Technical Notes

Renesas 78K/78K0R/RL78 Family In-Circuit Emulation

This document is intended to be used together with the CPU reference manual provided by the silicon vendor. This document assumes knowledge of the CPU functionality and the terminology and concepts defined and explained in the CPU reference manual. Basic knowledge of winIDEA is also necessary. This document deals with specifics and advanced details and it is not meant as a basic or introductory text.

Contents

Contents **1**

1 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

3 Setting CPU options **6**
3.1 CPU Options 6
3.2 Debugging Options 7
3.3 Advanced Options 8

4 Flash **10**

5 Memory Access **10**

6 Access Breakpoints **11**

7 Getting Started **13**

8 Trace **13**

9 Coverage **13**

10 Profiler **14**

11 Emulation notes **14**

1 Introduction

Renesas 78K, 78K0R and RL78 development systems are built around an FPGA, which contains the microcontroller core and the necessary debug logic controlling the core and the surrounding peripherals. Peripherals are emulated by a dedicated Renesas device, which contains all peripherals of the devices being emulated. The development system is configured specific for every target microcontroller being emulated.

Debug Features

- Unlimited execution breakpoints
- Access breakpoints
- Overlay emulation memory in the program and data flash area
- Real-time access
- Fail-safe exceptions
- Trace
- Code Coverage
- Profiler

1.1 Differences from a standard environment

As soon as the 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.

Besides 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. This allows the POD operating standalone without the target being connected.

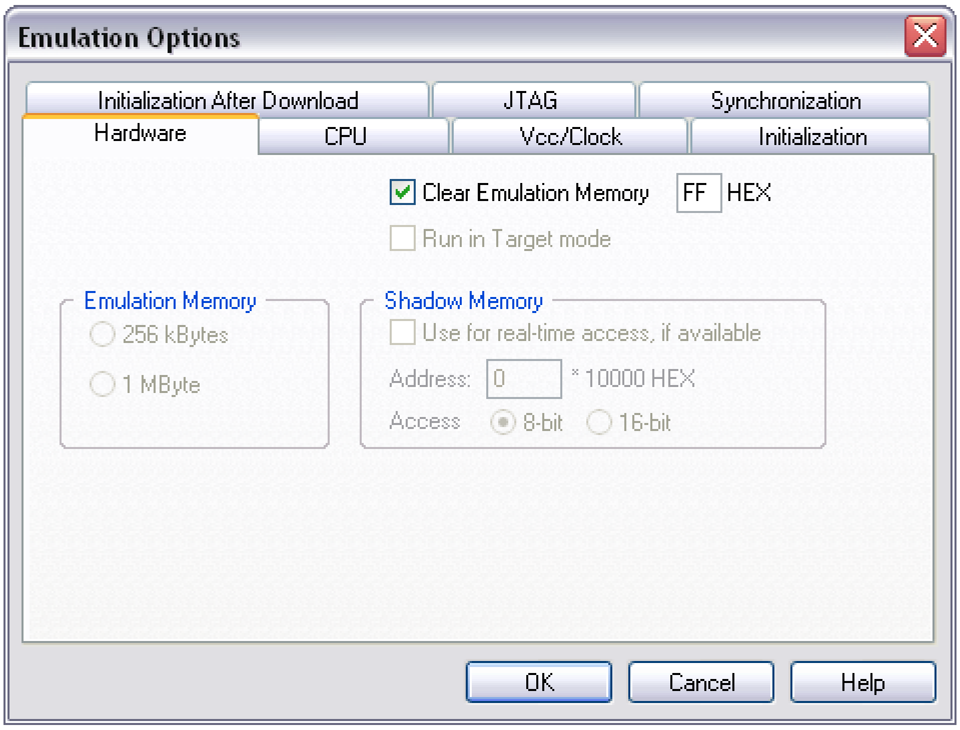
1.2 Common Guidelines

Here are some general guidelines that should be followed:

- Use external (target) Vcc/GND if possible (to prevent GND bouncing).
- Make an additional GND connection from the development system (iC3000HS) to the target to protect the emulator from being damaged due to different target and emulator ground potentials, at the time when the POD is plugged to the target.

2 Emulation Options

2.1 Hardware Options


Emulation Options

Tabs: Initialization After Download | JTAG | Synchronization | Hardware | CPU | Vcc/Clock | Initialization

[✓] Clear Emulation Memory FF HEX

[ ] Run in Target mode

Emulation Memory
- ( ) 256 kBytes
- ( ) 1 MByte

Shadow Memory
- [ ] Use for real-time access, if available
- Address: 0 * 10000 HEX
- Access (●) 8-bit ( ) 16-bit

In-Circuit Emulator Options dialog, Hardware page

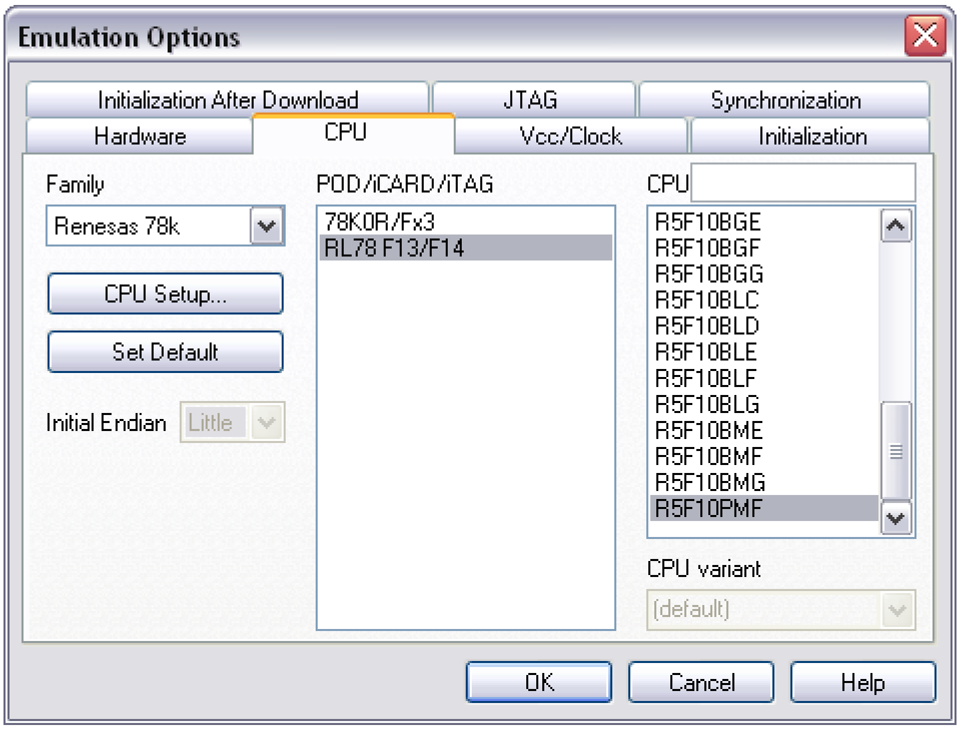
Clear Emulation Memory

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

Clearing emulation memory takes certain amount of time. Therefore 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).


Emulation Options

Tabs: Initialization After Download, JTAG, Synchronization, Hardware, CPU, Vcc/Clock, Initialization

Family: Renesas 78k

Initial Endian: Little

POD/iCARD/iTAG:
- 78K0R/Fx3
- RL78 F13/F14 (selected)

CPU:
- R5F10BGE
- R5F10BGF
- R5F10BGG
- R5F10BLC
- R5F10BLD
- R5F10BLE
- R5F10BLF
- R5F10BLG
- R5F10BME
- R5F10BMF
- R5F10BMG
- R5F10PMF (selected)

CPU variant: (default)

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

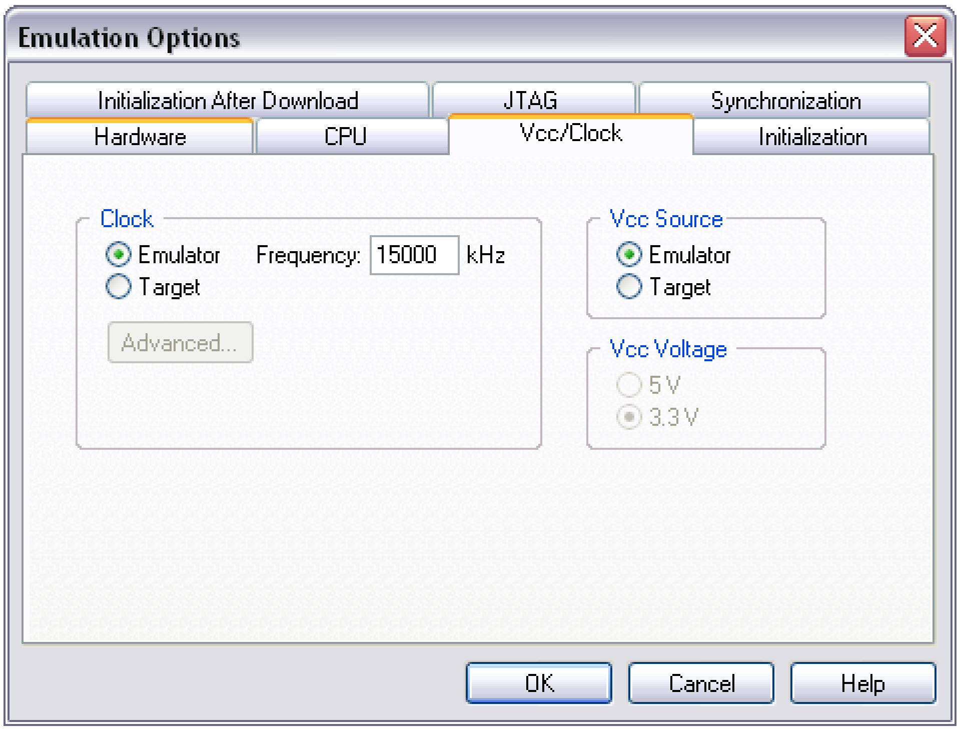
This button will set default options for currently selected CPU. These include:

- Vcc and clock source and frequency
- Advanced CPU specific options

Note: Default options are also set when the Family or POD/iCARD/iTAG selection is changed.

2.3 Power Source and Clock

The Vcc/Clock Setup page determines the emulated microcontroller power and clock source.



Clock: [✓ Emulator] Target — Frequency: 15000 kHz

Vcc Source: [✓ Emulator] Target

Vcc Voltage: 5 V [✓ 3.3 V]

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

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.

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.

Vcc Source

This setting defines whether the development system or the target system provides power supply for the CPU.

2.4 Initialization Sequence

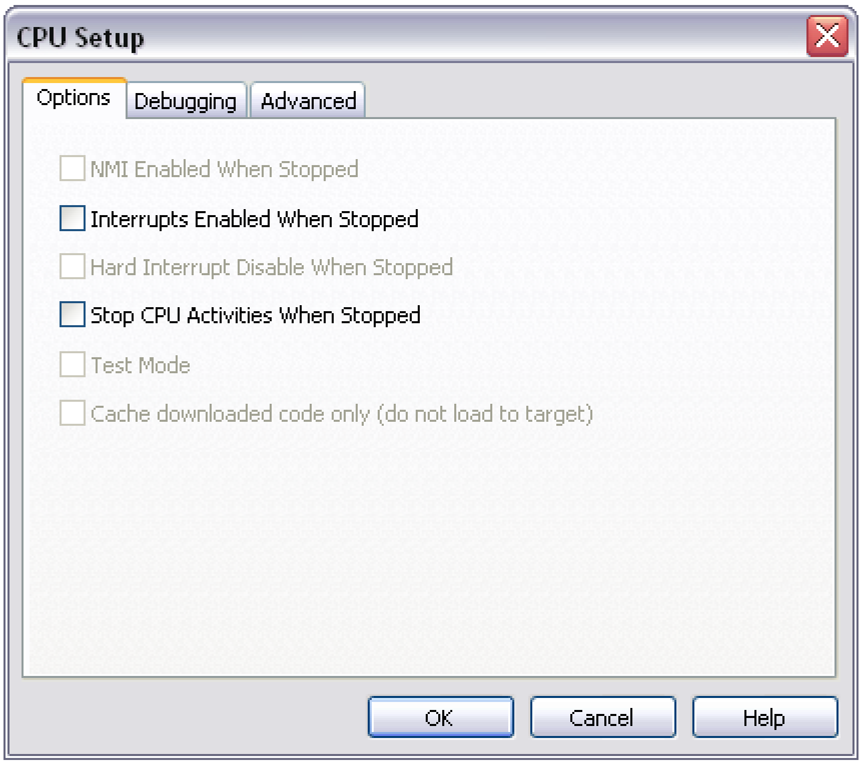
Usually, there is no need to use initialization sequence when debugging with an In-Circuit Emulator (ICE) a single chip application. Primarily, initialization sequence is used on On-Chip Debug systems to initialize the CPU after reset to be able to download the code to the target (CPU or CPU external) memory. With an ICE system, the initialization sequence may be required for instance to enable memory access to the CPU internal EEPROM or to some external target memory, which is not accessible by default after the CPU reset. The user can also disable CPU internal COP using initialization sequence if there is a need for that, etc.

Initialization sequence is executed immediately after the CPU reset and then the code is downloaded. Detailed information may be found in the Initialization Sequence help topic.

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.



☐ NMI Enabled When Stopped
☐ Interrupts Enabled When Stopped
☐ Hard Interrupt Disable When Stopped
☐ Stop CPU Activities When Stopped
☐ Test Mode
☐ Cache downloaded code only (do not load to target)

CPU Setup, Options page

Interrupts Enabled When Stopped

The emulator itself doesn't support servicing interrupts while the application is stopped (interrupts in background). Setting of this option impacts only the CPU behavior during the debug single step execution.

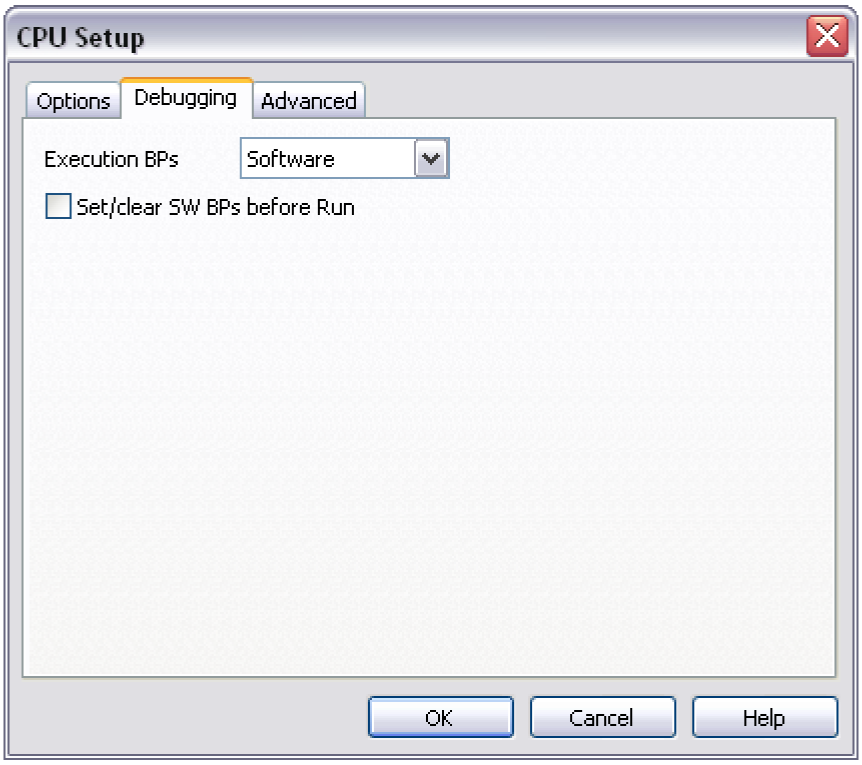
Default disabled option makes the emulator to mask the interrupts while the source step debug command is being executed, which yields more predictive behavior in case of the application using interrupts. Interrupts are disabled through the IE flag in the PSW register hidden from the user.

If this option is enabled, the emulator doesn't mask interrupts and they can occur while stepping through the application. If there is a periodic interrupt, it may happen that the user will keep re-entering the interrupt while stepping. In such applications, it's recommended to disable this option.

Stop CPU Activities When Stopped

When the option is checked, all internal peripheral modules, like timers and counters, are stopped when the user's program is stopped. Otherwise, timers and counters remain running while the program is stopped. Usually, when the option is checked, the emulation system behaves more predictable while stepping through the program. While being aware of the consequences, it is up to the user whether the option is checked or not.

3.2 Debugging Options


CPU Setup

Options | Debugging | Advanced

Execution BPs: Software

[ ] Set/clear SW BPs before Run

Debugging Options

Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited to sixteen. The advantage is that they function anywhere in the CPU space, which is not the case for software breakpoints, which normally cannot be used in the FLASH memory, non-writeable memory (ROM) or self-modifying code. If the option 'Use hardware breakpoints' is selected, only hardware breakpoints are used for execution breakpoints.

Note that the debugger uses execution breakpoints hidden from the user when executing the source step debug command. A single execution breakpoint is sufficient to implement the source step for almost all supported CPU families. Renesas 78K microcontroller is an exception to this rule. In worst case, the debugger can use all 16 available hardware breakpoints in order to carry out the source step and thus leaving non to the user. In contrary, when all available hardware breakpoints are used up for execution breakpoints, the debugger can fail to execute the source step. Software breakpoints should be used if the user finds this too annoying.

Software Breakpoints

Available hardware breakpoints often prove to be insufficient. Then the debugger can use unlimited software breakpoints to work around this limitation.

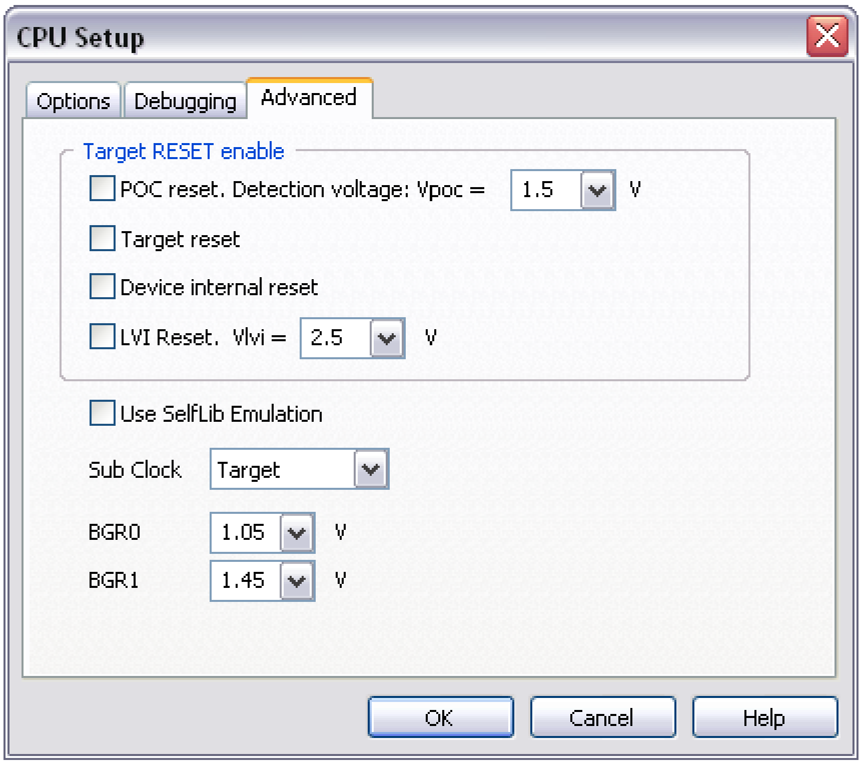
When a software breakpoint is being used, the program first attempts to modify the source code by placing a break instruction into the code. If setting software breakpoint fails, a hardware breakpoint is used instead.

Note: It is recommended to use unlimited software breakpoints since the 78K emulator provides RAM overlay memory for the complete code area.

Set/clear SW BPs before Run

When the option is checked, then a software breakpoint is not set/cleared immediately, but is just remembered. Only when the CPU is set to running are the breakpoints committed. This way several breakpoints can be changed but only one re-FLASH operation takes place. This is especially noticeable in testIDEA operation with many stubs and also during a regular debugging session when several breakpoints are set/cleared within the same flash erase block. This is also applicable for standard RAM type of memory.

3.3 Advanced Options



Target RESET enable

- [ ] POC reset. Detection voltage: Vpoc = 1.5 V
- [ ] Target reset
- [ ] Device internal reset
- [ ] LVI Reset. Vlvi = 2.5 V

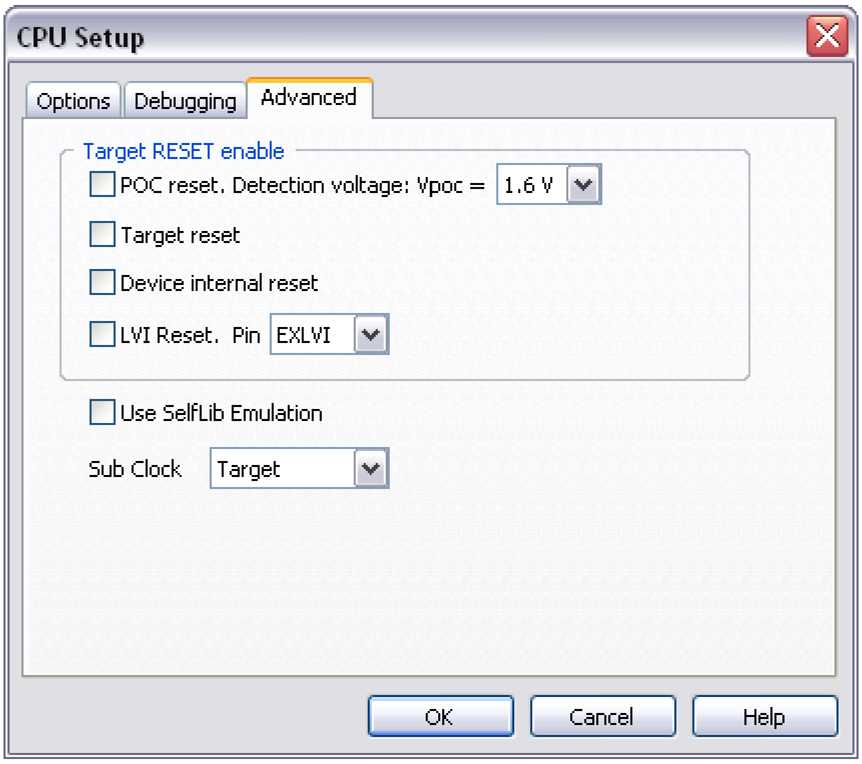
- [ ] Use SelfLib Emulation

Sub Clock: Target

BGR0: 1.05 V

BGR1: 1.45 V

Renesas RL78 Advanced Options


CPU Setup

Options | Debugging | Advanced

Target RESET enable

- [ ] POC reset. Detection voltage: Vpoc = 1.6 V
- [ ] Target reset
- [ ] Device internal reset
- [ ] LVI Reset. Pin EXLVI

- [ ] Use SelfLib Emulation

Sub Clock: Target

Renesas 78K and 78K0R Advanced Options

Target RESET enable

The emulation system provides support for different microcontroller reset sources out of which each can be individually enabled.

For a POC reset, a POC reset comparator is available with adjustable voltage between 1.6 and 2.7V.

Use SelfLib Emulation

Program and data flash are being replaced with RAM based memory. Only the debugger can directly access this memory while application sees it as a ROM type of memory, which cannot be modified through a simple memory write access.

When the application needs to modify program or data flash area, it can do this only through Renesas SelfLib libraries, which are linked to the application. Check the 'Use SelfLib Emulation.' option when application uses Renesas SelfLib libraries.

Sub Clock

Subsystem clock is one of the available microcontroller systems clocks. Either a fixed 32.768 kHz crystal being connected to the XT1 and the XT2 or an external (Target) subsystem clock supplied from the target can be selected.

BGR0, BGR1

These two settings affect the microcontroller A/D operation and are available on RL78 development system only.

This setting should keep default values, 1.05V for the BGR0 and 1.45V for the BGR1. The BGR0 setting concerns temperature compensation of the A/D reference voltage and the BGR1 is an internal reference voltage output, which can be A/D converted through the ADS (Analog input channel specification) register configuration (ADS = 0x81)

4 Flash

RL78 ActiveGT POD replaces **code flash** and **data flash** memory with an overlay memory (RAM), which allows fast debug download and memory modification directly through the debug memory window. Consequentially the user must be aware that after powering on the emulation system, flash areas are uninitialized respectively they don't hold empty flash values (e.g. 0xFF) nor any data stored from earlier debug sessions. This difference to the original microcontroller must not be forgotten. To better mimic the original microcontroller, the user can address this issue by making use of certain standard debug features. For example, code/data flash areas can be preset either using winIDEA initialization sequence or by downloading a dedicated binary file into the flash area presetting the whole flash to a known state.

Data flash memory is not accessible after reset while code flash memory is. In order to view data flash content in the debug memory window or for example to download a binary file into the data flash, user must enable read access to the data flash by configuring DFLCTL register accordingly first. This can be either performed via the target application but note that the data flash memory remains inaccessible until according application code has been executed. More convenient for debugging is to enable data flash read access through **winIDEA initialization sequence** (write 0x01 to the DFLCTL register), which is executed after every debug reset or debug download per default. This makes data flash accessible at any moment.

5 Memory Access

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

Real-Time Memory Access

The emulator features dual-port memory for the complete memory space except for the SFR area, which is covered by the shadow memory. This allows all memory to be read and write in real-time without intrusion on the application execution, except for the SFRs, which can only be read in real-time.

Monitor Access

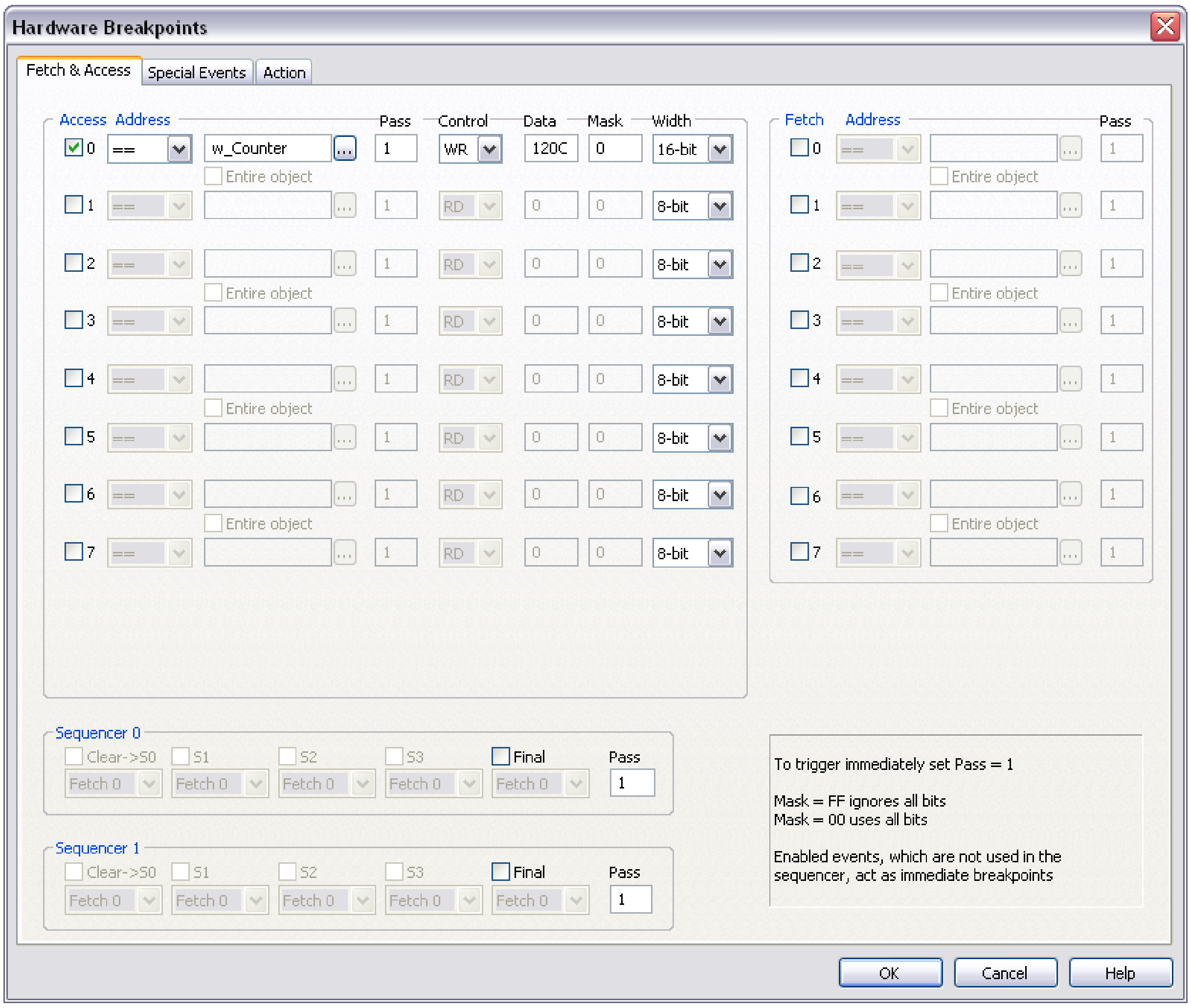
When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

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

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

6 Access Breakpoints

Access breakpoints dialog is opened via Debug/Hardware Breakpoints… menu.



Access Address

Access		Address	Pass	Control	Data	Mask	Width
✓ 0	==	w_Counter … ☐ Entire object	1	WR	120C	0	16-bit
☐ 1	==	…	1	RD	0	0	8-bit
☐ 2	==	… ☐ Entire object	1	RD	0	0	8-bit
☐ 3	==	…	1	RD	0	0	8-bit
☐ 4	==	… ☐ Entire object	1	RD	0	0	8-bit
☐ 5	==	…	1	RD	0	0	8-bit
☐ 6	==	… ☐ Entire object	1	RD	0	0	8-bit
☐ 7	==	…	1	RD	0	0	8-bit

Fetch		Address	Pass
☐ 0	==	… ☐ Entire object	1
☐ 1	==	…	1
☐ 2	==	… ☐ Entire object	1
☐ 3	==	…	1
☐ 4	==	… ☐ Entire object	1
☐ 5	==	…	1
☐ 6	==	… ☐ Entire object	1
☐ 7	==	…	1

Sequencer 0

☐ Clear->S0	☐ S1	☐ S2	☐ S3	☐ Final	Pass
Fetch 0	Fetch 0	Fetch 0	Fetch 0	Fetch 0	1

Sequencer 1

☐ Clear->S0	☐ S1	☐ S2	☐ S3	☐ Final	Pass
Fetch 0	Fetch 0	Fetch 0	Fetch 0	Fetch 0	1

To trigger immediately set Pass = 1

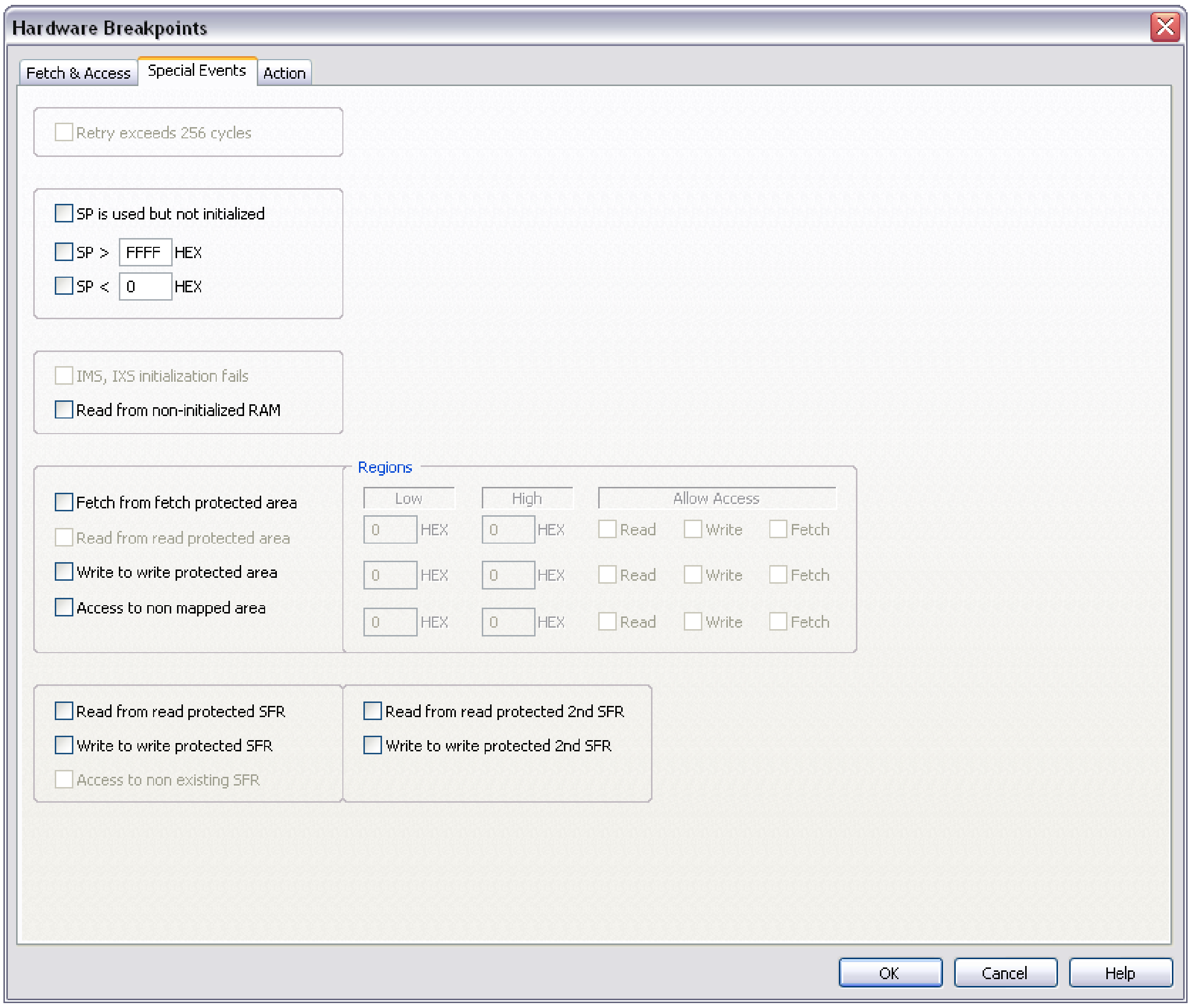
Mask = FF ignores all bits
Mask = 00 uses all bits

Enabled events, which are not used in the sequencer, act as immediate breakpoints

78K Hardware Breakpoints dialog

There are a number of access and fetch conditions which can be defined. For each condition the address can be specified (whether it is an exact address, inside or outside a specified range, higher or lower than the specified address or an entire object, specified with the '…' button), a counter of occurrences of a specified event can be defined, the access type (read, write, read/write) and the data and mask of the event.

Two sequencers are available to configure more complex access breakpoints.


Hardware Breakpoints

Fetch & Access | Special Events | Action

- [ ] Retry exceeds 256 cycles
- [ ] SP is used but not initialized
- [ ] SP > FFFF HEX
- [ ] SP < 0 HEX
- [ ] IMS, IXS initialization fails
- [ ] Read from non-initialized RAM
- [ ] Fetch from fetch protected area
- [ ] Read from read protected area
- [ ] Write to write protected area
- [ ] Access to non mapped area

Regions

Low		High		Allow Access		
0	HEX	0	HEX	[ ] Read	[ ] Write	[ ] Fetch
0	HEX	0	HEX	[ ] Read	[ ] Write	[ ] Fetch
0	HEX	0	HEX	[ ] Read	[ ] Write	[ ] Fetch

- [ ] Read from read protected SFR
- [ ] Write to write protected SFR
- [ ] Access to non existing SFR
- [ ] Read from read protected 2nd SFR
- [ ] Write to write protected 2nd SFR

78K / 78K0R / RL78 Special Events dialog

Additionally, there are a number of special events which can also be specified within access breakpoints. Each special event selected breaks the application too.

Few useful settings:

- Check the ‘SP is used but not initialized’ option to break the application, when SP is not initialized before use.
- Specify valid SP range (SP>, SP<) to break the application when the stack use exceeds allocated memory space.
- Check the ‘Read from non-initialized RAM’ option to break the application when it reads the non-initialized variable.

7 Getting Started

1) Connect the system.
2) Power up the emulator and then power up the target.
3) Execute debug reset.
4) The CPU should stop on the address where the reset vector points to.
5) Open memory window at RAM location and check whether it is possible to modify its content.
6) If above steps passed successfully, the debugger is operational.

8 Trace

The development system features iSYSTEM proprietary bus trace implementation, which provides time accurate time information for each recorded bus cycle and a complex trigger and filter configuration.

Trace Features

- 512K frames buffer depth on 78K ActivePRO POD
- 64M frames buffer depth on 78K0R and RL78 ActiveGT POD
- Unlimited time stamp reach
- Supported Trace Configurations:
  - Record Everything
  - Plain Trigger/Qualifier
  - Watchdog Trigger
  - Duration Tracker
  - Q between B and C events
  - Pre/Post Qualifier
  - Data Change

For more information on these trace functionalities and use refer to winIDEA Contents Help describing Bus Trace in details.

9 Coverage

Refer to winIDEA Contents Help, Coverage Concepts section for Coverage theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Coverage user interface and use.

10 Profiler

Refer to winIDEA Contents Help, Profiler Concepts section for Profiler theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Profiler user interface and use.

11 Emulation notes

- 78K0R series internal EEPROM area can be accessed only through Renesas library. Even for the read access, the user cannot see the EEPROM content in the standard memory window but must read it through its own application (to which Renesas library is linked), which features according “EEPROM memory read” function amongst other functions managing the emulated EEPROM.