



ICDPPCNEXUS
MPC55xx / MPC56xx In-Circuit Debugger

Quick Start Guide

Document Version History

Version	Date	Notes
1.0	21 Sep 2009	Initial version

CONTENTS

1 Introduction	4
1.1 P&E Compatible Hardware	4
2 Getting Started	5
2.1 Connecting to your Target.....	5
2.2 Reset Script	6
2.3 Loading Data and Debug Information	7
2.4 CPU and Memory Windows.....	8
3 Debugging.....	10
3.1 GOTIL command	10
3.1 Stepping through C instructions	11
3.3 Setting and Reaching Breakpoints	12
3.4 Using Code Window Popup Debug Evaluation Hints	13
3.5 Using the Variables Window	15
3.6 Modifying a Variable.....	16
3.7 Using the Register Interpreter	17
3.8 Adding Register Field Descriptions to the Variables Window	20

1 Introduction

This document is a step-by-step guide to using the P&E ICDPPCNEXUS in-circuit debugger software, which is compatible with Freescale MPC55xx / MPC56xx processors. This guide covers the most commonly used features of the debugger: loading binary & debug information, accessing CPU registers & memory, stepping code, setting breakpoints, and monitoring variables.

1.1 P&E Compatible Hardware

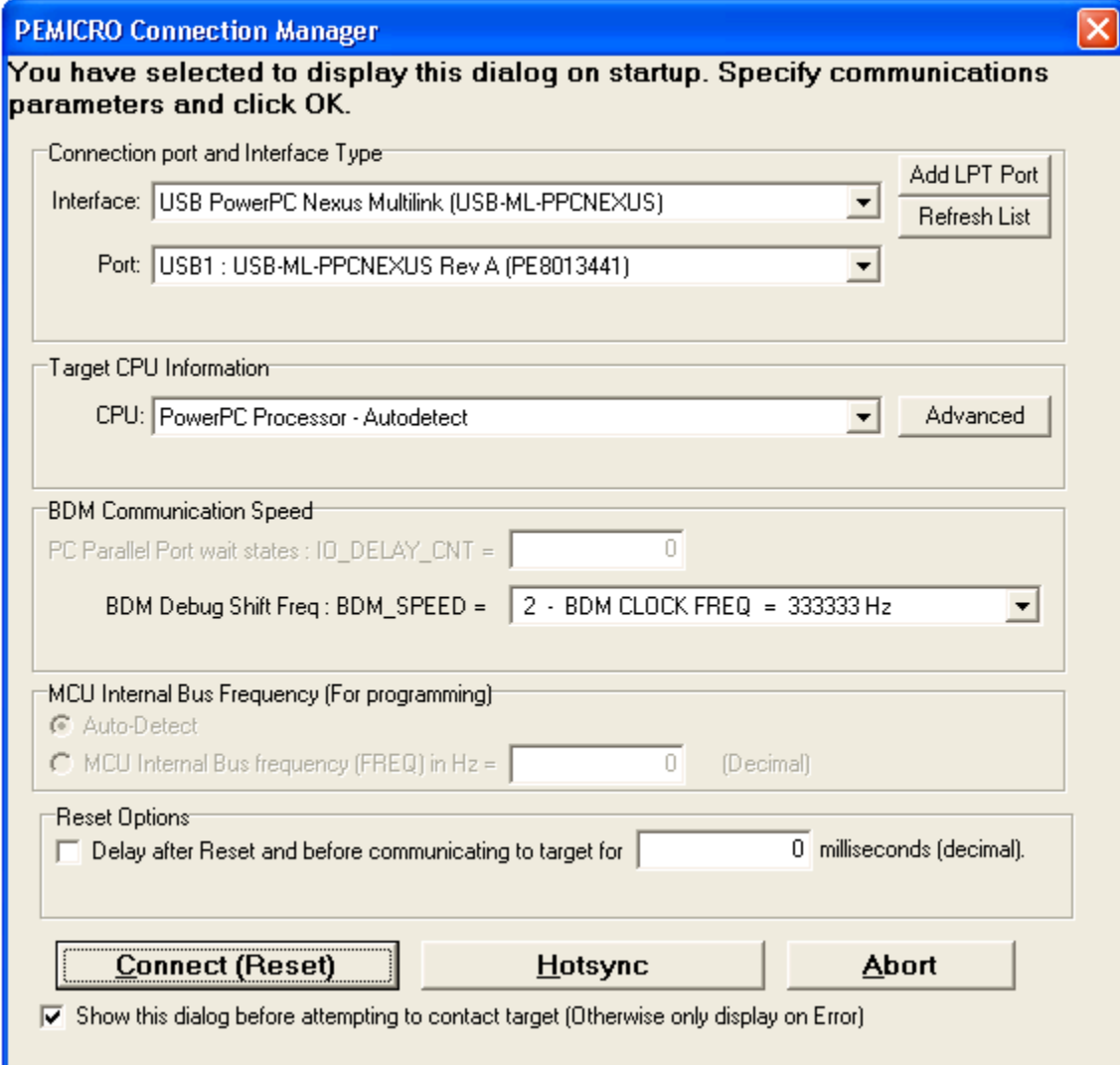
The following lists the P&E hardware compatible with the ICDPPCNEXUS debugger software.

P&E Part Number	Interface to host PC
CABPPCNEXUS	Parallel (LPT) port
USB-ML-PPCNEXUS	USB 2.0 (Backwards compatible with USB 1.1 ports)
Cyclone MAX	Serial (RS232) port USB 1.1 (Upwards compatible with USB 2.0 ports) Ethernet

2 Getting Started

2.1 Connecting to your Target

Upon starting the debugger, the connection assistant dialog appears:



The image shows a screenshot of the "PEMICRO Connection Manager" dialog box. The title bar is blue with the text "PEMICRO Connection Manager" and a close button. The main area has a light beige background and contains several sections:

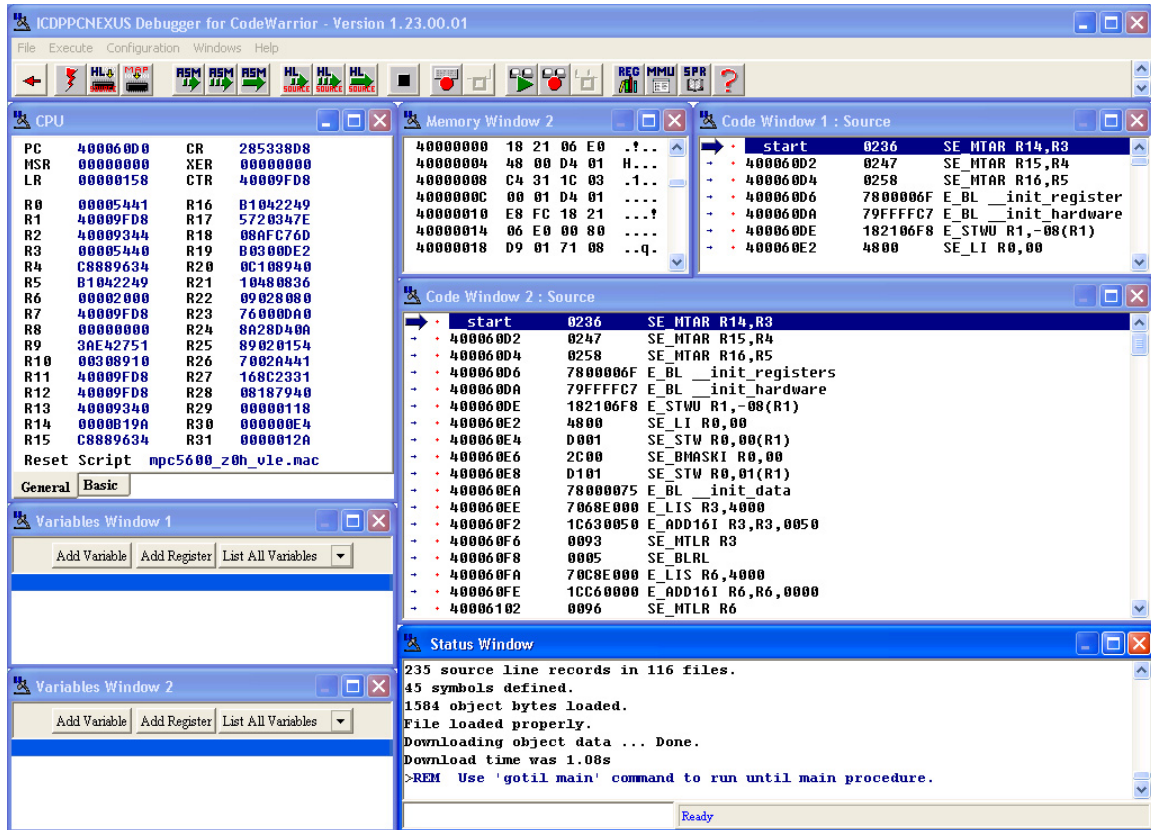
- Connection port and Interface Type:** A section with two dropdown menus. The "Interface:" dropdown is set to "USB PowerPC Nexus Multilink (USB-ML-PPCNEXUS)". The "Port:" dropdown is set to "USB1 : USB-ML-PPCNEXUS Rev A (PE8013441)". To the right of these dropdowns are two buttons: "Add LPT Port" and "Refresh List".
- Target CPU Information:** A section with a "CPU:" dropdown menu set to "PowerPC Processor - Autodetect" and an "Advanced" button to its right.
- BDM Communication Speed:** A section with two input fields. The first is "PC Parallel Port wait states : IO_DELAY_CNT =" with a value of "0". The second is "BDM Debug Shift Freq : BDM_SPEED =" with a dropdown menu set to "2 - BDM CLOCK FREQ = 333333 Hz".
- MCU Internal Bus Frequency (For programming):** A section with two radio buttons. The first is "Auto-Detect" (selected). The second is "MCU Internal Bus frequency (FREQ) in Hz =" with a value of "0" and "(Decimal)" to its right.
- Reset Options:** A section with a checkbox "Delay after Reset and before communicating to target for" followed by a value of "0" and "milliseconds (decimal)".

At the bottom of the dialog, there are three buttons: "Connect (Reset)" (highlighted with a dashed border), "Hotsync", and "Abort". Below these buttons is a checked checkbox "Show this dialog before attempting to contact target (Otherwise only display on Error)".

- Use the “Interface” and “Port” drop-down menus to choose the P&E hardware interface connected between the PC and your target board.
- The “Target CPU” setting can safely be left at the “Autodetect” setting for most users. If you experience problems connecting, you can try specifying the exact Freescale device that you are connecting to.
- A BDM_SPEED parameter between 2 to 4 can typically be used. Processors running at slower clock speeds will require higher values.

Click the Connect button, and ICDPPCNEXUS will attempt to contact the processor. Using the default debugger settings, ICDPPCNEXUS will establish communications and reset the processor.

After establishing communications, the main debugger screen will appear, and a debugger reset script macro should automatically execute and complete.



2.2 Reset Script

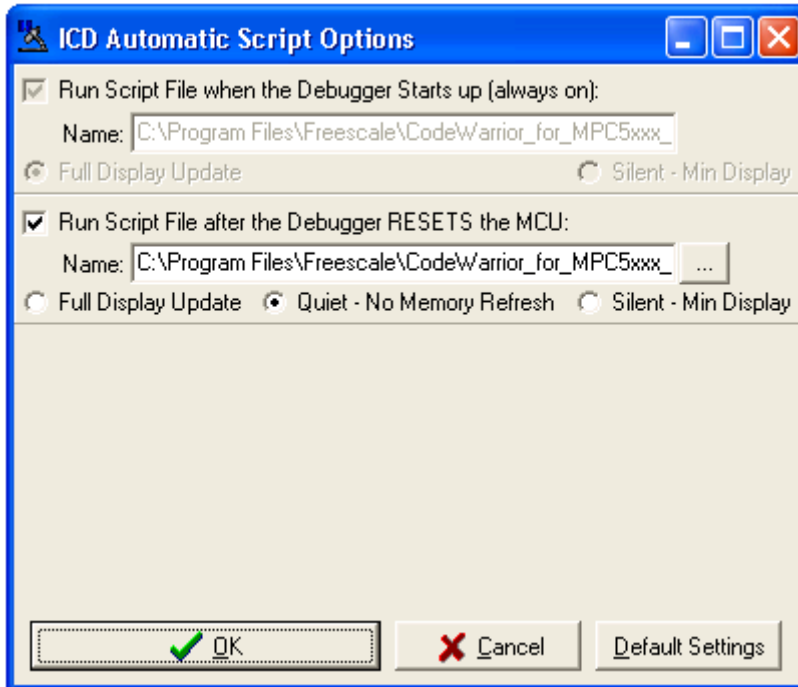
This section explains the initialization that the debugger, using a reset script macro file, performs on the processor. The user can view and modify all of the macro file's initialization tasks.

The processor Boot Assist Module (BAM) would normally initialize the memory of the processor. However, when running the target application from the debugger, the BAM functionality is disabled. To account for this, the debugger must run a script file on reset. The script initializes the memory of the processor similar to the way in which the BAM would initialize the processor.

If ICDPPCNEXUS is launched from the Freescale CodeWarrior IDE, the correct reset script file is automatically selected.

If ICDPPCNEXUS is launched stand-alone, the reset script file may need to be configured. Several reset script macros are included with the ICDPPCNEXUS debugger and have a .mac extension. For detailed information, you can view each macro file using a simple text editor such as Notepad. The macro contents will contain useful comments, such as which devices are supported by that particular macro.

To configure the debugger reset script macro, select the debugger Configuration menu, Automated Script Options dialog, shown here:



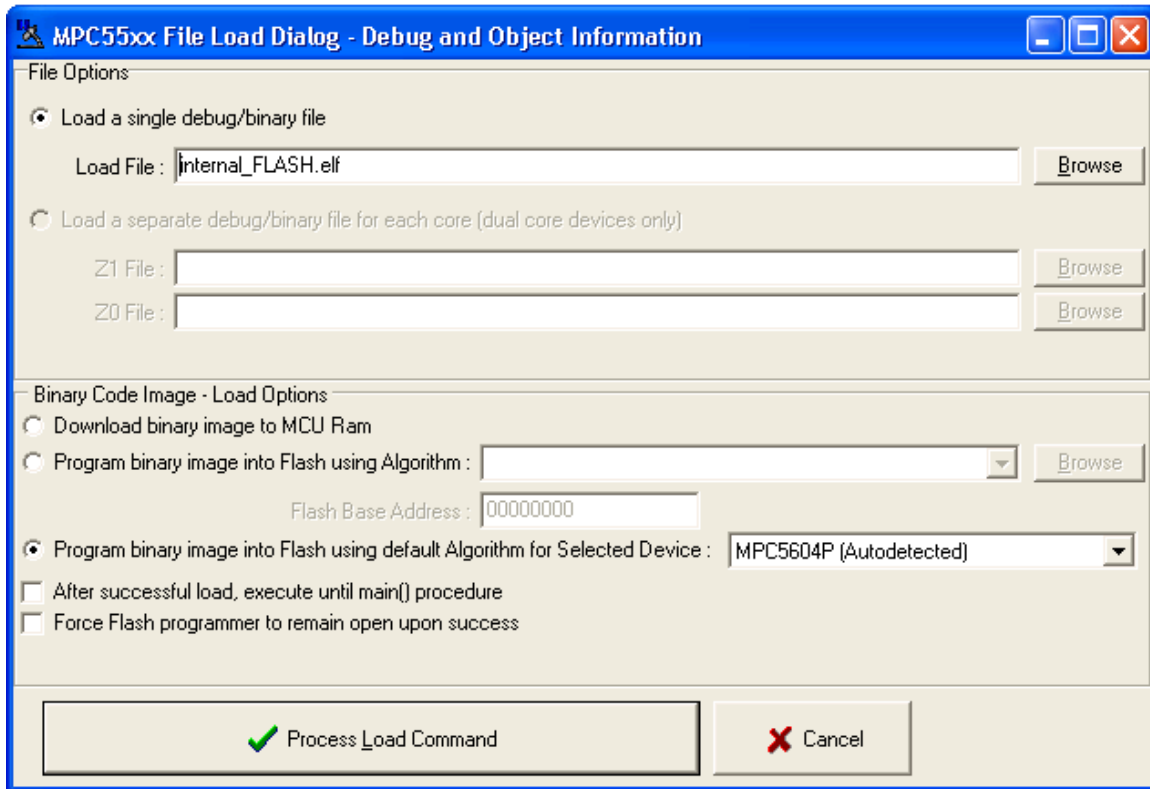
2.3 Loading Data and Debug Information

If ICDPPCNEXUS is launched from the Freescale CodeWarrior IDE, your code will automatically be downloaded to the processor.

- RAM projects are loaded into the processor's internal SRAM.
- FLASH projects will invoke the CPROGPPCNEXUS Flash programming software to burn the code into the processor's internal FLASH.

The debug information is also automatically loaded from CodeWarrior, which will allow you to debug using your high level source code and variables.

If ICDPPCNEXUS is launched stand-alone, you will need to manually download the code and debug information. Launch the Load Dialog by clicking on the High Level Load button on the debugger tool bar:



This dialog allows you to specify the binary/debug file and whether to load into RAM or FLASH. Once you are satisfied with your settings, press the “Process Load Command” button to begin the download process. This step will also load the debug information.

2.4 CPU and Memory Windows

The CPU Window displays all CPU core registers, including the Program Counter (PC) and all general purpose registers.

- To modify CPU register contents, double-click the register value. You will be prompted for a new value.

The Memory Window displays data at any given memory address. It can be used to view RAM contents, FLASH contents, and values of peripheral registers.

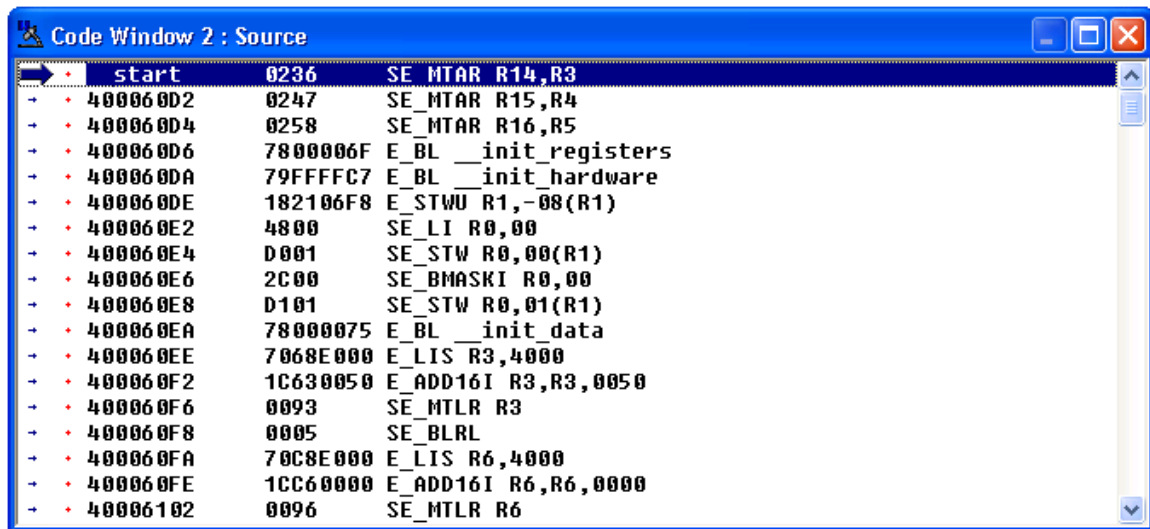
- To change the memory address, right-click inside the Memory Window and select “Set Base Address”. You will be prompted for a new address to begin displaying data.
- To change the contents in memory, double-click the value in memory that you would like to change. You will be prompted for a new value.

3 Debugging

This section outlines the different debugging capabilities available in the ICDPPCNEXUS debugger once the debug information has been loaded.

3.1 GOTIL command

At this point, your source window will show the assembly language startup code generated by the compiler:



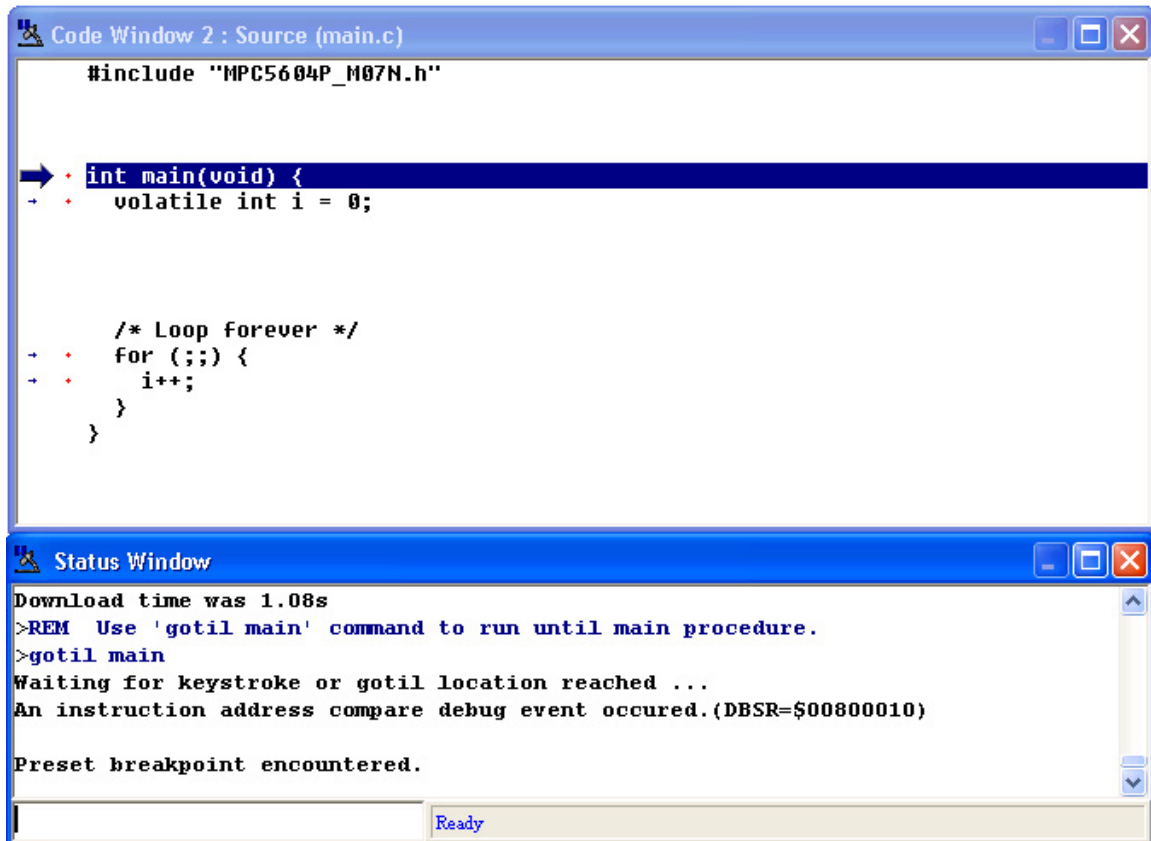
The screenshot shows a window titled "Code Window 2 : Source" with a list of assembly instructions. The instructions are as follows:

Address	Hex	Instruction
start	0236	SE_MTAR R14,R3
400060D2	0247	SE_MTAR R15,R4
400060D4	0258	SE_MTAR R16,R5
400060D6	7800006F	E_BL __init_registers
400060DA	79FFFFFF	E_BL __init_hardware
400060DE	182106F8	E_STWU R1,-08(R1)
400060E2	4800	SE_LI R0,00
400060E4	D001	SE_STW R0,00(R1)
400060E6	2C00	SE_BMASKI R0,00
400060E8	D101	SE_STW R0,01(R1)
400060EA	78000075	E_BL __init_data
400060EE	7068E000	E_LIS R3,4000
400060F2	1C630050	E_ADD16I R3,R3,0050
400060F6	0093	SE_MTLR R3
400060F8	0005	SE_BLR
400060FA	70C8E000	E_LIS R6,4000
400060FE	1CC60000	E_ADD16I R6,R6,0000
40006102	0096	SE_MTLR R6

If you do not need to debug this section and would like to run the processor until the beginning of your “main” function, you can use the “GOTIL” command.

- Type “GOTIL main” in the Status window to tell the debugger to run code until it reaches the “main” function of your code.

The “GOTIL” command works with any function in your code.



The screenshot shows a debugger interface with two windows. The top window, titled "Code Window 2 : Source (main.c)", displays the following C code:

```
#include "MPC5604P_M07N.h"

int main(void) {
    volatile int i = 0;

    /* Loop forever */
    for (;;) {
        i++;
    }
}
```

The bottom window, titled "Status Window", displays the following text:

```
Download time was 1.08s
>REM Use 'gotil main' command to run until main procedure.
>gotil main
Waiting for keystroke or gotil location reached ...
An instruction address compare debug event occured.(DBSR=$00800010)
Preset breakpoint encountered.
```

The status window has a "Ready" indicator at the bottom.

3.1 Stepping through C instructions

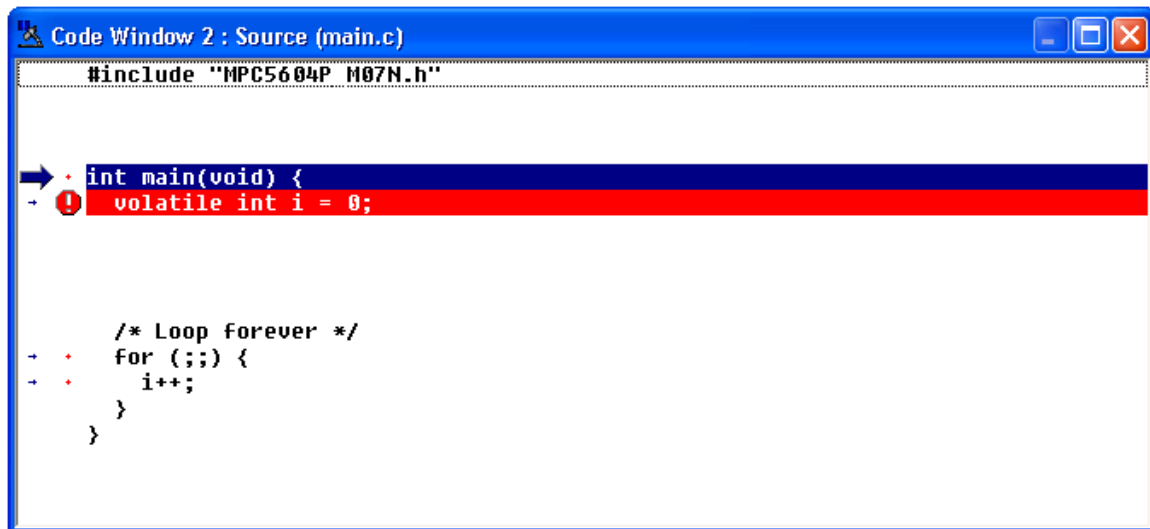
Step through the initialization code, or any source code, using the high-level language source step command. Use this feature by typing "HSTEP" in the Status window or by clicking the high-level step button on the debugger tool bar:



Each time the HSTEP command executes, the debugger will rapidly single step assembly instructions until it encounters the next source instruction, at which point target execution will cease. When the debugger reaches the next source instruction, all visible windows will be updated with data from the board. After reaching the main function, step through several C language instructions. Notice that some instructions will take longer to step through than others because each C instruction may consist of a greater or fewer number of underlying assembly instructions.

3.3 Setting and Reaching Breakpoints

In the source code window, there will be a small red dot and a small blue arrow next to each source instruction that has underlying object code. If a large blue arrow appears on a source line, this indicates that the program counter (PC) currently points to this instruction. If a large red stop sign appears on the source line, this indicates that a breakpoint exists on this line.



The screenshot shows a window titled "Code Window 2 : Source (main.c)". The code contains the following lines:

```
#include "MPC5604P_M07N.h"

int main(void) {
    volatile int i = 0;

    /* Loop forever */
    for (;;) {
        i++;
    }
}
```

In the code, the line `volatile int i = 0;` is highlighted in red and has a large red stop sign icon to its left, indicating a breakpoint. The line `int main(void) {` is highlighted in blue and has a small blue arrow icon to its left, indicating the current instruction being executed. The line `/* Loop forever */` is highlighted in yellow.

- Set a breakpoint at an instruction by double-clicking the tiny red dot.
- To remove a breakpoint, double-click the large red stop sign.

Execution will begin in real-time when you issue the HGO command or click the high-level language GO button on the debugger tool bar:



If the debugger encounters a breakpoint, execution will stop on this source line. If it does not encounter a breakpoint, target execution will continue until you press a key or use the stop button on the debugger tool bar:

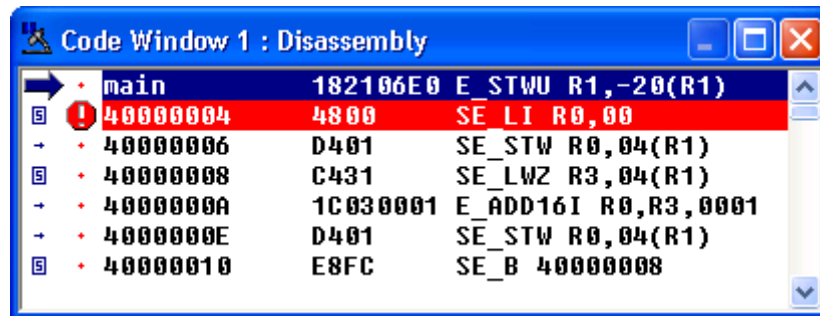


- By double clicking the small blue arrow, you will be issuing a GOTIL command to the address of this source line.

A GOTIL command will set a single breakpoint at the desired address, and the processor will begin executing code in real-time from the current program counter (PC). When the debugger encounters the GOTIL address, execution stops. If the debugger does not encounter this location, execution continues until you press a

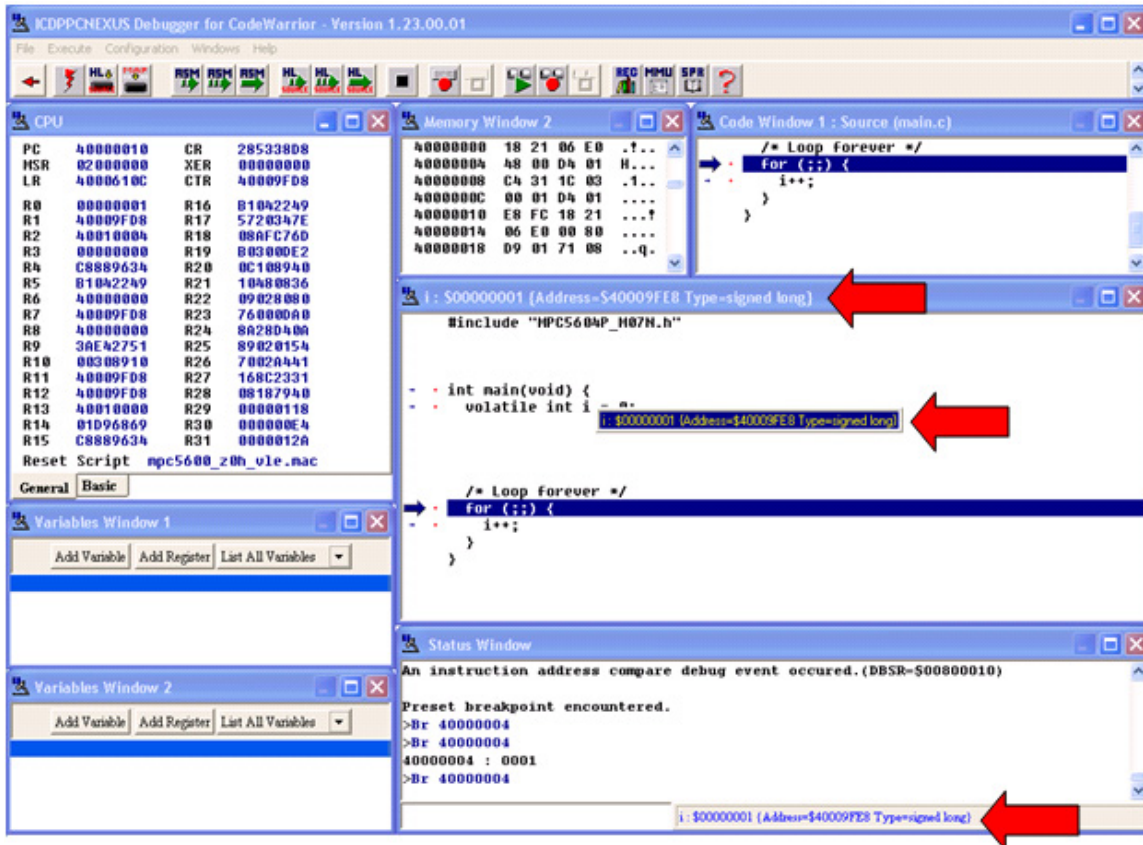
key or use the stop button on the debugger tool bar. Note that all user breakpoints are ignored when the GOTIL command is used.

You may also double-click the red and blue symbols in the disassembly window. The disassembly window may display an additional symbol, a small, blue "S" enclosed in a box. This indicates that that a source code instruction begins on this disassembly instruction.



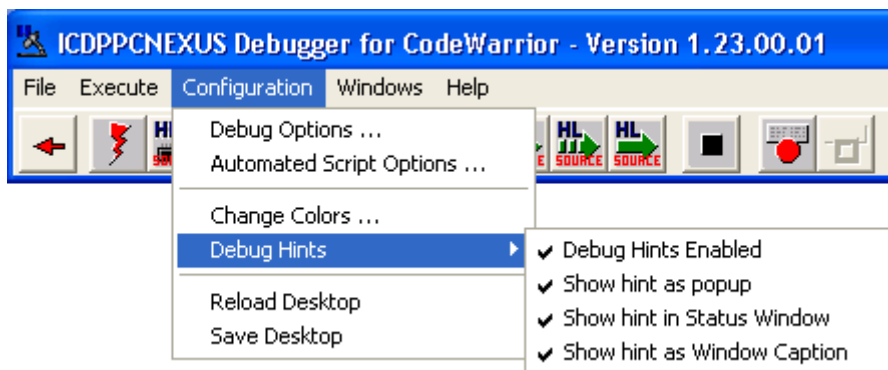
3.4 Using Code Window Popup Debug Evaluation Hints

When debugging source code, it is convenient to view the contents of a variable while viewing your source code. The in-circuit debugger has a feature, debug hints, which displays the value of a variable while the mouse cursor is held over the variable name. The hint may be displayed in any of three locations, as shown below.



The three locations for the debug hints are the code window title bar, the status window caption bar, and a popup hint that appears over the variable in source code. You can configure the hints to display in any combination.

- Set the locations of debug hints in the configuration menu of the debugger



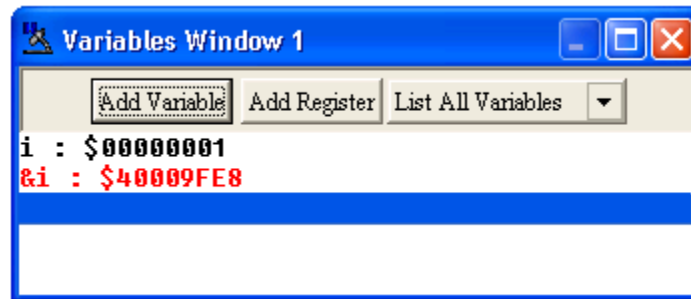
The information in the popup hint box is similar to the information displayed in the variables window.

```
i: $00000001 {Address=$40009FE8 Type=signed long}
```

The information includes the variable name (i), value (\$1), and type (signed long).

3.5 Using the Variables Window

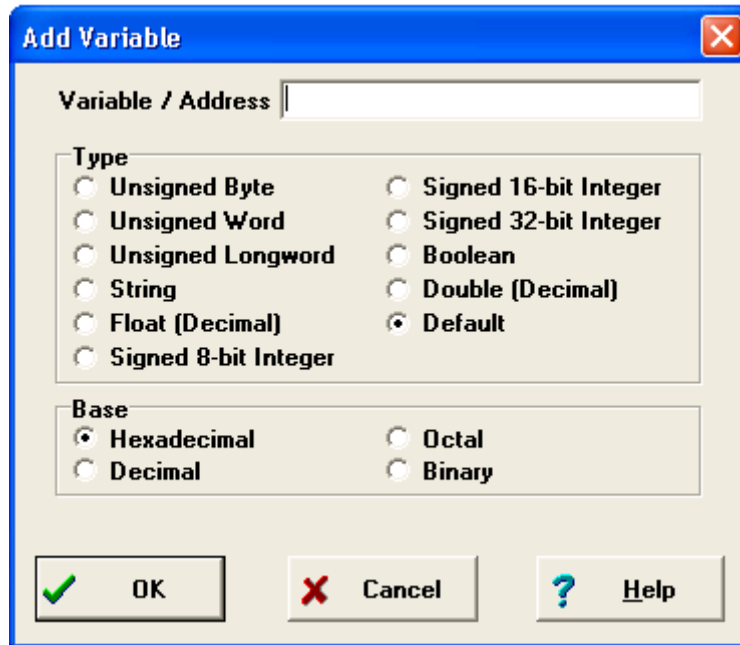
The variables window displays the current value of application variables. The following window shows a display of variables from the example application.



Variables that are pointer or reference types are displayed in red. Normal variables are displayed in black.

- Add a variable by typing the VAR command, by right clicking the variables window and choosing "Add a variable", or by hitting the "Add Variable" button in the variables window.

When adding a variable using the pop-up menu, the debugger displays the following screen.

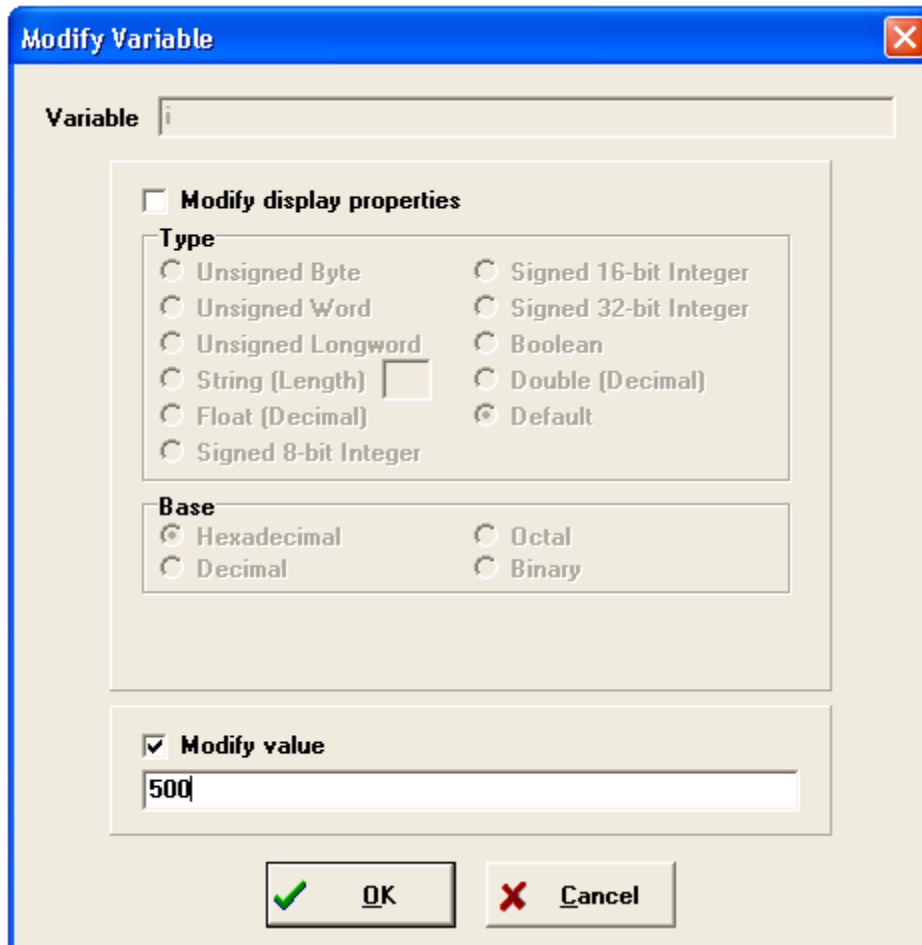


In the variable field, type the address or name of the variable. Typically, set the type of the variable to “Default”, which means that the variable will be displayed as it is defined in the debugging information. When adding a variable, you may specify the numeric display base of the variable.

3.6 Modifying a Variable

- To modify the current value of a variable, right-click the variable name in the variables window and select “Modify Variable” to display a dialog.

Check the “Modify value” checkbox, and type the variable’s new value. After you click the OK button, the debugger updates the variable value on the target, and the debugger refreshes the variable window to display the new value. Note that the debugger will not edit certain user-defined types, such as enumerated types.



- You may also modify a variable's display properties, such as the type or numeric display base using this dialog.

3.7 Using the Register Interpreter

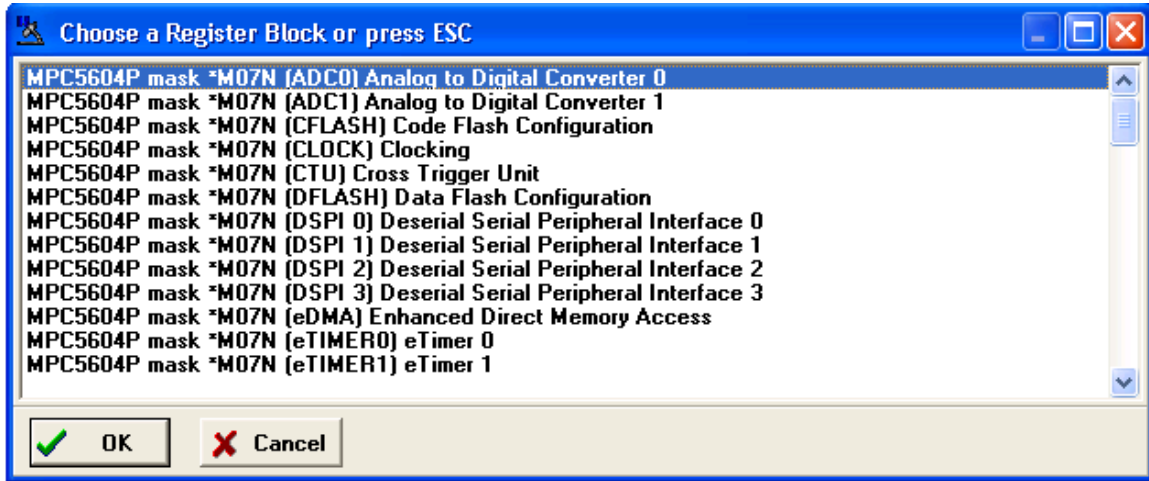
The register interpreter provides a descriptive display of bit fields within the processor's peripheral registers. The register interpreter allows you easily to change the value of these registers. You may quickly check the current state of a peripheral and examine the configuration of the target device.

When you use the register interpreter within the debugger, it reads the current value of the peripheral register, decodes it, and displays it.

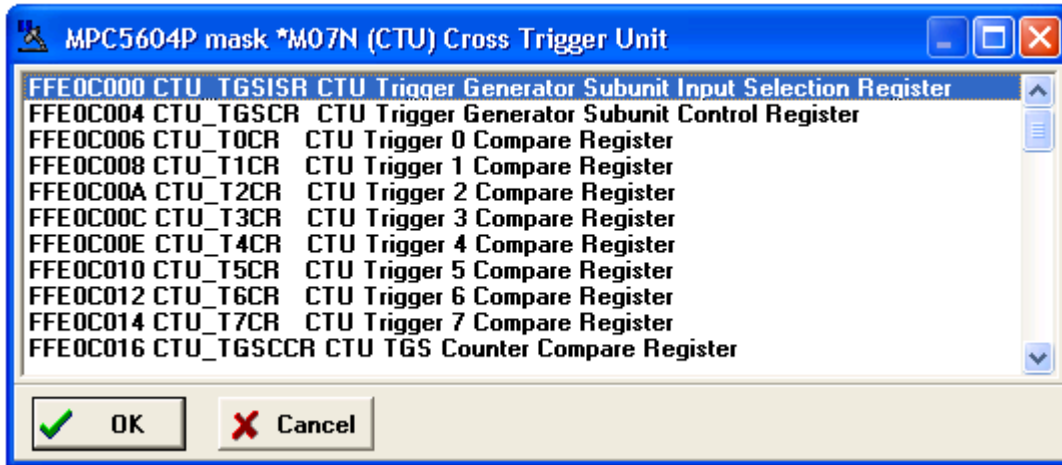
To launch the register interpreter in the debugger, either use the “R” command or click the view/edit register button on the tool bar:



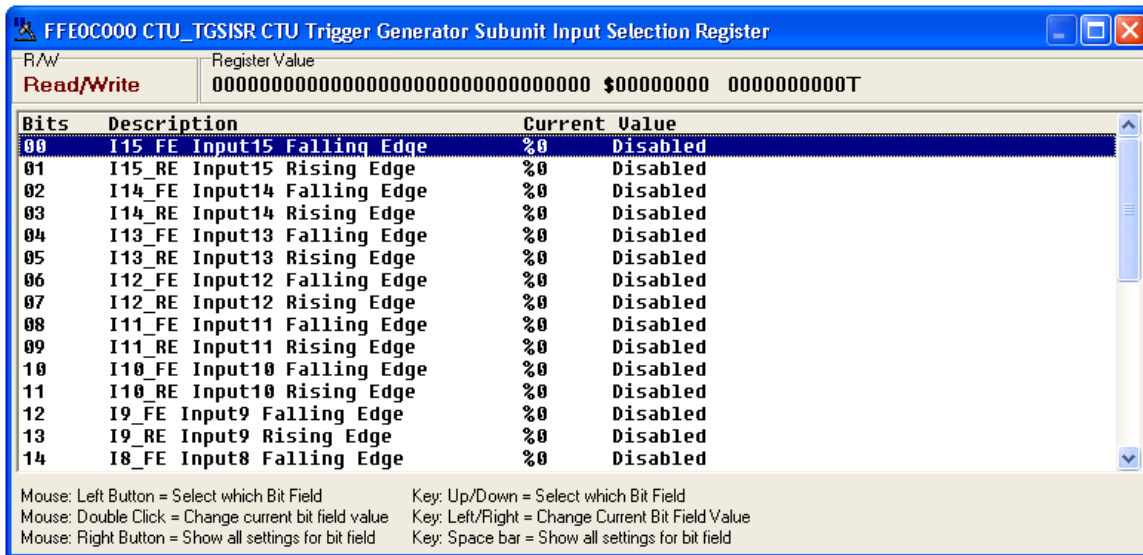
A window will appear that allows you to select a peripheral block to examine.



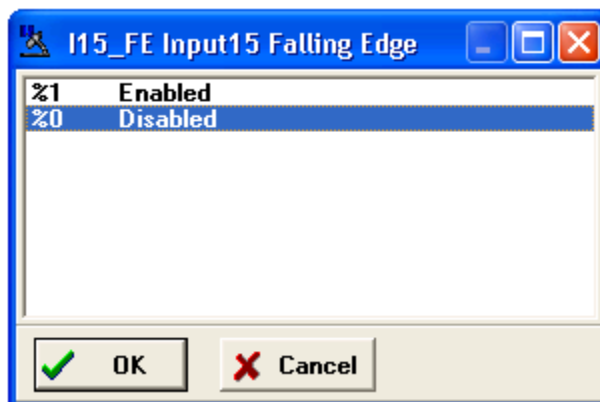
Double clicking the module of choice will launch the register selection window.



Double clicking a specific register will launch the edit/display window for that register.



The window lists the keystrokes and mouse actions, allowing you to modify the values of each of the fields. After right clicking on a specific field, the register interpreter will display all options for that field.



When you quit the register view/edit window by hitting the ESC key, you will be given the opportunity to write the new value into the register, as shown in the following window.



3.8 Adding Register Field Descriptions to the Variables Window

Add register bit fields to the variables window by using the “_TR” command in the debugger or by clicking the "Add Register" button in the variables window. After selecting the register field, the field appears in the debugger variables window, and the debugger will continually update its value.

