
Technical Notes

Zilog Z80 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
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 Advanced Options.....	12
3.3 Debug Areas.....	13
3.4 Bank Switching	14
3.5 Memory Mapping	15
4 Debugging Interrupt Routines	16
5 Memory Access	17
6 Bank Switching Support.....	18
6.1 Z80 Family Bank Switching	18
6.2 Hardware Configuration.....	18
6.3 Software Configuration	18
7 Emulation Guidelines when using RETI Refetch mechanism.....	21
7.1 Background	21
7.2 Avoiding Bus Collisions	22
7.2.1 Emulating Z80.....	22
7.2.2 Emulating Z180 and Z182.....	22
7.3 RETI Refetch - Differences between masks	22
8 Emulation Notes	23
8.1 Reserved CPU Resources.....	23
8.2 Requirements	23
8.3 Z180 MMU	23
8.4 Real-Time Watch	23
8.5 Writing and Debugging Interrupt Routines.....	23
8.6 M1 (LIR) Signal.....	24
8.7 Trace	24
8.8 Trigger/Qualifier Settings	24
8.9 Other Things to Pay Attention To	24

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.

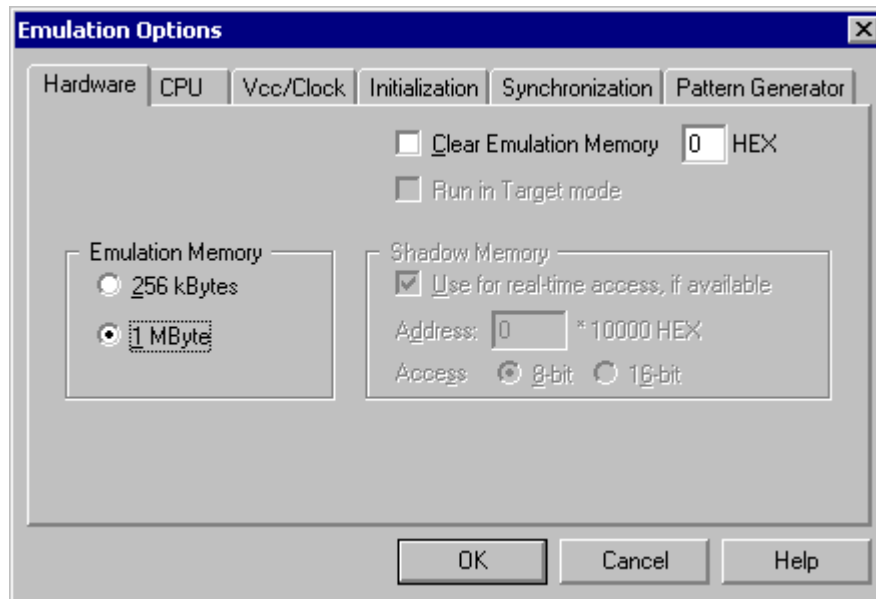
1.2 Common Guidelines

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.

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.

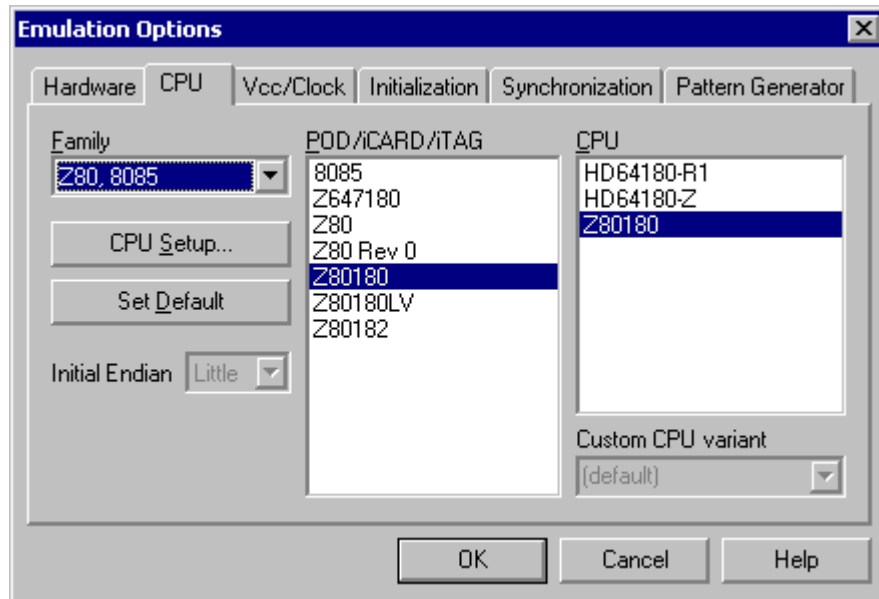
Clear Emulation Memory

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

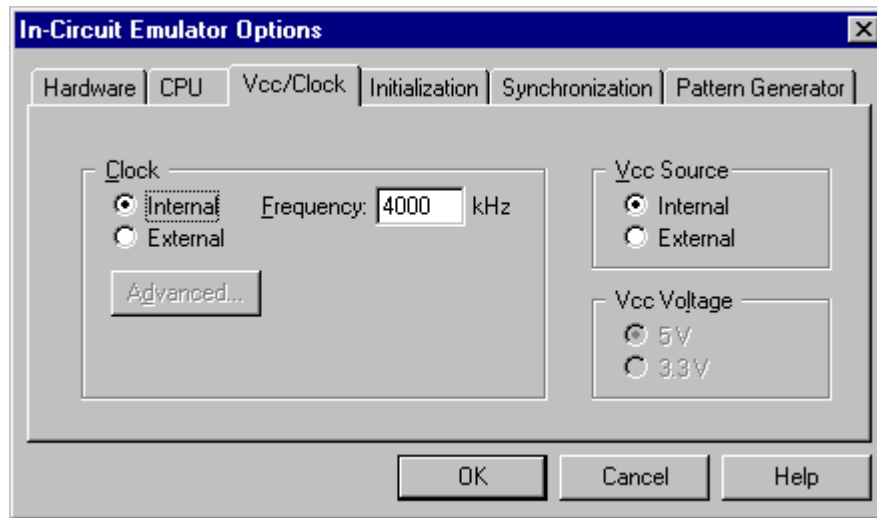
This button will set default options for currently selected CPU. These include:

- 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

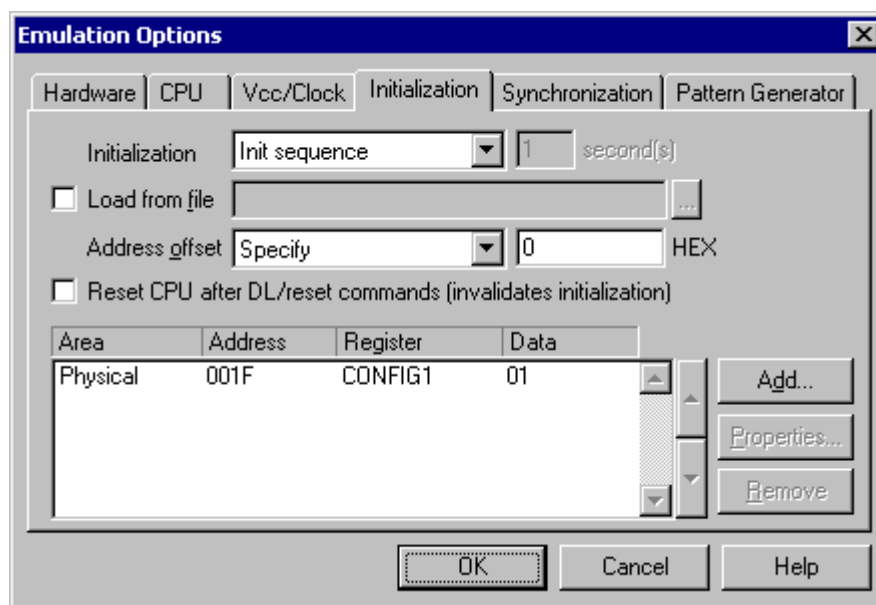
Determines whether Emulator or the target system provides power supply for the CPU.

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:

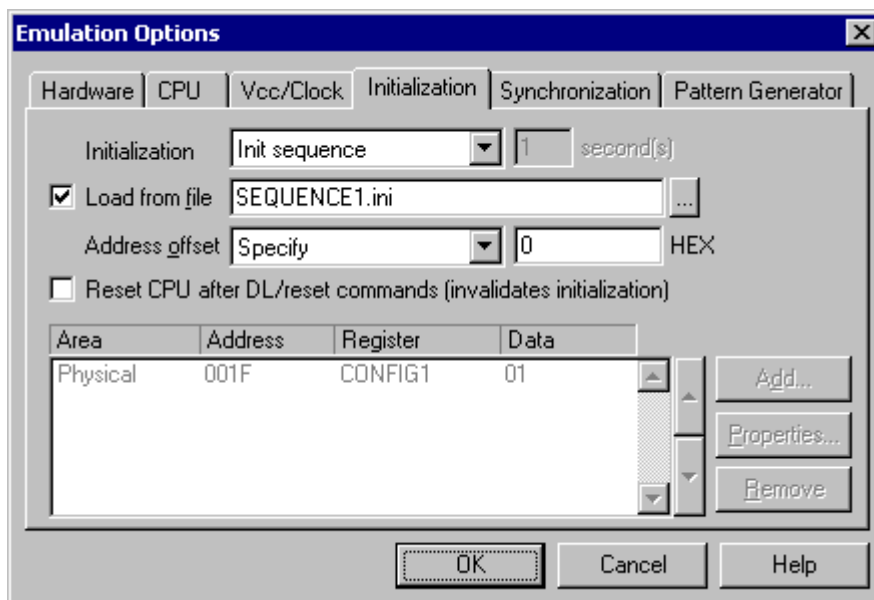
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

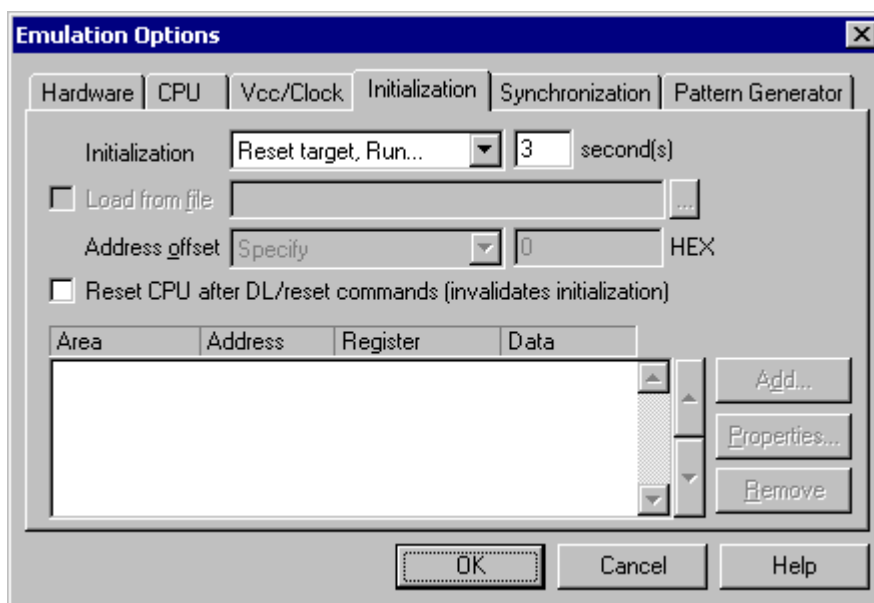
Excerpt from the sample SEQUENCE1.ini file:

```
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

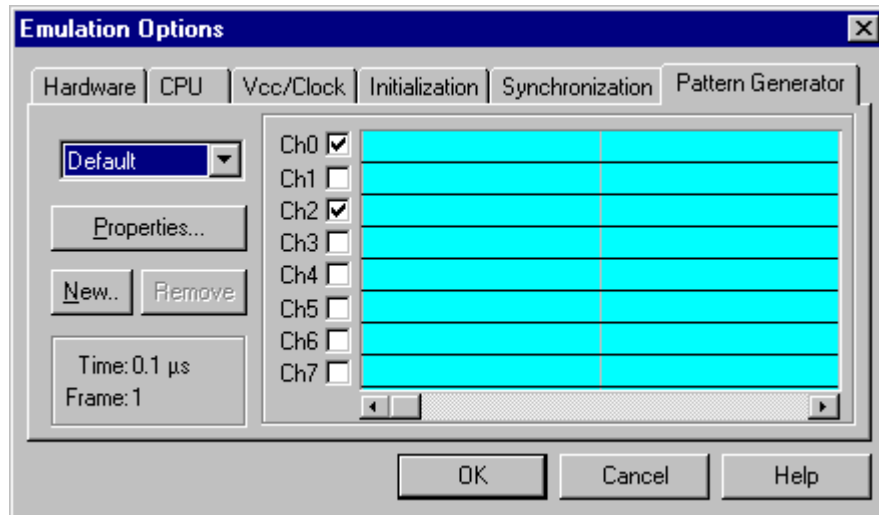
iC1000, iC2000 and iC4000 provide an 8-channel waveform programmable pattern generator capable of continuous or single shot operation at up to 10MHz-clock rate with up to 512 samples.

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.



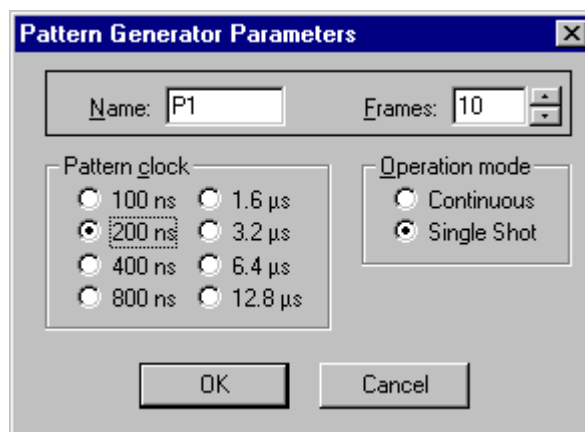
In-Circuit Emulator Options dialog, iC1000/iC2000/iC4000 Pattern Generator page

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.

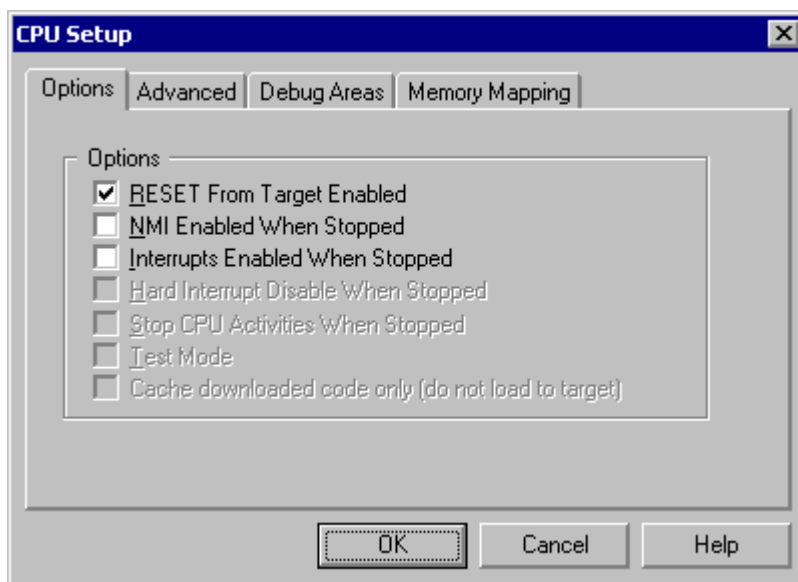
In single shot mode selecting the 'Run Pattern' executes a single pattern shot.

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.

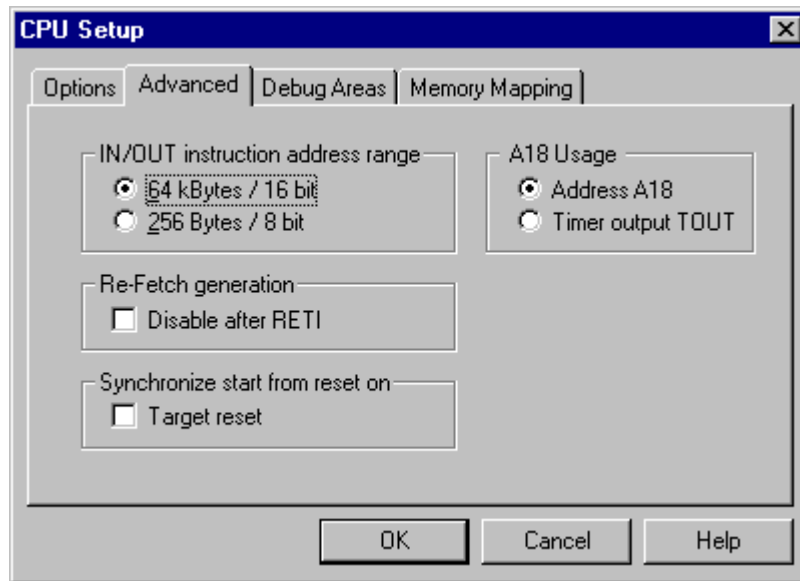
For example, interrupts are active and the program is stopped. The emulator remembers the Interrupt Enable flag state and disables interrupts. Now the user executes single steps in the disassembly window and, for example, once the SWI instruction (software interrupt) is stepped. At this moment, the CPU 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.

3.2 Advanced Options

These settings are available for Z180 derivatives only.



Z180 Family Advanced Options

IN/OUT instruction address range

specify what address range you will use to access the I/O address area. If you only use 256 bytes and address it with instructions:

IN A,(n)
OUT (n),A

The register A content is also put on the high byte of the address bus, which makes usage of I/O access breakpoints very difficult. In this case select the 256 bytes option.

A18 Usage

The user has to specify how he uses the MCU's line A18/TOUT since it can operate as address A18 or as timer output TOUT.

Re-Fetch Generation Disable After RETI

When this option is turned on, the emulator does not open the buffer line on the POD towards the Target, so removing EPROMs from the target is not necessary.

Note: This option can only be used if the target does not contain any periphery that generates interrupts or if such periphery is installed that does not require RETI Refetch. If this option is turned on on a system with the periphery that requires Refetch, the interrupts will malfunction.

This option is only available on iC1000 and Power Emulator. iC181 does not support this function.

Synchronize Start From Reset On

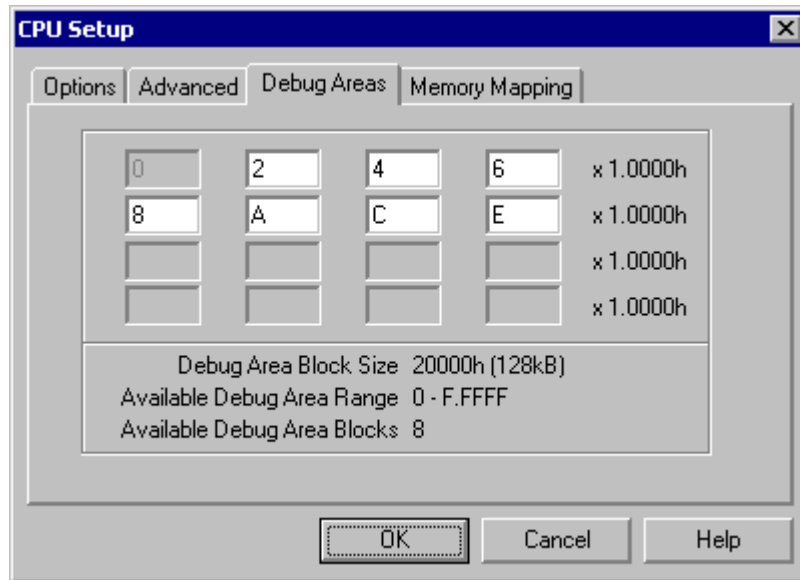
The target reset signal resets the CPU immediately. However, when the target reset becomes inactive, the CPU reset is overdue for few hundred milliseconds by the emulator. If external watchdog is active, the CPU restart must be synchronized with the external watchdog, therefore "Reset from target enabled" 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.

3.3 Debug Areas

These settings are available for Z180 derivatives only.

The Debug Areas page determines address ranges within, which the Emulator can debug. Depending on the Emulator that you are using and its emulation memory size, different numbers of such areas are available. The size of the debug areas depends on the Emulator (64k on iC181 and 128k on iC1000 and the PowerEmulator).



CPU Setup dialog, Debug Areas page

This page is visible only if the size of emulation memory cannot cover the entire memory addressable by the CPU.

Debug area blocks that are required to be present on certain locations (like 0 on 80186) are displayed but cannot be moved.

Since debug area blocks can only be moved on multiples of 64k (128k on the PowerEmulator), you are required to enter only the high word of the address.

Note: Any changes on this page will reset memory mapping page settings.

Large Memory Applications (>1MB)

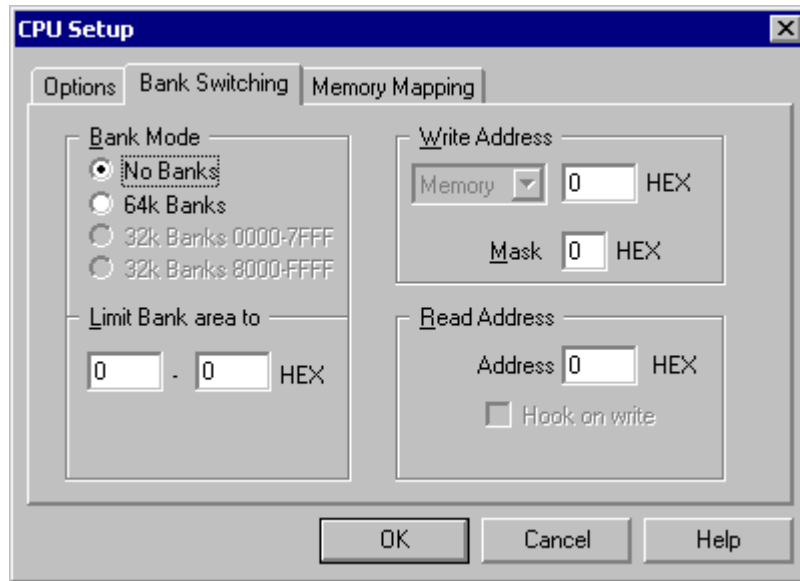
On large memory applications (determined by the POD type) the debug area block size is 256 KB.

Outside debug areas all memory accesses are routed to target. Additionally the following restrictions apply:

- No breakpoints are available
- The trace cannot trigger on addresses outside a debug area
- Shadow memory cannot be used

3.4 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. Bank Switching is supported on the Z80 POD only.



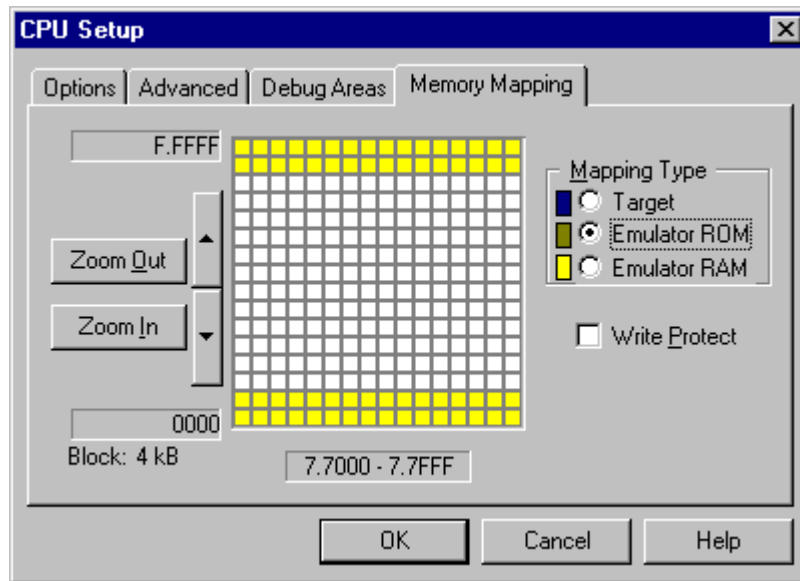
CPU Setup dialog, Bank Switching page

Note: Only 64k bank switching is supported.

For more information on Bank Switching, please see Chapter 6, Bank Switching Support.

3.5 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 Memory Access

Z80/Z180 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 $n \cdot 20\mu s$ 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.

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.

6.1 Z80 Family Bank Switching

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

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

Compiler Support

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

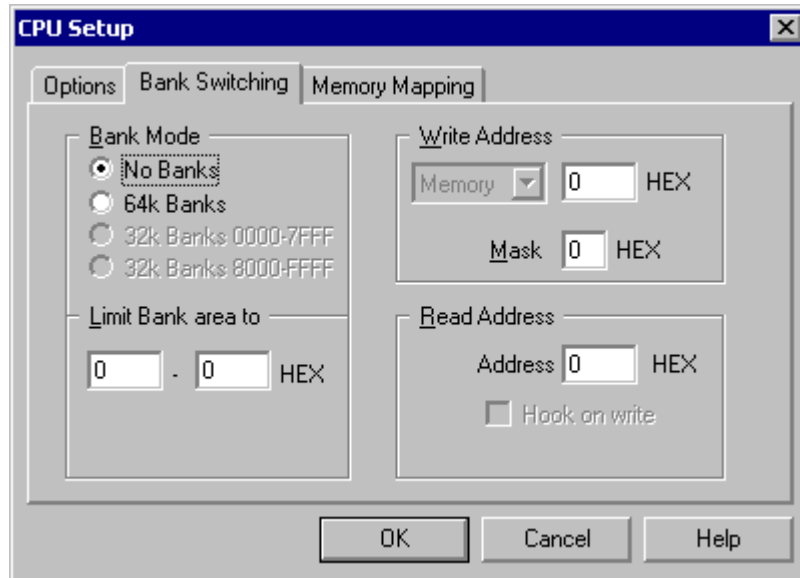
Since the Emulator does not support Bank 0, the user must not put any program modules in Bank 0. To do this, simply omit all references to Bank 0 in the linker command file.

6.2 Hardware Configuration

The bank switching logic must be implemented externally to provide information about the target's chip select logic, i.e. the address of a memory-mapped latch of the chip select unit.

6.3 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.



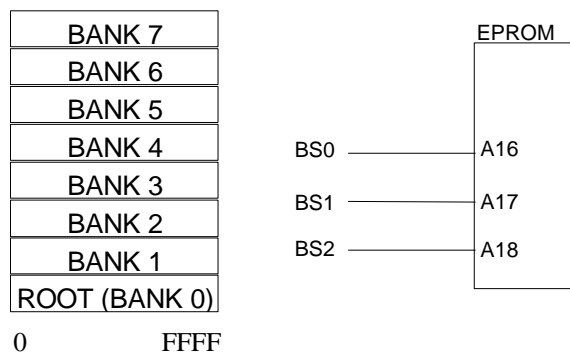
CPU Setup dialog, Bank switching page

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

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

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:

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.

For the above example:

- set the read address to 90

7 Emulation Guidelines when using RETI Refetch mechanism

When emulating the Z80, Z180 or Z182 in the target application using RETI Refetch interrupt mechanism, some emulation guidelines must be taken into consideration. The emulation guidelines can be ignored when all memory is being mapped to the target.

7.1 Background

When the CPU executes RETI instruction (0xED, 0x4D), the data bus values 0xED and 0x4D must occur on the target data bus if some target peripheral devices use interrupts. When specific peripheral device whose interrupt was just serviced, detects RETI (0xED4D) on the data bus, it signals to the other peripheral devices with lower interrupt priority that the CPU can service now them if there is some pending interrupt.

In case of the Z180 and Z182, after the RETI is being fetched, the CPU re-fetches RETI (0xED, 0x4D) once again. Second RETI occurs on the data bus consecutively after the first one.

In case of Z80, the refetch is not supported. At every read cycle from the emulator the POD forces all data also to the target side. Because of this all programs (all memory) should be removed from, which the CPU can execute the program normally.

Using the emulator, the actual problem is the data bus. Target peripheral devices must see all the CPU's RETI fetches, so the peripheral device, which interrupted the CPU, can signal others that its interrupt was serviced. Therefore, the request is that any executed RETI instruction by the CPU (on the POD) is visible on the target data bus, even when the code is executed from the emulation memory. When the CPU (on the POD) refetches RETI instruction from the emulation memory, the buffer on the POD is opened to allow data bus propagation to the target. Bus collision may occur in both cases, when the target memory device is connected directly to the CPU or when it is connected via the target buffer.

Different code in the target memory device (EPROM) than in the emulation memory, where the download file is loaded, is actually the reason for the data bus collision.

Typical Application Scenarios

As mentioned, target memory devices can be connected either directly to the CPU or via the buffer. Note that in both cases the program is executed from the emulator.

- Target memory device connected directly to the CPU

For example, the target EPROM has fixed code and 0x1000 and 0x1001 addresses contain values 0x12 and 0x34. At the same time, RETI instruction is located at the address 0x1000 on the emulator side. While debugging the application, the sources are modified and consequently the location of RETI alters in the address space. While the CPU on the POD executes RETI from the emulation memory, the POD forces the same value to the target data, so the adequate target peripheral device can detect that its interrupt was serviced. A collision on the data bus occurs because POD forces 0xED and 0x4D, and the target EPROM forces 0x12 and 0x34 to the data bus.

- Target memory device connected to the CPU via buffer

Typically, target buffer is transparent when CPU's RD signal is active. It is a similar situation as in the previous case.

For example, while the CPU on the POD executes RETI from the emulation memory, the POD forces the same value to the target data, so the adequate target peripheral device can detect that its interrupt was serviced. A collision on the data bus occurs because POD forces 0xED and 0x4D, and the target buffer forces 0x12 and 0x34 to the data bus.

7.2 Avoiding Bus Collisions

Note that collisions on the data bus may damage the emulator or the target.

7.2.1 Emulating Z80

Collisions on the data bus don't occur if the code in the emulation memory is the same (100%) as in the target memory.

7.2.2 Emulating Z180 and Z182

A data bus value of both, the emulation memory and the target memory, must match 100% when RETI instruction is forced by the emulator (emulation memory).

If above conditions cannot be guaranteed, the following must be done:

- Target memory devices must be removed on addresses that are mapped to the emulator
- Buffers, separating system data bus from the CPU data bus must be removed and associated bus lines bridged

7.3 RETI Refetch - Differences between masks

To emulate the Z180 without limitations, the emulator has to distinguish among all CPU cycles, which are decoded from MREQ, IORQ, RD, WR,... signals, including M1. All signals are generated constantly, except M1 can be disabled by the M1E bit in the OMCR register.

Since M1 is required during emulation, the emulator auto detects the instruction, which disables M1, enables it back and simulates M1 to the target like it would be disabled. The POD behaves to the target exactly the same as the original CPU by generating M1 during the RETI refetch.

The CPU generates refetch of the RETI instruction independent from the M1E register. It doesn't matter, whether M1 is enabled or disabled.

Above behavior holds for "old" CPUs. It has been discovered that newer Z180 CPUs generate refetch only when M1 is disabled. Since the emulator re-enables M1 after it is disabled by the target application, it cannot recognize anymore when to force (simulate) RETI refetch to the target. Therefore, such CPU cannot be emulated without restrictions.

Unfortunately, 'new' and 'old' CPU designations don't differ at all. Likewise, there is no remark concerning this issue in the CPU datasheets. Zilog has changed the operation of RETI refetch without any extra notification. The only way to distinguish the CPUs is to measure whether RETI refetch is generated or not.

Hence, to emulate the Z180 without restrictions, an 'old' CPU must be inserted in the POD or M1 must be left enabled when inserting a 'new' CPU.

8 Emulation Notes

8.1 Reserved CPU Resources

No CPU resources are used.

8.2 Requirements

Memory devices on addresses that are mapped to the Emulator, must be removed from the target system, because the Emulator will assert data from each memory read cycle on the data bus. It is necessary to insert a RETI instruction to acknowledge interrupt servicing to interrupting peripherals that request this type of acknowledge (KIO, PIO, etc.). If the memory device is not removed, collisions on data bus and damage to the Emulator can occur.

In addition, no buffer or serial resistor may be between the CPU data lines and the system data bus. This is because the asserted RETI instruction is issued by the emulation CPU and not by the target EPROM as is the case when the target system runs without an Emulator.

If you have such a target you must try to bridge the interfacing device or contact technical support.

8.3 Z180 MMU

On the iC181 Emulator the block on CPU's logical zero (0-FFFh) must be mapped to physical zero (after MMU translation).

This limitation does not apply on the iC1000 and the PowerEmulator.

8.4 Real-Time Watch

For the Z180 only physical addresses are accessible in the real-time watch window. You can specify logical addresses in the (normal) watch window. If you write in the real-time watch window: "Physical:0x4000" it will work OK. If you write "Logical" instead of "Physical" the memory probably won't be accessible. You can calculate, which address you get from Logical 0x4000.

8.5 Writing and Debugging Interrupt Routines

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

- All CPU registers must be preserved, i.e. on exit from a routine all registers must hold the value they held when the routine was entered.
- Interrupt routines must return with corresponding instruction (RETI for IRQ, RETN for NMI).
- The return address must not be changed in the interrupt routine.

If only the first interrupt is generated and serviced

Some processors that generate a re-fetch RETI instruction (80180, 80182, 64180Z, 647180, 648180,...) that is meant to acknowledge the interrupt to the interrupting device. The Emulator will output the (repeated) RETI instruction on the target bus, even if the program code memory is mapped to the Emulator memory. If your target contains a device that responds on the data bus that on the address (of the RETI instruction in the Emulator memory), it will cause collision on the data bus.

If this occurs remove the device in question from the target (since it is not used anyway), or use the interrupt routines from the target memory when they are debugged.

Also note: On the Z80 POD, the NMI interrupt is synchronized with the instruction FETCH cycle; other interrupts and PODs introduce no interrupt latency.

8.6 M1 (LIR) Signal

Before the LIR signal is disabled, make sure that the SP is set to a point to a valid RAM location. This may not be necessary in the target system, but is required by the Emulator.

8.7 Trace

In the trace window, physical address is displayed in the Address column, but in the Content column logical address is displayed when code is disassembled.

8.8 Trigger/Qualifier Settings

The Trigger and Qualifier must always be set to a physical address. If they are set to a logical address, the trigger/qualifier condition will not occur. From a logical address the emulator cannot decode the physical address because of the banking system of the Z180 and the calculation of addresses depends completely on the target system configuration.

8.9 Other Things to Pay Attention To

- Be sure to have a good clock from the target if a crystal is used. Zilog has very tight limitations on the clock (duty cycle, rising and falling edges).
- When Z80 is used and if the target periphery uses the MCU clock for its own operation, then the clock source must be set to external in the 'Vcc/Clock' tab. The reason is that the MCU doesn't provide any clock output that could synchronize the MCU and the periphery. The Z180 has a clock output.
- If you use the adapter for the (TMP)Z84C(0)15/13, you can only use the external clock.
- Use the RESETOUT output to reset target peripherals when CPU is reset.
- Call Stack is not supported.

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.