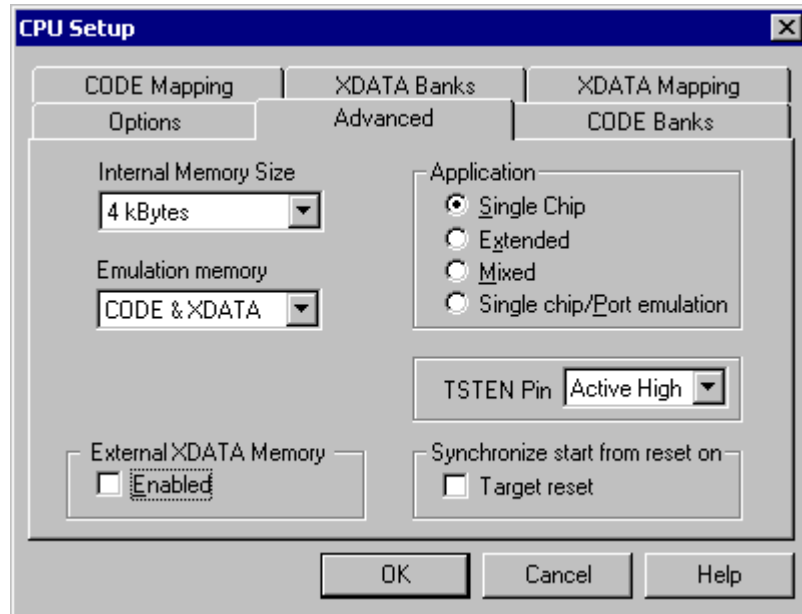
While in stop and executing a single step in the disassembly window there are no problems. During single step in the disassembly window the emulator itself detects any instruction that changes the state of Interrupt Enable flag and handles it correctly.

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 Standard 8051 Advanced Options



8051 CPU Family Advanced Options

Internal Memory Size

Defines the size of the on-chip CODE memory on single chip PODs.

This option is available only on PODs with bondout CPUs.

Emulation Memory

Defines whether only CODE memory is used or if also XDATA memory is used.

External XDATA Memory

Check this option if you are using XDATA memory.

Application

Specify the type of application:

- single chip
- extended mode
- mixed mode (internal ROM + external EPROM)
- single chip/Port Emulation

This option is available only on 8051 Hooks-mode POD and for PODs with Port Emulation Mode. On other CPUs the type of application is determined by the state of the EA pin – with a POD.

For more information on Port Emulation Mode, please see "Port Emulation Mode" on page 28.

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 edge level 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.

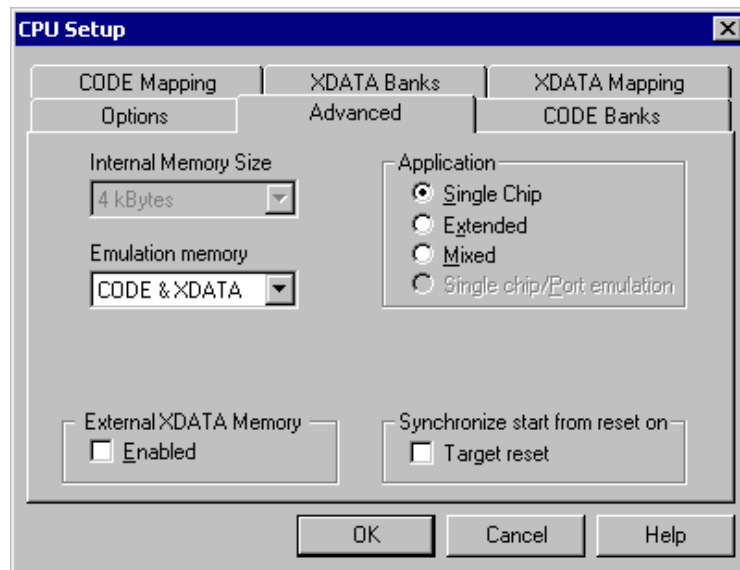
TSTEN pin

This option is available for the Hyundai HMS9XC8032 POD only. It defines whether the TSTEN pin is active high or low.

Reserved Resources

The PSW register does not have the default reset value after debug reset.

3.3 Philips H-8xC52 Advanced Options



Philips H-8xC52 Advanced In-Circuit Emulation Options

Internal Memory Size

Defines the size of the on-chip CODE memory on single-chip PODs. This option is only available on PODs with bondout CPUs.

Emulation Memory

The emulation memory can be selected in this option.

- CODE & XDATA
- CODE only

If the 'CODE & XDATA' option is selected, the emulation memory is divided between the Code and the XDATA and therefore up to eight 64K banks are available. If the 'CODE only' option is selected, the XDATA area is mapped to the target and all memory is available for code, therefore up to 16 64K banks are available.

External XDATA Memory

Check this option if you are using XDATA memory.

Application

Specify the type of application:

- 1 single chip
- 2 extended mode
- 3 mixed mode (internal ROM + external EPROM)
- 4 single chip/Port Emulation

This option is available only on 8051 Hooks-mode POD and for PODs with Port Emulation Mode. On other CPUs the type of application is determined by the state of the EA pin – with a POD.

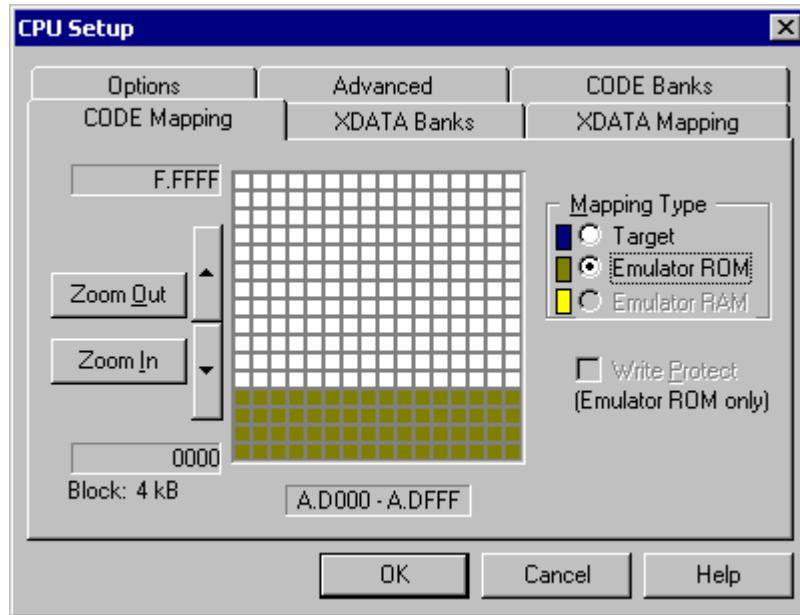
For more information on Port Emulation Mode, please see "Port Emulation Mode" on page 28.

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 edge level 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.

3.4 Memory Mapping

The mapping page displays currently configured memory mapping. There will be one mapping page visible for every memory area that can be externally addressed by the CPU. Usually this will only be CPU's memory, on 8051 family however you will see CODE and XDATA 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.

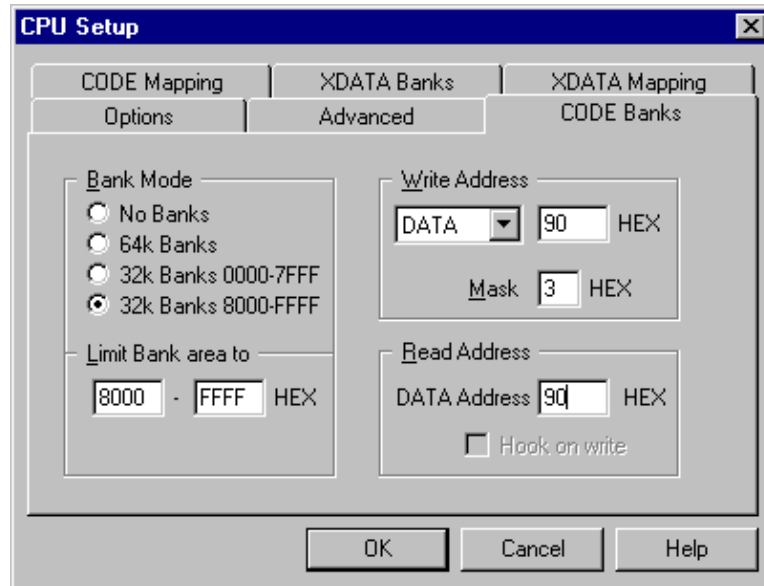
Write Protect

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.

3.5 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. Usually this will only be the CPU's memory, on 8051 family however; you will see CODE and XDATA Banks pages.



CPU Setup dialog, 8031 CODE Bank page

You will only see this page if the currently selected POD supports bank switching.

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

8051 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. XDATA CPU space 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 $n*20\mu s$ 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 Bank Switching Support

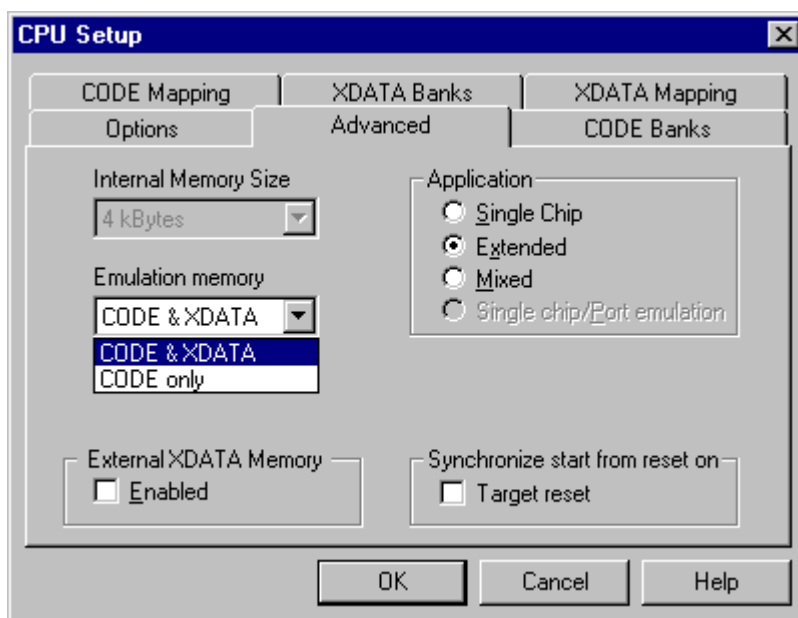
Bank switching is an extension of the CPU's addressable memory. It is used mainly on CPUs where programs have grown larger than 64k.

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.

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.

Bank switching is supported only on some 8051 PODs, supporting extended mode. Both, CODE and XDATA memory area banking are supported. Number of supported banks depends on the amount of emulation overlay memory and emulator configuration.



Emulated CODE & XDATA memory area:

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

Number of available CODE and XDATA banks

Emulated CODE memory area only:

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

Number of available CODE banks

Note: Call stack is not supported for bank applications.

Compiler Support

CODE bank switching is supported on Keil and IAR compilers; you must however configure startup files according to the banking scheme, which you will be using. Refer to your compiler manual for further information.

Since no compiler provides support for XDATA switching, you must be careful when implementing your own bank switch routines. If the C compiler puts variables in the XDATA area, it is best that you use only 32KB banks and put all compiler variables, except those that are to reside in banks, in the lower 32KB. Before you access any variable in the banked XDATA region you must select the correct bank.

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

6.1 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.

Additionally, if 64k CODE bank switching is used, BSC jumper on the POD must be bridged to GND. Otherwise this jumper must be open. If 64k XDATA bank switching is used, BSX jumper on the POD must be bridged to GND. Otherwise this jumper must be open. Refer to specific POD Hardware Reference document for more details on signals related to bank switching.

Example:

8051 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

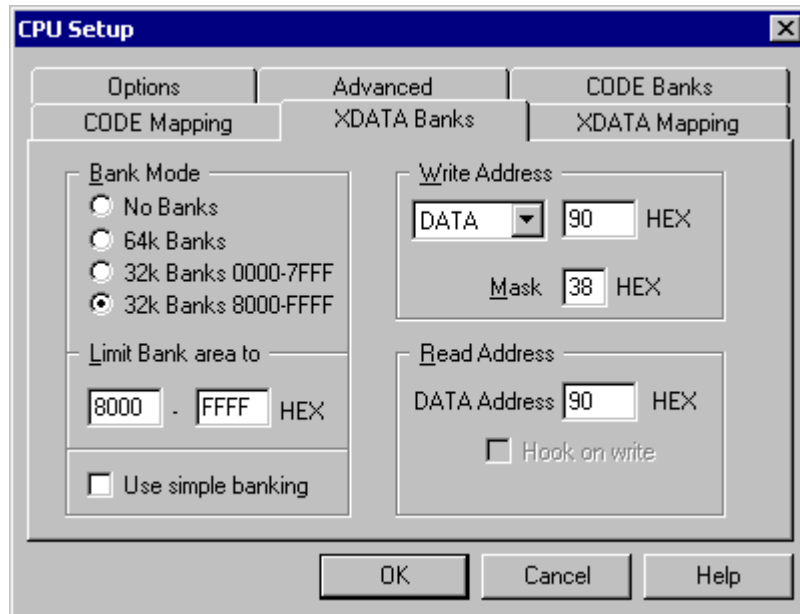
Check whether the necessary signals are already available on the POD.

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).

6.2 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.



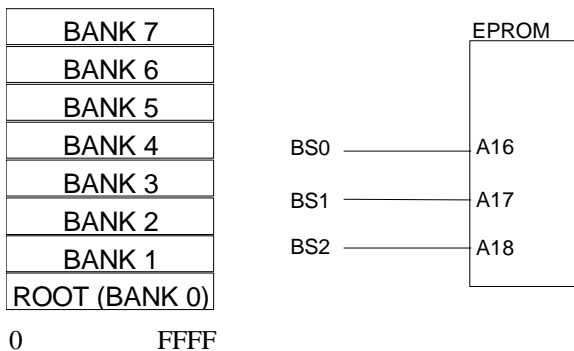
CPU Setup dialog, 8051 XDATA Bank page

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

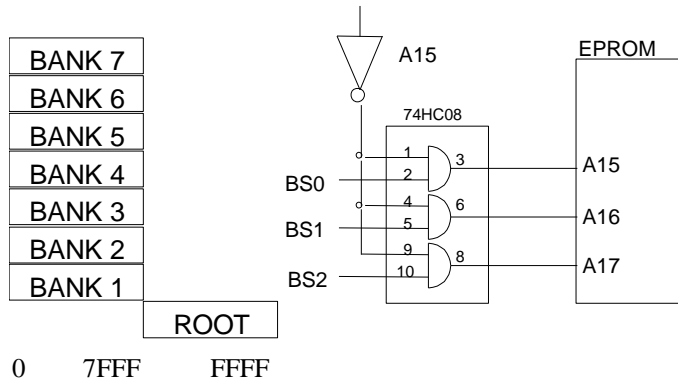


Target chip select logic for ROOT+7 64k banks configuration

An application using 64k banks must have a common area respectively a code that is accessible within any bank without switching the bank. For instance, interrupt routines and bank switching code needs to be available in all banks.

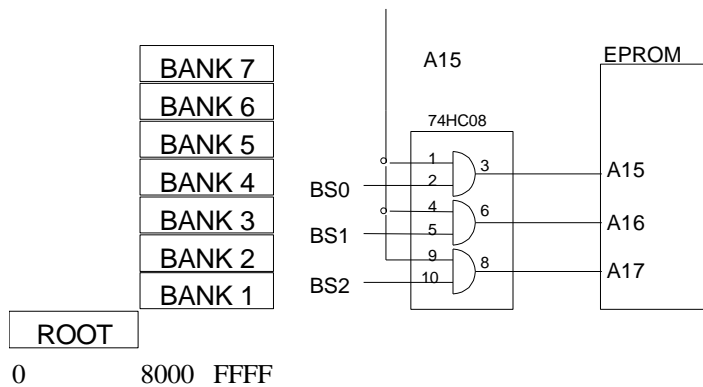
After the emulator obtains the information where the common area resides, it loads that part of code in all emulated banks. In most cases, this common area is defined through 'Limit Bank area to' option, where the user defines, where the bank area is. For instance, when the bank area is limited to 0x0300-0xFFFF range, then the emulator takes 0x0-0x300 as a common area and loads the code that fits into this area in all configured banks (0.000h-0.300h, 1.000h-1.300h, 2.000h-2.300h,...). If Keil compiler is used to build the project and debug information (OMF1 format) is included in the download file, winIDEA is able to extract the common area information from the debug information. Only Keil provides this information in the debug information. In this case, the user doesn't need to limit bank area since winIDEA extracts this information from the debug information and downloads all the code correctly. 'Limit Bank area to' option must be used when downloading a binary or a hex file.

- **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, and 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:

8051 CPU banks are switched through port P1, bits 3 through 5 - yielding 7 banks.

In such case:

- set write address to 90 (address of port P1)
- set memory area to DATA (where P1 is mapped to)
- set mask to 38 (bits 3,4,5 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. **The read address is always related to the DATA memory area.**

For the above example:

- set the read address to 90

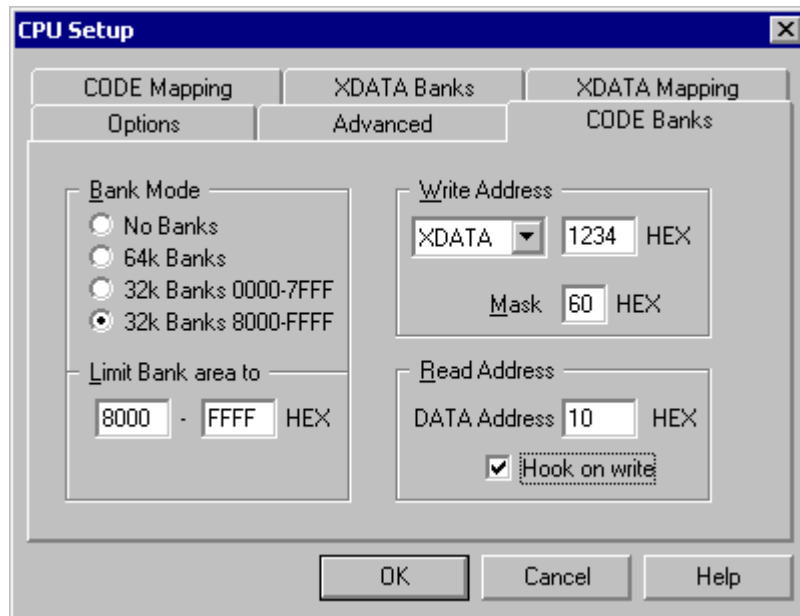
Use Simple Banking

Target mapping is forced if Simple Banking is selected. This feature is useful for applications which use more banks than the emulator supports, for example 128 XDATA banks.

If Simple banking is selected, all run-time bank aware logic does not work reliably, since full address decoding is not possible. Stop mode access is fully operational.

Hook on Write

The in-circuit Emulator must be able to preset banks and to read, which bank the user's program is stopped in. In general, to preset banks the in-circuit Emulator writes to the specific address and to get the information of current bank where program is stopped, it reads specific address. When using bank switching on the 8051 family usually port 1 is used for bank switching. In case of using the port 1 writing/reading to/from DATA 0x90 suffice. Sometimes, ports are used for other purposes and special approaches must be used to implement bank switching. Extra target mechanisms must be implemented.



CPU setup for Philips H-8xC554, CODE Banks settings

Examples:

- 1) Extra latch device mapped as XDATA is implemented on the target to switch banks. The Emulator can write only to the latch and not read back (feature of latch) to get the status of bank switching signals. The easiest way to get bank signals status is when writing to XDATA address to preset banks by user's program, an extra write into DATA area (e.g. 0x10) should be added into the project. When the Emulator requires the bank signals status it just reads from the DATA area (0x10 in our example).
- 2) Sometimes special peripheral devices are used. Memory and bank switching is implemented already in the peripheral device and no switching signals are available externally, even if they are required by the Emulator to support bank switching (e.g. waferscale devices).

For the second case a special solution was implemented. The same mechanism as it is in the peripheral device is implemented on the POD. Equivalent latch that switches bank is implemented in the Xilinx on the POD. In the winIDEA dialog you just enter the XDATA address (latch address) that switches banks in your peripheral device. A mirror image (bank signals status) is written into specific internal RAM location where the Emulator can read current bank switching status. Mirror image address must be entered in the winIDEA as well. Of course, the option 'Hook on write' option must be checked in the 'CPU Setup/Code banks' tab.

When 'Hook on write' option is checked in, the 'XDATA' latch is implemented on the POD instead on the target.

Note: This solution is implemented on the Philips H-8xC554 and Philips H-8xC52 PODs only and supported only by the iC1000, iC2000 and iC4000 in-circuit Emulators.

7 External XDATA - Custom Application

Special attention must be paid to the necessary winIDEA settings in applications, where only port 0 is used to access external XDATA memory. Port 2 may not be used at all or is used as I/O port.

- select application type (typically single-chip) in the 'CPU Setup/Advanced' dialog
- Make sure 'External XDATA memory' is enabled in the 'CPU Setup/Advanced' dialog
- MAP COMPLETE 64kB XDATA memory block to the target
- use 'MOVX @Ri' instruction to access external XDATA memory

Access Breakpoints

It is dissuaded to use access breakpoints on external XDATA variable in such applications. When using access breakpoints, both ports 0 and 2 are decoded by the access breakpoint logic. When setting access breakpoint on the external XDATA variable being addressed by 8-bit address bus (P0), the access breakpoint logic decodes 16-bit address bus (P0 & P2). Port 2, that may have any value (not connected or operates as I/O), is decoded and relevant for the logic too.

Setting trigger or qualifier

When specifying the address item to trigger on (or qualify) the external XDATA variable, make sure you use 'Mask' field in the 'Trace Address Item' dialog. High byte (P2) of the address must be masked since it is not relevant.

Troubleshooting XDATA Access

When having problem with XDATA Access, check the following:

- Check if you have enabled external XDATA in the 'CPU Setup/Advanced' tab.
- RD, WR and ALE signals are required and active during XDATA access. All three signals are connected directly from the CPU on the POD to the target. The emulator has no influence on RD and WR generation. They are not generated even if the CPU addresses the internal XDATA (XRAM) memory. If RD and WR signal are not active on your target side then measure them on the CPU on the POD. If they are present on the CPU on the POD but not in the target then the problem is the POD/target connection. Verify the connections. There may also be some short circuit between "XDATA access" signal (RD, WR, ALE) and some other target signal.

8 Reserved CPU Resources

The following CPU resources are used:

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

9 Emulation Notes

9.1 *POD operating mode*

If a single chip POD is used standalone (not attached to a target board), the CPU always runs in single chip mode. When used in a target system, the EA pin determines the operation:

- EA=0 forces extended mode
- EA=1 forces single chip mode

This only applies to bondout and standard Hooks PODs. When using Extended Hooks PODs, the EA setting is ignored. The operation is defined in the software.

On mixed mode applications (CODE memory available by internal ROM is used from the CPU, the rest from external EPROM):

- internal memory must be mapped to the Emulator
- the size of internal memory (on bondout CPU PODs) must be configured properly

9.2 *Writing and Debugging Interrupt Routines*

To allow background interrupt execution, interrupt routines must meet the following conditions:

- The following CPU registers must be preserved:

DPTR, ACC, B

R0, R1, R2 and R3 from the current bank.

- Interrupt routines must return with corresponding instruction (RETI).
- The return address must not be changed in the interrupt routine.

RTX-51

The RTX-51 executes a RETI in the dispatcher to lower the interrupt level prematurely. Once finished the dispatcher returns with a RET instruction.

On extended mode PODs this is allowed if the (dispatcher) routine is not located between F000h and FFFFh.

9.3 *Things to remember*

ONCE, IDLE, PowerDown Mode

Neither IDLE nor Power Down nor OnCE modes are supported by any POD, except by the 8031/DS80C320 POD (IC81025).

Internal Watchdog

When the CPU features internal watchdog, it must be disabled while debugging. Otherwise, the emulation fails.

Checksum

When performing any kind of checksum in the emulated (code) area, note that all breakpoints must be removed before, otherwise the results are distorted.

Ports

Port 0 is always an open-drain port when using single-chip PODs.

Ports on Enhanced Hooks PODs (C500 family), Hooks PODs H-8xC52 and H-80C52

Original port 0 and port 2 are used for emulation and rebuilt by standard integrated circuits on the POD therefore electrical characteristics are changed.

Whenever operating close to electrical limits and having problems with rebuilt ports please check pull-up and pull-down resistors. They shouldn't be too strong, neither too weak. Check the voltage level. Try to withdraw from voltage limits.

Note: When using Philips Hooks PODs, as clock source a standalone clock must always be used, either internal or external from the target. The Clock signal is not directly connected to the CPU pin because the clock signal is required for HOOKS mode implementation. Thus when external clock is selected, the oscillator must be used in the target and not the crystal, which would not oscillate at all since it's not connected directly to the CPU pin.

10 Port Emulation Mode

Note: Port Emulation Mode is supported on few 8051 PODs only.

Port Emulation Mode is a special mode for emulation of single-chip applications. Normally when emulating single-chip applications the CPU is operating in a mode, similar to the extended mode, where the program is executed from an external memory. In extended mode the CPU uses ports P0 and P2 as address/data bus, while the P0 and P2 ports are mapped to the SFR area. This means that writes and reads to or from the port are not visible on the external bus and typically there are no standard port replacement solutions available.

The basic idea for Port Emulation mode is to map the ports P0 and P2 to an area, that can be seen and a typical port replacement can be built. The areas available on 8031 are CODE and XDATA areas. A simpler solution would be to use the XDATA area, but in this case also P3.6 and P3.7 would be lost, since the CPU uses these pins for read and write in XDATA cycles. This would require target modification; therefore the CODE area is used. The emulator uses the MOVC instruction to access data. Since this function is designed for reading data only, the "Write by Read" technology is used.

To use the Port Emulation Mode, the software in the target application must be modified. To make the transfer from the debug mode (with Port Emulation Mode) to the release version as simple as possible, the change to the program is advised in the way ports are accessed through functions. This would mean generating four functions for P0 and P2 access:

- void writeP0(unsigned char value);
- unsigned char readP0();
- void writeP2(unsigned char value);
- unsigned char readP2();

With this definitions, different access types can be set if the source is built for debugging or for release purposes.

Port redirection functions implemented in C

File: Port_ACC.h

```
#ifndef __PORT_ACC_h__
#define __PORT_ACC_h__

sfr PA_P0=0x80;
sfr PA_P2=0xA0;

#ifdef PA_REDIRECT

#define WRITE_P0(VALUE) PA_WRITE_P0(VALUE)
#define READ_P0() PA_READ_P0()

#define WRITE_P2(VALUE) PA_WRITE_P2(VALUE)
#define READ_P2() PA_READ_P2()

void PA_WRITE_P0(unsigned char byValue);
unsigned char PA_READ_P0();
void PA_WRITE_P2(unsigned char byValue);
```

```

unsigned char PA_READ_P2();

#else // PA_REDIRECT

#define WRITE_P0(VALUE) PA_P0=VALUE
#define READ_P0() PA_P0

#define WRITE_P2(VALUE) PA_P2=VALUE
#define READ_P2() PA_P2

#endif // PA_REDIRECT

#endif // __PORT_ACC_h__

File: Port_ACC.c

#include "PORT_ACC.h"

#ifdef PA_REDIRECT

#define CBYTE ((unsigned char volatile code * ) 0)

void PA_WRITE_P0(unsigned char byValue)
{
    unsigned char by1, by2;
    by1=CBYTE[0xFFFC];
    by2=CBYTE[byValue];
}

unsigned char PA_READ_P0()
{
    unsigned char by1;
    by1=CBYTE[0xFFFD];
    return CBYTE[0];
}

void PA_WRITE_P2(unsigned char byValue)
{
    unsigned char by1, by2;
    by1=CBYTE[0xFFFE];
    by2=CBYTE[byValue];
}

unsigned char PA_READ_P2()
{
    unsigned char by1;
    by1=CBYTE[0xFFFF];
}

```

```

    return CBYTE[0];
}

```

```

#endif // PA_REDIRECT

```

Port redirection functions implemented in assembler, assuming parameters are transferred in register R7

```

;-----
WRITEP0:
$IF PA_REDIRECT
    MOV A,R7
    MOV P0,A
$ELSE
    CLR A
    MOV DPTR #0FFFCH
    MOVC A,@A+DPTR
    CLR A,
    MOV DPL,R7
    MOVC A,@A+DPTR
$ENDIF
    RET
;-----
READP0:
$IF PA_REDIRECT
    MOV A,P0
$ELSE
    CLR A
    MOV DPTR #0FFFDH
    MOVC A,@A+DPTR
    MOV DPTR #0H
    CLR A,
    MOVC A,@A+DPTR
$ENDIF
    MOV R7,A
    RET
;-----
WRITEP2:
$IF PA_REDIRECT
    MOV A,R7
    MOV P2,A
$ELSE
    CLR A
    MOV DPTR #0FFFEH
    MOVC A,@A+DPTR
    CLR A,
    MOV DPL,R7
    MOVC A,@A+DPTR

```

```

$ENDIF
    RET
;-----
READP2:
$IF PA_REDIRECT
    MOV A,P2
$ELSE
    CLR A
    MOV DPTR #0FFFFH
    MOVC A,@A+DPTR
    MOV DPTR #0H
    CLR A,
    MOVC A,@A+DPTR
$ENDIF
    MOV R7,A
    RET
;-----

```

The Port Emulation Function always, as visible above, consists of two sequential MOVC operations. The first one starts the function, the second one executes it.

To start the sequence, access to one of four reserved addresses (according to the required function) must be performed.

The addresses are:

- 0xFFFC for write to P0
- 0xFFFD for read of P0
- 0xFFFE for write to P2
- 0xFFFF for read of P2

The second MOVC will execute the sequence.

During the sequence interrupts don't have to be disabled, although other MOVC instructions are executed in the interrupt routine – when the interrupt is completed, the sequence will finish successfully.

The contents of ports P0 and P2 are also visible in the SFR window – the emulator performs the same sequence to read or write to the port. The limitation with the display though is that the emulator can not access neither of ports P0 or P2, if the user's program has started a sequence and it is not yet completed (if single steps are performed). This means that if the target has started the sequence, for example, to write to P2 with the first MOVC instruction, the emulator can not access neither P0 nor P2 for neither read nor write, until the sequence from the target side to write to P2 has completed. Any read or write to these ports made in these time will not corrupt the target sequence, it will just not be executed. This situation can occur only if the sequence is performed in single step mode (instruction by instruction).

If Port Emulation Mode is used, the emulator decodes all instructions performed by the CPU in real-time and in case a wrong instruction is used to access ports P0 or P2 (for example a direct write), the program stops and the emulator displays an error message that a wrong instruction has been used.

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.