
Technical Notes

NEC V850 Family In-Circuit Emulation

Contents

Contents.....	1
1 Introduction	2
1.1 Differences from a standard environment	2
1.2 Common Guidelines.....	2
1.3 Port Replacement Information	3
2 Emulation Options.....	4
2.1 Hardware Options	4
2.2 CPU Configuration	5
2.3 Power Source and Clock	6
2.4 Initialization Sequence	7
3 Setting CPU options	9
3.1 CPU Options	9
3.2 Advanced Options	10
4 Memory Access.....	10
5 Access Breakpoints	11
6 Emulation Notes	12
6.1 Error 228: Execution stopped. Writing to internal ROM is prohibited.	12
6.2 SFR Relocation - BPC Register	12
6.3 Other Notes	12
7 Getting Started.....	12
8 Trace.....	13
9 Execution Coverage.....	13
10 Execution Profiler.....	15
11 On-Chip Profiler.....	17

1 Introduction

The NEC V850 pod is built by a combination of a special NEC CPU bondout chip and a target device. The bondout provides for debug interface, trace, execution coverage, partial port replacement and real-time watch memory. The target device provides a system clock and specific port replacement.

Note: on the ActivePRO a powerful FPGA replaces and exceeds the bondout's trace, coverage and real-time watch capabilities.

Debug Features

- Unlimited number of breakpoints
- 2 access breakpoints
- 1MB emulation memory
- Trace
- Execution coverage
- Profiler on ActivePRO
- Target voltage 5V (3.3V on ActivePRO), lower voltages with future devices
- Real-time access

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.

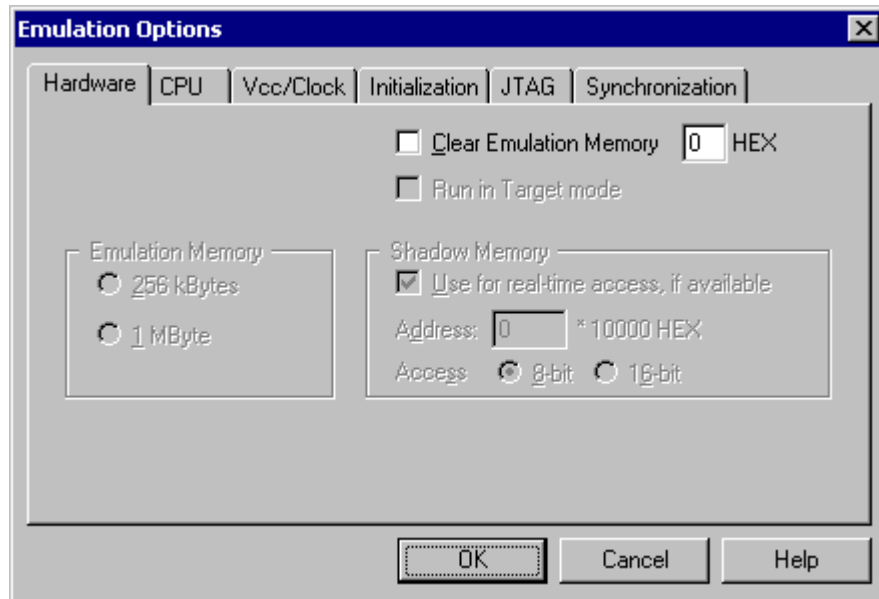
- Use external (target) Vcc/GND if possible (to prevent GND bouncing),
- Make an additional GND connection from POD to the target if the Emulator behaves strangely,
- Use the reset output line on the POD to reset the target whenever Emulator resets the CPU,
- Make sure the appropriate CPU is used on 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

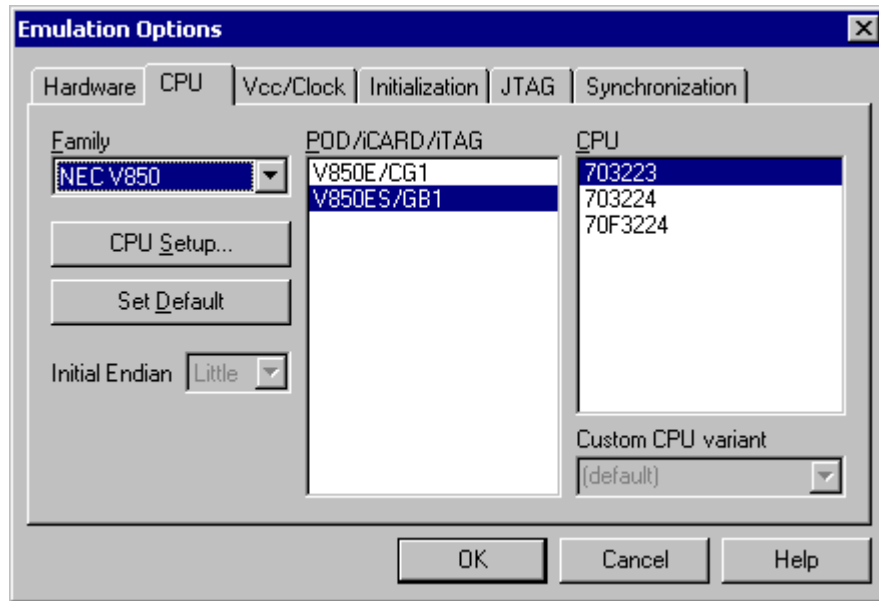
Clear Emulation Memory

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

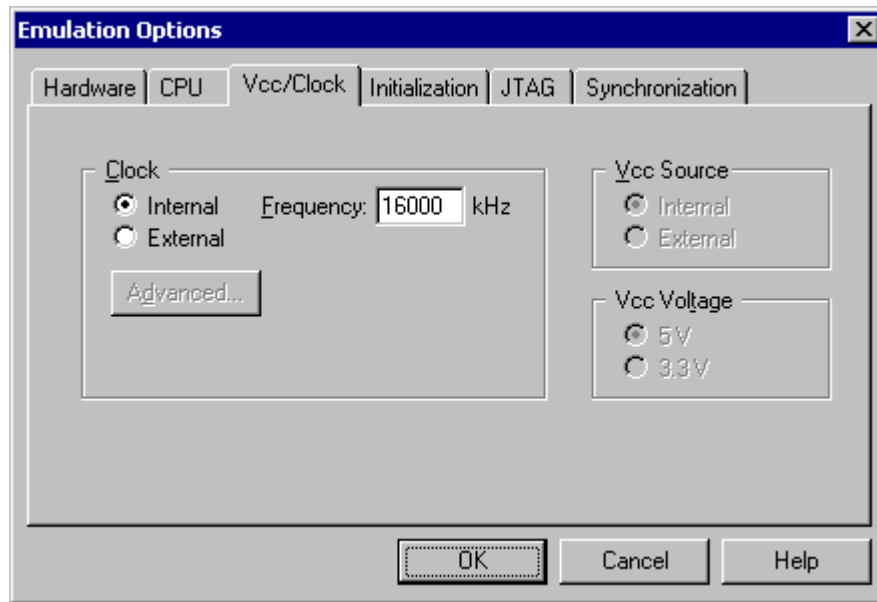
This button will set default options for currently selected CPU. These include:

- 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.

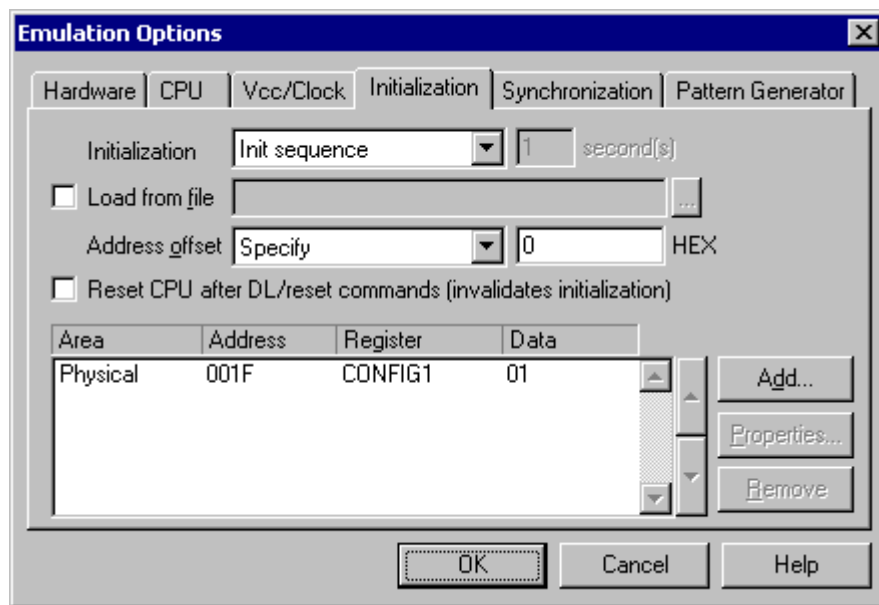
Note: The clock frequency is the frequency of the signal on the CPU's clock input pin. Any internal manipulation of it (division or multiplication) depends entirely on the emulated CPU.

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:

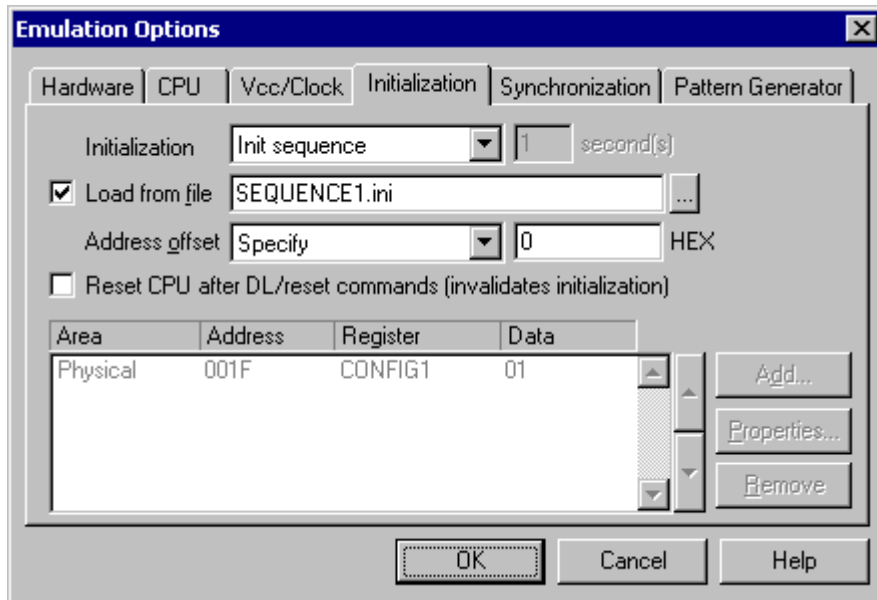
1. Set up the initialization sequence by adding necessary register writes directly in the Initialization page within winIDEA.



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.

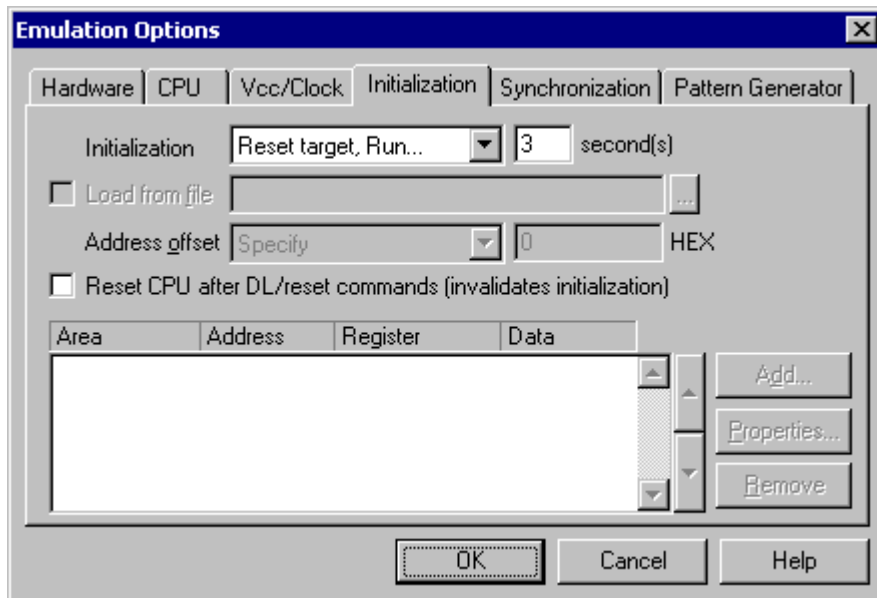
Excerpt from the sample SEQUENCE1.ini file:

```
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.

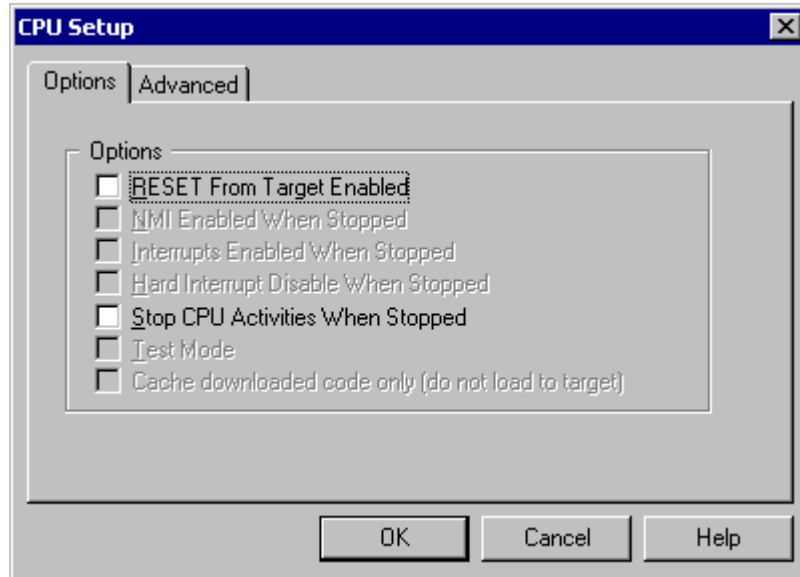
There is also a third method, which can be used too but it's not highly recommended for the start up. The user can initialize the CPU by executing part of the code in the target ROM for X seconds by using 'Reset and run for X sec' option.



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

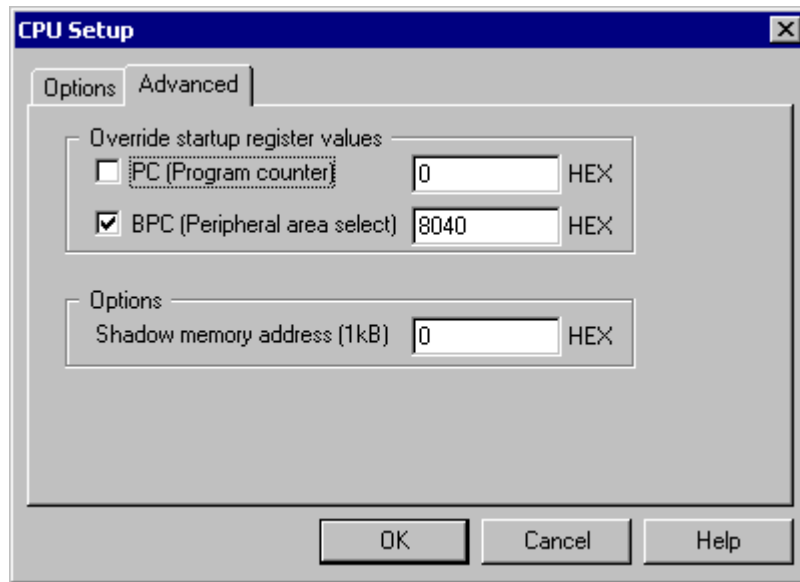
RESET From Target Enabled

When checked, the target's RESET line can reset the CPU while the CPU is running.

Stop CPU Activities When Stopped

When the option is checked, all internal periphery, like timers and counters, is 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 regularly 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 Advanced Options



NEC V850E Advanced ActiveEmulation Options

Override Startup Register Values

Allows you to override the default values of PC (Program Counter) and BPC (Peripheral Area Select).

Shadow Memory Address

Defines the address of the shadow memory.

4 Memory Access

NEC V850 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

V850 Active PODs use bondout real-time resources (shadow memory technique), which limit real-time access to 1KB data memory area. User must specify a base address of the variables of interest in the CPU Setup/Advanced dialog. For example, variable `byTestByte` is located at `0xFFFFE011`. Set the Shadow address base entry to `0xFFFFE000`. Thus the available range is `0xFFFFE000 - 0xFFFFE03FF`.

The Shadow memory address base must and can only be aligned to a 1KB (0x400) boundary, because the lower 10 bits of the bondout register are not implemented.

V850 Active PRO PODs feature real-time access based on shadow memory. 8KBytes of shadow RAM is divided into 128 blocks, which are dynamically allocated and capable to cover 128KBytes of memory.

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

Monitor Access

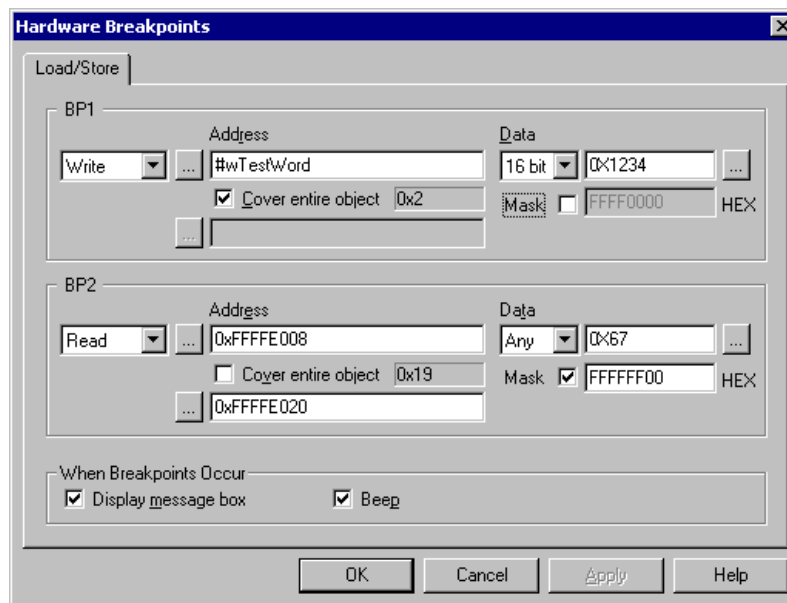
When monitor access to the CPU's memory is requested, the emulator stops the CPU and instructs it to read the requested number of bytes.

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.

5 Access Breakpoints

The load/store breakpoints dialog is open by clicking Hardware breakpoints button in the Breakpoints dialog.



V850E Hardware Breakpoints dialog

In the left-hand listbox an access type is selected. It can be Disabled, Read, Write, Read or Write.

Address field accepts directly a number or a name of a selected variable. By pressing on the [...] button a variable can be selected from the Source browser. By unchecking the *Cover entire object*, also an ending address can be specified.

The listbox in the Data section specifies an access-width, it can be 8-bit, 16-bit, 32-bit or Any, for any kind of access width. In the field, next to the listbox, a data value can be entered. Optionally, a data mask can be entered in the field below. If a mask bit is, the corresponding data value bit is disregarded in a comparison.

6 Emulation Notes

6.1 *Error 228: Execution stopped. Writing to internal ROM is prohibited.*

NEC bondout breaks on invalid memory access, e.g. write to the internal ROM. A message box is displayed when it happens. The warning indicates a program error - bug. Usually it results from a store instruction with an incorrectly set address register. The cause is either a coding error (null/uninitialized pointer or interrupt handler not saving/restoring all modified registers), or, less likely, a compiler optimization error. Use Trace in Continuous mode for analysis.

6.2 *SFR Relocation - BPC Register*

Due to a bug of the NEC bondout chip, the Programmable peripheral I/O SFRs need to be located below 0x001FFFFFF. Under Emulation options/CPU Setup/Advanced dialog there is an option to preset the BPC accordingly. By default winIDEA suggests a value of 0x8040, per NEC recommendation. This puts, for example, the CAN module registers just above 1MB limit, at 0x00100nnn.

6.3 *Other Notes*

For correct A/D converter operation, the AVDD and AVSS reference voltages must be provided by the target. In standalone POD operation (not connected to target) the two pins are pulled low to VSS..

The clock to peripherals cannot be stopped during STOP mode with the exception of V850ES/Fx2 devices. For example, timers continue to run when the CPU is halted. For this reason, a user application must refresh watchdog, where present.

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 reset location 0x0.
- 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

V850 development system features a powerful trace named Active Trace, which is implemented externally to the CPU.

Refer to a separate document describing Active Trace features and use.

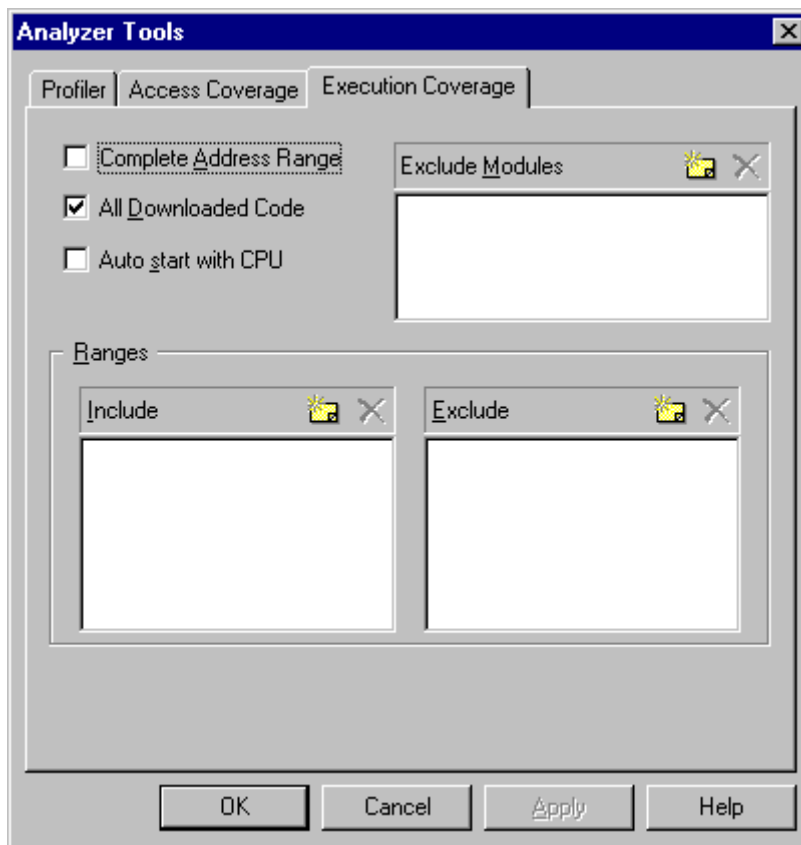
9 Execution Coverage

Execution coverage records all addresses being executed, which allows the user to detect the code or memory areas not executed. It can be used to detect a so called “dead code”, the code that was never executed. Such code represents undesired overhead when assigning code memory resources.

Execution coverage covers complete CPU address space. It can run infinite time, which means in practice that the application can run for days and then the execution coverage results can be analyzed.

- First, make sure that ‘Active’ Trace is selected in the Hardware/Analyzer Setup dialog.
- Next, select ‘Execution Coverage’ window from the View menu and configure Execution Coverage settings. Normally, ‘All Downloaded Code’ option has to be checked only. The debugger extracts all the necessary information like addresses belonging to each C/C++ function from the debug info, which is included in the download file and configure hardware accordingly.

Refer to software user’s guide for more details on configuring Execution Coverage and its use.



Execution Coverage is configured. Reset the application, start Execution Coverage and then run the application. When it's assumed that the complete application code was executed, stop the Execution Coverage and inspect the results.

Red boxes on the left in the source window and disassembly window depict not executed code. Execution Coverage window shows which code was executed/not executed for selected modules.

Address	Data	Disassembly	Registers
		Type Pointers	PC 00000474
		c='A';	PSW
0000045A	200E4100	MOVEA 0041,R0,R1	R0 00000000
0000045E	430F0400	ST.B R1,0004[R3]	R1 00000042
		pC=&c;	R2 C7ABC74
00000462	230E0400	MOVEA 0004,R3,R1	R3 FFFEBF4
00000466	630F0100	ST.W R1,0000[R3]	R4 00028000
		++c;	R5 00000003
0000046A	830F0500	LD.BU 0004[R3],R1	R6 00000004
0000046E	410A	ADD 01,R1	R7 FFFEBD4
00000470	430F0400	ST.B R1,0004[R3]	R8 00000006
		++*pC;	R9 00000003
00000474	230F0100	LD.W 0000[R3],R1	R10 6ED8CDE1
00000478	232F0100	LD.W 0000[R3],R5	R11 67BCE0BE
0000047C	852F0100	LD.BU 0000[R5],R5	R12 71B0F9CF
00000480	412A	ADD 01,R5	R13 D552CA31
00000482	412F0000	ST.B R5,0000[R1]	R14 23D5C9BF
		ppC=&pC;	R15 EB708D9D
00000486	0308	MOV R3,R1	R16 8F1404BF
		++(**ppC);	R17 A3910B1E
00000488	212F0100	LD.W 0000[R1],R5	R18 DF2D500F
0000048C	210F0100	LD.W 0000[R1],R1	R19 5EAD4F3B
00000490	810F0100	LD.BU 0000[R1],R1	R20 C22F513B
00000494	410A	ADD 01,R1	R21 A96ACB24
00000496	450F0000	ST.B R1,0000[R5]	R22 FB29EF75
		++iCounter;	R23 F7917C7B
0000049A	200F0100	LD.W -2000[R0],R1	R24 9AAF2541
0000049E	410A	ADD 01,R1	R25 C069BDFD
000004A0	600F0100	ST.W R1,-2000[R0]	R26 D6BB0703
		}	R27 ADE265DF
000004A4	481A	ADD 08,R3	R28 EFB5D4FA
000004A6	7F00	JMP [R31]	R29 E8F2FFB2

Source & Disassembly Window: Execution Coverage results

Modules	Lines Graph	Lines	Sizes Graph	Sizes
Modules		433/781 (55%)		F00/1FB0 (47%)
+ crt0.s		43/45 (96%)		AC/B4 (96%)
+ CPUtest.c		3/20 (15%)		30/158 (14%)
+ main.c		2/34 (6%)		18/148 (7%)
+ long main()		2/20 (10%)		18/78 (20%)
+ {		1/1 (100%)		14/14 (100%)
+ {		1/1 (100%)		4/4 (100%)
+ test.c		10/177 (6%)		150/B84 (11%)
+ void Type_Simple()		1/31 (3%)		24/1D0 (8%)
+ d=c;		1/1 (100%)		24/5C (39%)
+ 00000454 - 00000477				
+ void Address_TestScopes()		1/19 (5%)		10/114 (6%)
+ ++X;		1/1 (100%)		10/10 (100%)
+ float Func4(float, unsigned)		8/8 (100%)		11C/11C (100%)
+ {		1/1 (100%)		24/24 (100%)
+ float fRet=(float)0.0;		1/1 (100%)		8/8 (100%)
+ for (i=0;i<5;++i)		1/1 (100%)		14/14 (100%)
+ for (i=0;i<5;++i)		1/1 (100%)		10/10 (100%)
+ *(pC+i)=0xA+i;		1/1 (100%)		1C/1C (100%)
+ fRet+=f+(float)*(pC+i)+(f		1/1 (100%)		88/88 (100%)
+ return fRet;		1/1 (100%)		8/8 (100%)
+ }		1/1 (100%)		20/20 (100%)
+ fp-bit.c		148/219 (68%)		484/674 (70%)
+ dp-bit.c		191/250 (76%)		778/9A4 (77%)
+ libgcc2.c		36/36 (100%)		C0/C0 (100%)

Execution Coverage Window results

10 Execution Profiler

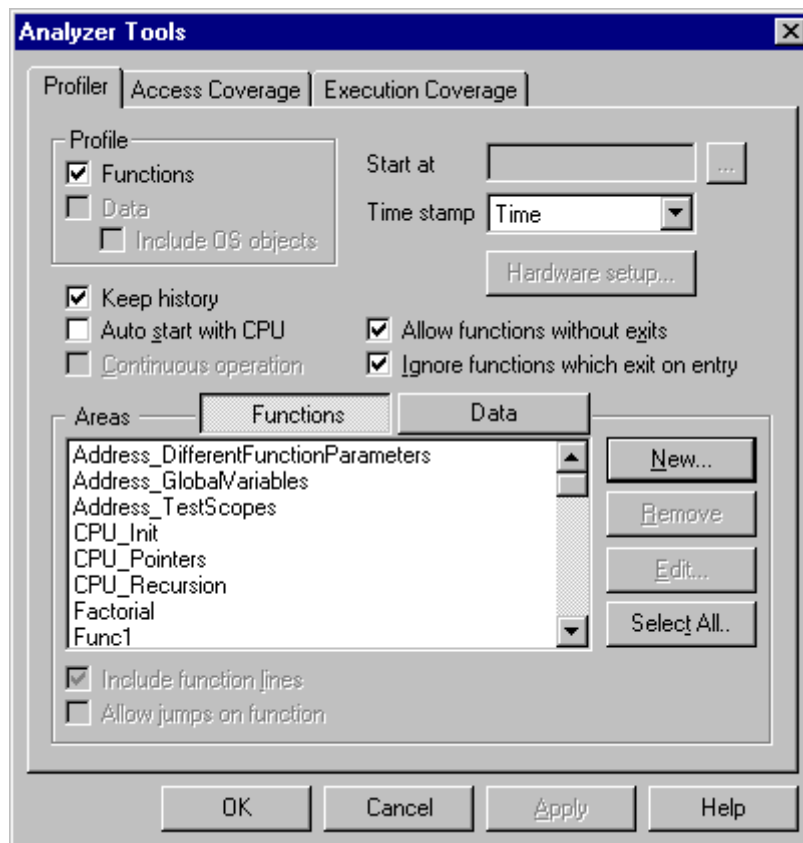
Profiler records executed function entry and exit points and then run time-analysis over the collected information. As a result it gives details on how much time (minimum, maximum, average) has the CPU spent in the particular function. Available information allows the user to optimize those parts of code, which are most time consuming or time critical.

The debug download file must contain accurate debug information when using Profiler to analyze C/C++ application. Normally Profiler extracts all the necessary information from the debug information and becomes useless if configured for wrong function entry and exit points.

- First, make sure that 'Active' Trace is selected in the Hardware/Analyzer Setup dialog.
- Next, select 'Profiler' window from the View menu and configure Profiler settings. Select 'Functions' option in the 'Profile' field.
- Make sure that 'Keep history' option is checked if Code Execution view is going to be used during results analysis.
- Finally, profiled C/C++ functions are selected by pressing 'New...' button. It's recommended that 'All C Functions' is selected for the beginning. Additionally, 'Include lines' can be checked which will yield in time analysis of each source line belonging to the function.

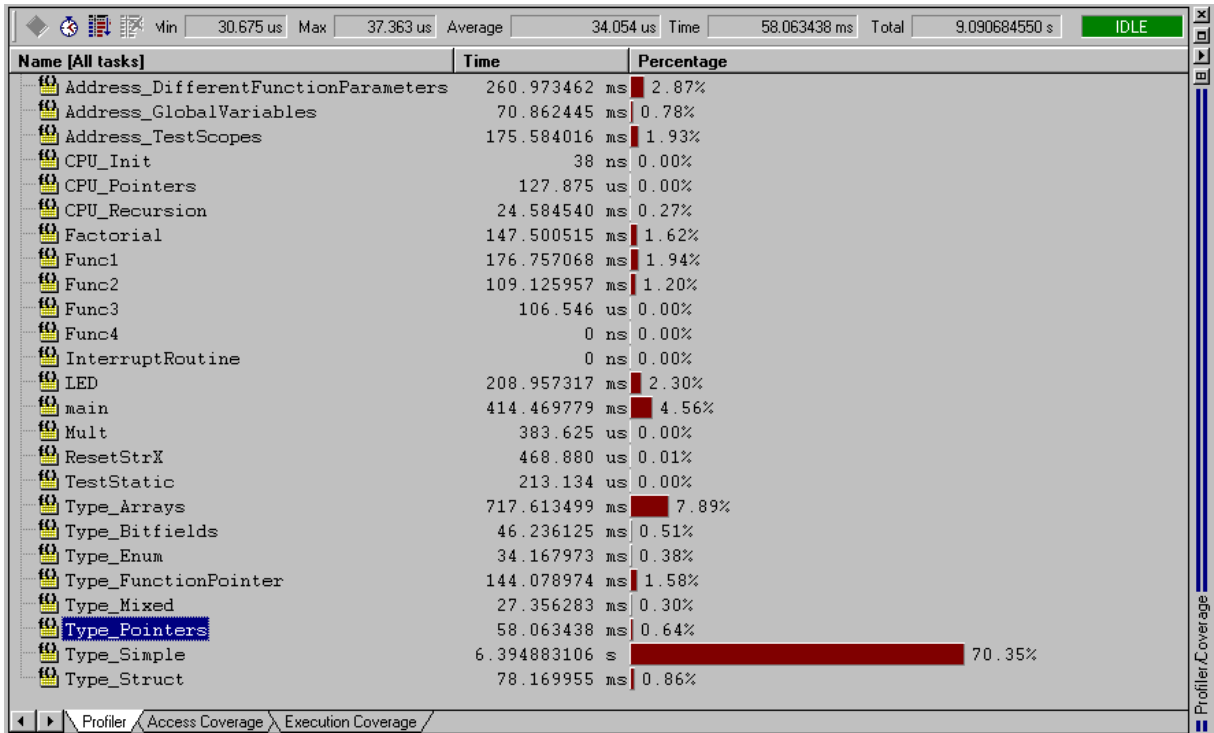
The debugger extracts all the necessary information from the debug info, which is included in the download file and configure the hardware accordingly.

Refer to software user's guide for more details on configuring Profiler and its use.

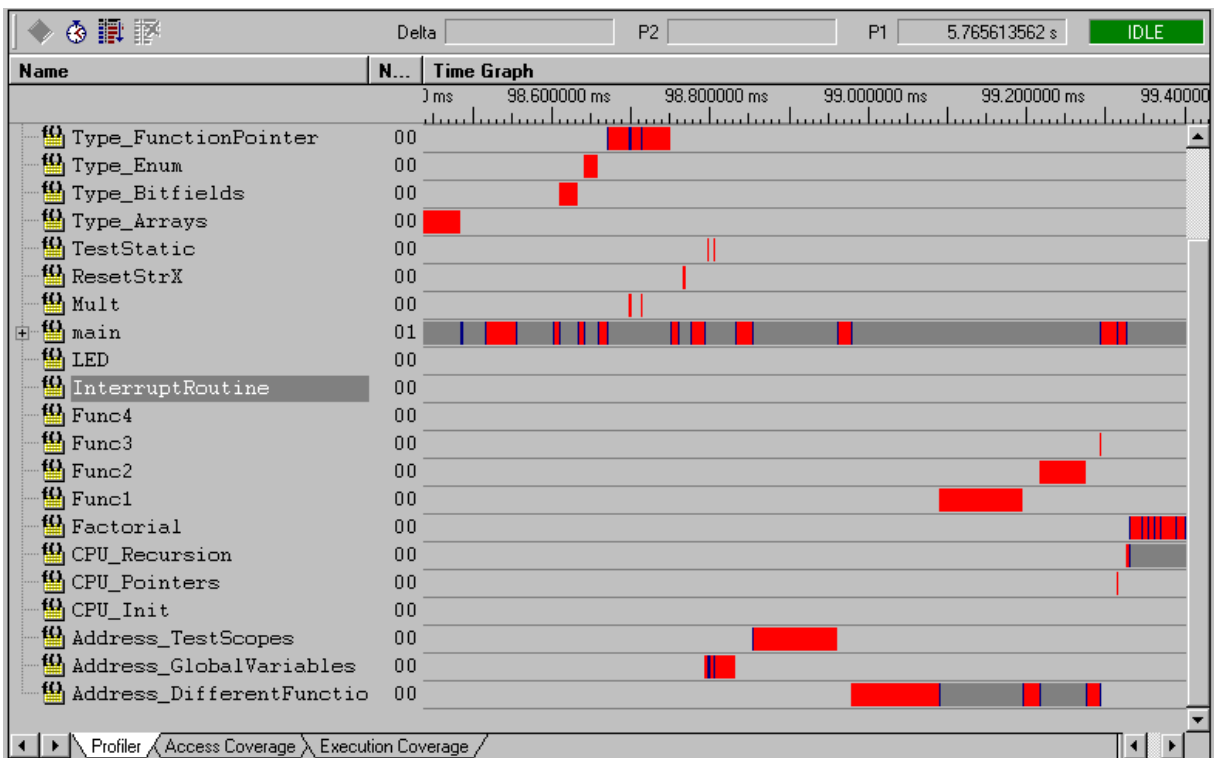


Profiler configuration settings

Profiler is configured. Reset the application, start Profiler and then run the application. The Profiler will stoop collecting information on a user demand or after the trace buffer becomes full. Then the recorded information is analyzed and profiler results displayed.



Profiler results – Code Statistics view

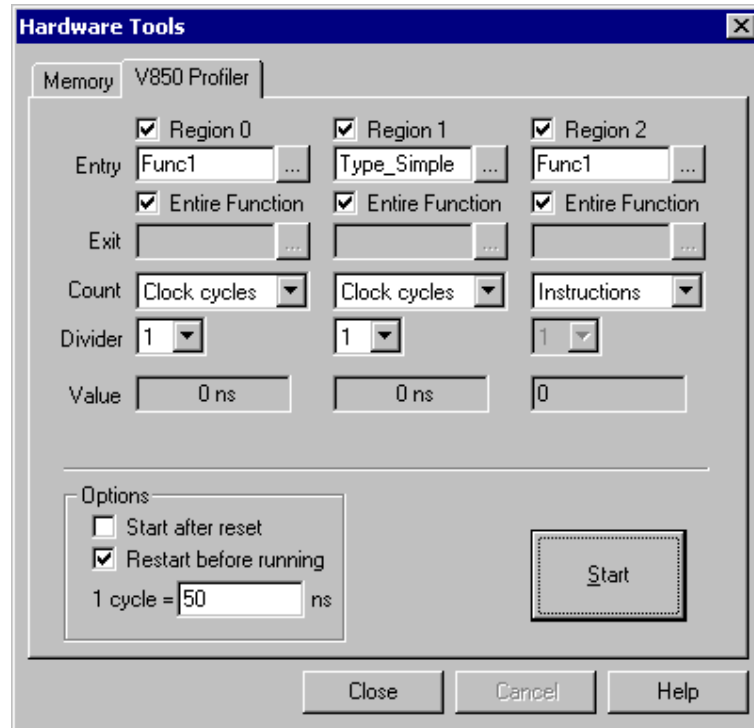


Profiler results – Code Statistics view

11 On-Chip Profiler

Note: This functionality is available on V850 Active PODs only since it's a bondout functionality.

The V850 On-Chip Profiler is invoked by the Hardware/Tools menu. Since this is a special NEC bondout feature, it can not be shown in the regular Trace/Coverage/Profiler window.



NEC V850 Profiler

Supported are three code regions. For every region, an Entry and Exit addresses can be specified in hex numbers (prefixed with '0x') or selected from the Symbol Browser [...]. The profiler is actually a counter, and in the Count entry it can be selected to count either CPU Clock cycles or a number of Instructions executed. When counting CPU cycles a Divider can be set at f/1, f/2, f/4, f/8, f/16 or f/32 division ratio.

Entry

Defines the address of region entry

Exit

Defines the address of region exit

Entire Function

If checked, the region Entry is set to the specified function entry point and the region Exit is set to function exit point.

Time

Shows the time spent in the region.

Count

Shows the number of entries in the region.

Divider

If Timer is set to System clock, the configured value is used as divider.

Value

Shows the actual results. Note that results are available only after the profiler is stopped.

Start after reset

If checked, the profiler is started automatically after the CPU is released from reset.

Restart before running

If checked, the profiler is restarted prior to any advancement of the execution point (run, step,...)

1 cycle=

Defines the duration of one system clock cycle. If set to zero, the number of recorded system clocks is displayed.

Several options can be specified to enhance the tool usability. The Start after reset will reset the target, configure the profiler and then run the target CPU. The Restart before running will reconfigure the profiler every time the CPU is instructed to Run or Single-step. winIDEA automatically inserts a clock period in the 1 cycle option field, calculated from a frequency defined in the Emulation Options/Vcc/Clock dialog. The number of clock cycles is therefore converted to time values. To display the actual number of cycles, simply enter 0 in that field.

A limitation of the on-chip profiler is that Entry and Exit addresses must be at least 3 instructions apart.

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.