




Debugging the Project

The Nios II IDE contains an integrated debugger that allows you to debug your program on Nios II hardware or on the instruction set simulator (ISS). This topic introduces the main features of the debugger. The tutorial assumes you are working with a Nios II hardware target. The process is the same for debugging on the ISS, with the exception of steps that involve the target hardware.


 **Note:** When debugging on Nios II hardware, the FPGA on the development board must be configured with your project's associated SOPC Builder system. If you configured the FPGA before running your project in the previous tutorial step, you do not have to reconfigure the FPGA again unless you reset the board or remove power. If you need to configure the FPGA, follow the steps to configure the hardware in the [Running the Project](#) tutorial topic.



These sections describe how to control the flow of a debug session.

▶ To display line numbers next to each line of code in the editor:

1. Click **Preferences** on the Window menu.
2. Expand **General**.
3. Expand **Editors**.
4. Click **Text Editors**.
5. Turn on **Show line numbers**.
6. Click **OK**.

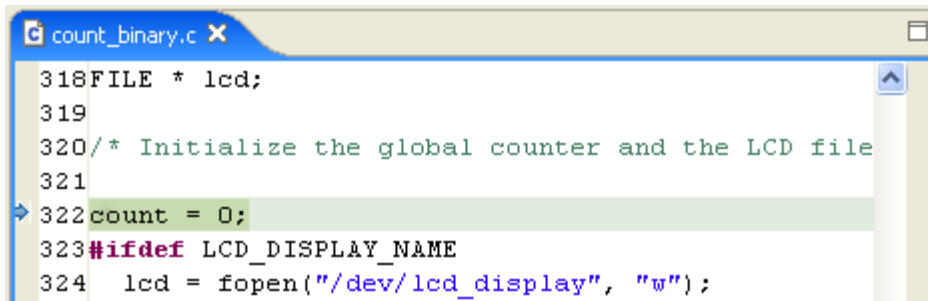
▶ To download executable code and start the debugger:

1. Right-click the **tutorial** project in the Nios II C/C++ Projects view, point to **Debug As**, and then click  **Nios II Hardware**.

 **Note:** To debug on the ISS instead, point to **Debug As**, and then click  **Nios II Instruction Set Simulator**. The `count_binary.c` program used in this tutorial primarily interacts with hardware on the development board which cannot be simulated using the ISS. Some of the steps below do not work with the ISS for this program.

2. If the **Confirm Perspective Switch** dialog box appears, click **Yes**.

After a moment, you see the `main()` function of the Count Binary design in the editor. There is a blue arrow next to the first line of code, as shown below, indicating that execution is stopped on this line. Note that the exact line numbers might vary on your screen.



```


318 FILE * lcd;
319
320 /* Initialize the global counter and the LCD file
321
322 count = 0;
323 #ifdef LCD_DISPLAY_NAME
324   lcd = fopen("/dev/lcd_display", "w");

```


Notice that the perspective of the Nios II IDE changed from the Nios II C/C++ development perspective to the Debug perspective. A perspective is a different configuration of the Nios II IDE workbench. Refer to [Related Topics](#) for more information about perspectives. You can switch between perspectives anytime by pointing to **Open Perspective** on the Window menu or by clicking on the shortcut buttons near the upper-right of the Nios II IDE window.

When targeting Nios II hardware, the **Debug As** command does the following:




1. Creates a default run/debug configuration for the target board.


 **Note:** This step usually completes automatically without user intervention. If it cannot, the IDE displays an error message and you must manually set up a run/debug configuration. The most common reason for manual intervention is having multiple JTAG download cables installed. In this case you need to select one manually.

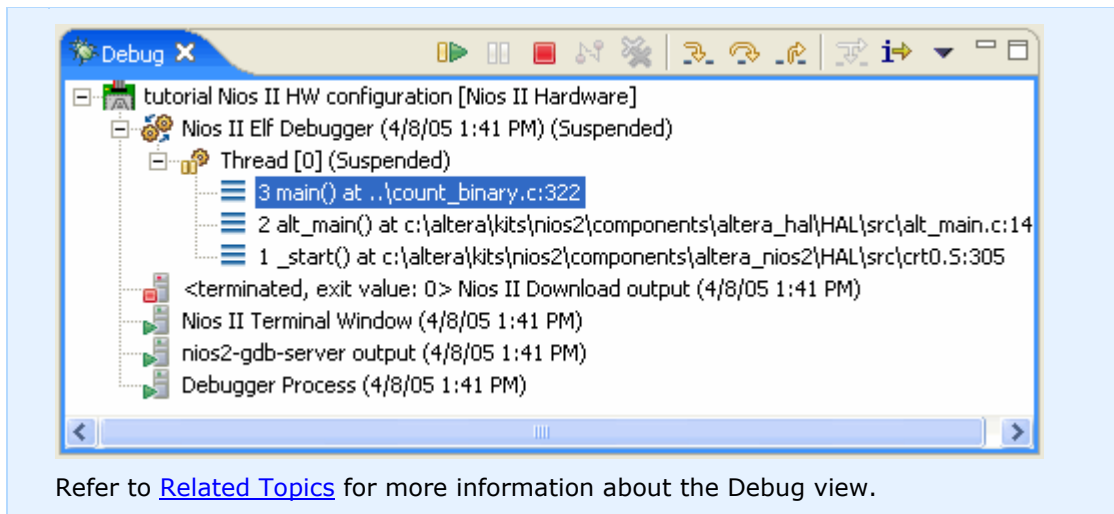
2. Builds the project. If the project is not up-to-date, then the IDE builds it first to generate an up-to-date executable file.
3. Establishes communication with the target board, and verifies that the expected SOPC Builder system is configured in the FPGA.
4. Downloads the executable file (**.elf**) to memory on the target board.
5. Sets a breakpoint at `main()`.
6. Instructs the Nios II processor to begin executing the code.


After using the **Debug As** command once, you can click  **Debug** on the toolbar to start the debugger again.

▶ To resume and suspend execution:




- Click  **Resume** in the Debug view to resume execution. You can also resume execution by pressing **F8**.
- Click  **Suspend** in the Debug view to suspend execution. If the processor suspends outside the scope of the current file, the IDE opens the source file corresponding to the current program counter.
- Click  **Terminate** in the Debug view to end the debug session and disconnect from the target.

 **Note:** The Debug buttons are context sensitive, depending on the currently highlighted selection in the Debug view. Make sure you select an item under **Nios II Elf Debugger**, as shown below, to ensure you are debugging the desired thread.



If you accidentally terminate the debug session, or the download cable connection is interrupted, you can easily start a new debugging session by clicking  **Debug** on the toolbar.



►To step through the C/C++ code line by line:

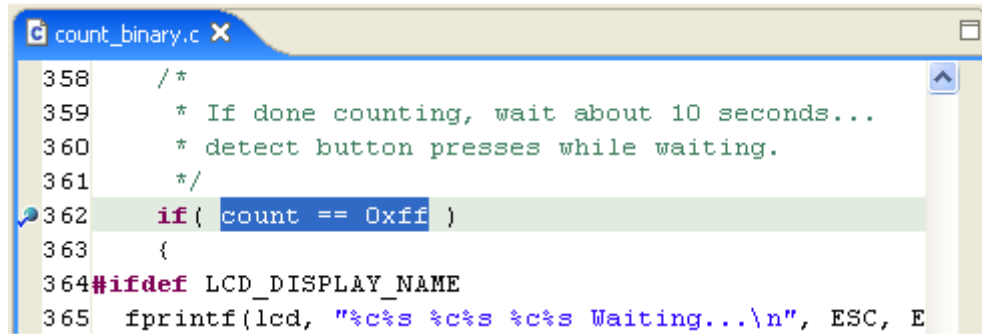
- Click  **Step Into**. If executing a line of code that calls a function, the debugger steps into the function. Otherwise it executes the line of code and suspends on the next line in the current function. You can also step into a function by pressing **F5**.
- Click  **Step Over**. If executing a line of code that calls a function, the debugger executes the entire called function and suspends on the next line in the current function. Otherwise it executes the line of code and suspends on the next line in the current function. You can also step over a line of code by pressing **F6**.
- Click  **Step Return**. The debugger finishes executing the current function, returns to the calling function, and suspends on the next line in the calling function. You can also step return from a function by pressing **F7**.

►To use breakpoints and watchpoints:

You can set breakpoints on specific lines of code, remove breakpoints, or disable them temporarily. Enabled breakpoints suspend execution when the processor reaches that line of code.

To set a breakpoint for this example:


1. If the processor is running, select **Thread [0] (Running)** in the Debug view and click  **Suspend**. You can only add breakpoints while the processor is suspended.
2. Click the **count_binary.c** tab in the editor.
3. On the Edit menu, click **Find/Replace....**
4. Type `count == 0xff` in the **Find** box, then click **Find**. The editor displays the appropriate line of code.
5. Click **Close** in the **Find/Replace** dialog box.
6. Double-click in the margin next to the line `if(count == 0xff)` to set a breakpoint. You can also right-click the margin and click **Toggle Breakpoint**. The  breakpoint symbol appears in the margin, as shown below.




```



358  /*
359  * If done counting, wait about 10 seconds...
360  * detect button presses while waiting.
361  */
362  if( count == 0xff )
363  {
364  #ifdef LCD_DISPLAY_NAME
365  fprintf(lcd, "%c%s %c%s %c%s Waiting...\n", ESC, E

```


 **Note:** You must click in the margin to the left of the line of code. Clicking within the editor does not affect breakpoints.

- Click  **Resume** in the Debug view. The processor resumes and then suspends just before executing the line of code with the breakpoint. The editor displays an arrow in the margin next to the suspended line of code. It might take a moment for the program to execute to the breakpoint. Resume again to iterate through the loop another time.

To remove a breakpoint:

Double-click the  breakpoint symbol in the margin. You can also right-click the  breakpoint symbol, and then click **Toggle Breakpoint**.

To disable a breakpoint:

Right-click the  breakpoint symbol, and then click **Disable Breakpoint**. Disabling temporarily prevents a breakpoint from suspending the processor while leaving it in place for future reference.

To use Breakpoints view:

- Click the **Breakpoints** tab in the upper-right pane of the Debug perspective to display the Breakpoints view. This view displays the location and status of all breakpoints you have previously set on specific lines in the code every time the processor hits a breakpoint or suspends. Values that have changed since the last time the processor suspended display in red.
- Right-click a breakpoint in the list, and then click **Enable**, **Disable**, or **Remove** to change the status of the breakpoint.



The Breakpoints view also displays watchpoints. Refer to [Related Topics](#) for more information about debugging with watchpoints.


Several default views in the Debug perspective help you to organize, navigate, and analyze your project during a debug session. The Nios II IDE updates each view every time the processor hits a breakpoint or suspends. Values that have changed since the last time the processor suspended display in red.

► To view disassembly:

When the processor suspends, Disassembly view automatically appears. You can also open the Disassembly view from the Window menu by pointing to **Show View**, and clicking

Disassembly. This view displays the assembly language instructions interleaved with the C/C++ source code.

- Click  **Instruction Stepping Mode** in the Debug view toolbar to allow single stepping through the individual assembly instructions. Stepping through assembly code advances the instruction pointer in the Disassembly view. Because multiple assembly instructions represent a single line of C/C++ code, the instruction pointer might not advance in the C/C++ Editor view with each step through the assembly instructions.
- Click  **Instruction Stepping Mode** a second time to return to single stepping in the Editor view at the C/C++ statement level.

 **Note:** If you do not have the source code for a function, stepping through the code automatically uses Disassembly view regardless of whether instruction stepping mode is on or off.

►To view stack trace:

The Debug view displays the program execution stack and dynamically updates it as you step through code. Any time the processor suspends, the Debug view displays the name of the suspended function, and the sequence of function calls that led up to the current program counter. This view provides a snapshot of your current position within the program execution.

Refer to [Related Topics](#) for more information about the Debug view.

►To view execution trace:

On the Window menu, point to **Show View**, and then click **Trace**. When executing a program on a Nios II hardware target, the Trace view displays the exact execution trace of the program running in hardware. This view provides a snapshot of the specific code that executed to arrive at the current position.


►To track variables:


Click the **Variables** tab in the upper-right pane of the Debug perspective to display the Variables view. You can also point to **Show View** on the Window menu, and then click **Variables**.

Local Variables

The Variables view automatically displays all variables local to the scope where the processor is suspended. Use the Variables view to track and change variable values on-the-fly during a debug session. This is useful to test your program's response to specific conditions, or to force a loop index to skip over a loop.




The Variables view is context sensitive, depending on the currently selected function in the Debug view's stack trace display. Selecting different functions allows you to see the variables (and their current values) defined at each level of the stack trace.

 **Note:** Hovering the mouse over a variable in the source code displays the variable's value as a tool-tip. This is often the easiest way to see the value of a variable in the current scope.

To change the value of a variable, right-click the variable name and then click  **Change Value....** This opens the **Set Value** dialog box, which allows you to specify a new value for the variable.

Global Variables

You can also selectively display global variables, which are variables defined outside the scope of all functions, but are available from within any function. In the **count_binary.c** example, to track the global variable `count` in the Variables view, do the following:

1. If the processor is running, click **Thread [0] (Running)** in the Debug view, and click  **Suspend**.
2. Right-click in the Variables view and then click  **Add Global Variables...**, or click  **Add Global Variables** on the Variables view toolbar.
3. Scroll down and turn on **count**.
4. Click **OK**. The variable `count` and its current value appears in the Variables view. Because `count` is declared as a `char` type, it displays in ASCII format by default in the Variables view.

Now, every time execution suspends, the Variables view displays the value for the variable `count`.

Variables Display Format

You can change the display format of variable values appearing in the Variables view in two ways. For example, to change the display format of a single variable to hexadecimal, do the following:

- Right-click the variable, point to **Format**, and then click **Hexadecimal**. The format of the value changes.

To change the display format of all variables to hexadecimal, do the following:

1. On the Window menu, click **Preferences**.
2. Expand **C/C++, Debug**.
3. Select **Hexadecimal** in the **Default variable format** list.
4. Click **OK** to close the **Preferences** dialog box.

The format of the values for local variables changes the next time you resume execution. The format of the values for global variables changes the next time you restart the debug session.

Refer to [Related Topics](#) for more information about variables.

▶ To track watch expressions:

Click the **Expressions** tab in the upper-right pane of the Debug perspective to display the Expressions view. You can also point to **Show View** on the Window menu, and then click **Expressions**. This view displays user-specified C expressions evaluated at the current scope. When the processor suspends, the Expressions view evaluates each of the expressions, and displays the value.

In the **count_binary.c** example, to track the arbitrary expression `count==5` in the Expressions view, do the following:

1. If necessary, use a breakpoint to stop execution on the line `if(count == 0xff)`, as described in the [breakpoints](#) section of this topic.

```

358  /*
359   * If done counting, wait about 10 seconds...
360   * detect button presses while waiting.
361   */
362  if( count == 0xff )
363  {
364  #ifdef LCD_DISPLAY_NAME
365    fprintf(lcd, "%c%s %c%s %c%s Waiting...\n", ESC, E

```

- Highlight the expression `count == 0xff` in the code.
- Right-click and then click **Add Watch Expression...**. The **Add Watch Expression** dialog box appears with the **Expression to watch** box automatically filled in with the selected text: `count==0xff`.
- Change **Expression to watch** to `count==5`.
- Click **OK**. The expression `"count==5" = 0` appears in the **Expressions** view in the upper-right window.

Now, every time execution suspends at a breakpoint, the Expressions view evaluates the expression `count==5`. If execution suspends when `count` equals 5, then `count==5` evaluates true, signified by the expression `count==5=1`.

You can assign an IDE preference to show expressions values in a different format, such as hexadecimal, as described in [variables](#) section of this topic. However, you must restart the debug session to force the format to change. Refer to [Related Topics](#) for more information about expressions.

► To view and edit registers:

Click the **Registers** tab in the upper-right pane of the Debug perspective to display the Registers view. You can also point to **Show View** on the Window menu, and then click **Registers**. This view displays the contents of registers.

To edit the contents of a particular register, right-click it in the Registers view and click **Change Value...**

Note: Altera recommends that you do not manually edit register values, because it could cause your program to behave unpredictably.





You can assign an IDE preference to show register values in a different format, such as hexadecimal, as described in [variables](#) section of this topic. However, you must restart the debug session to force the format to change. Refer to [Related Topics](#) for more information about registers.


► To view and edit memory:

Click the **Memory** tab in the upper-right pane of the Debug perspective to display the Memory view. You can also point to **Show View** on the Window menu, and then click **Memory**. This view displays the contents of memory.

The Memory view allows you to track multiple locations in memory without having to continually type in addresses by hand. You can track constants and expressions, which allows you to search for a memory location by symbolic name.

The following steps demonstrate various uses of the Memory view.

1. If the processor is running, click **Thread [0] (Running)** in the Debug view, and then click  **Suspend**.
2. In the Memory view, click  **Add Memory Monitor**.
3. Type `0x00` and click **Ok**. Note the contents.
4. Right-click in the memory contents area, and click **Format...**
5. Select **Column Size**, and then click **2**. Note how the display changes.
6. Click  **Add Memory Monitor** a second time.
7. Type `&count` and click **Ok**. This displays memory contents at the location where variable `count` is stored.
8. Click on the highlighted values in the memory contents area and highlight just the two digits representing the eight bit value for `count`.
9. Type a value, such as `66`, and press Enter. Note the memory contents in the Memory view and the value of `count` in the Variables view change.
10. Click  **Add Rendering** at the upper-right of the view.
11. Click **ASCII** and then click **Ok**. Note the additional tab displaying memory contents in ASCII format.

 **Note:** If the memory region displayed is located in read-only memory, the Nios II IDE does not respond to attempts to edit the memory content.

Refer to [Related Topics](#) for more information about memory.

Next: [Editing the Project Properties](#)

Previous: [Running the Project](#)



Related Nios II IDE Help Topics

- [About Running and Debugging Projects](#)
- [Instruction Set Simulator \(ISS\)](#) - Contains details on the capabilities and limitations of the ISS.
- [Advanced Debugging Features by FS2](#)



Related Eclipse and CDT Help Topics

- C/C++ Development User Guide > Reference > C/C++ Views and Editors > Debug view > Debug view
- C/C++ Development User Guide > Tasks > Running and Debugging > Adding expressions
- C/C++ Development User Guide > Tasks > Running and Debugging > Debugging a program > Adding watchpoints
- C/C++ Development User Guide > Tasks > Running and Debugging > Working with memory
- C/C++ Development User Guide > Tasks > Running and Debugging > Working with registers
- C/C++ Development User Guide > Tasks > Running and Debugging > Working with variables
- Workbench User Guide > Tasks > Working with perspectives