

Use xTIMEcomposer to debug a program

IN THIS DOCUMENT

- ▶ Launch the debugger
- ▶ Control program execution
- ▶ Examine a suspended program
- ▶ Set a breakpoint
- ▶ View disassembled code

The xCORE Debugger lets you see what's going on "inside" your program while it executes on hardware or on the simulator. It can help you identify the cause of any erroneous behavior.

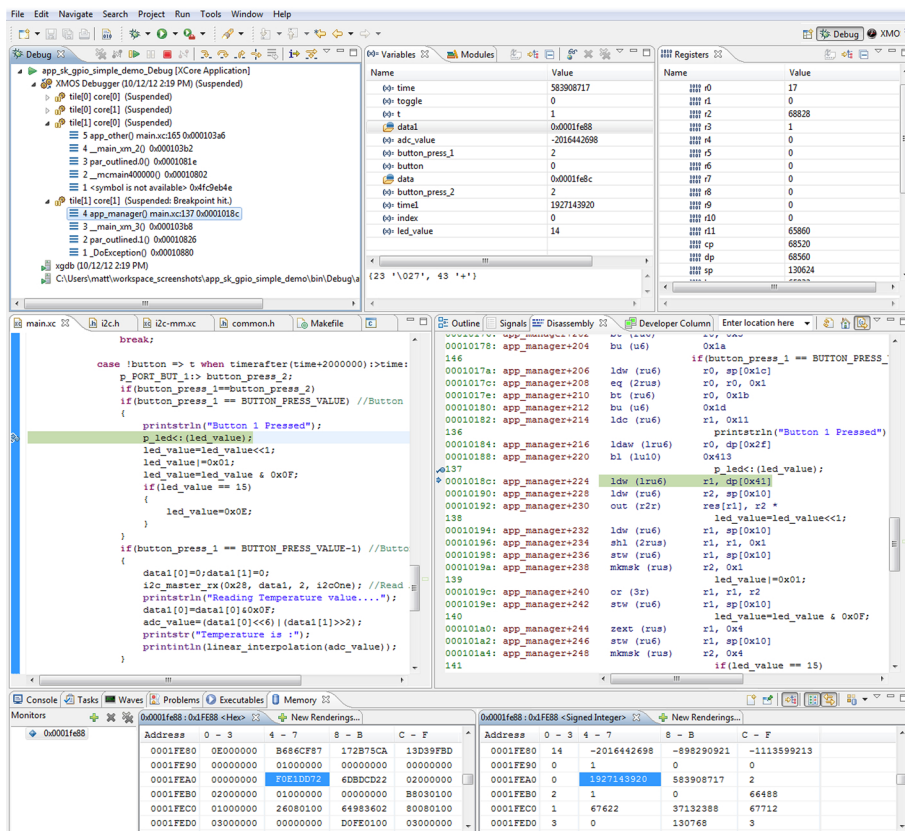


Figure 1:
Debug
perspective

Publication Date: 2013/11/11

REV B

XMOS © 2013, All Rights Reserved





For full visibility of your program, you must compile it with debugging enabled (see [XM-000927-PC](#)). This causes the compiler to add symbols to the executable that let the debugger make direct associations back to the source code. Note that compiling with optimizations enabled (see [XM-000927-PC](#)) can also make debugging more difficult.

1 Launch the debugger

To load a program under control of the debugger, follow these steps:

1. Select a project in the **Project Explorer**.
2. Choose **Run ► Debug Configurations**.
3. In the left panel, double-click **XCore Application**. xTIMEcomposer creates a new configuration and displays the default settings in the right panel.
4. In the **Name** text box, enter a name for the configuration.
5. xTIMEcomposer tries to identify the target project and executable for you. To select one yourself, click **Browse** to the right of the **Project** text box and select your project in the **Project Selection** dialog box. Then click **Search Project** and select the executable file in the **Program Selection** dialog box.



You must have previously compiled your program without any errors for the executable to be available for selection.

6. If you have a development board connected to your system, in the the **Device options** panel check the **hardware** option and select your debug adapter from the **Adapter** list. Alternatively, check the **simulator** option to run your program on the simulator.
7. To save the configuration and launch the debugger, click **Debug**. If you are asked whether to open the **Debug** perspective, check **Remember my decision** and click **Yes**.

xTIMEcomposer loads your program in the debugger and opens it in the **Debug** perspective.



xTIMEcomposer remembers the configuration last used to load your program. To debug the program later using the same settings, just click the **Debug** button. To use a different configuration, click the arrow to the right of the **Debug** button and select a configuration from the drop-down list.

2 Control program execution



Once launched, the debugger runs the program until either an exception is raised or you suspend execution by clicking the **Suspend** button .



Click the **Resume** button to continue executing a suspended program, or use one of the step controls to advance the core selected in the **Debug** view incrementally:



► **Step Into:** Executes a single line of source code on the core selected in the **Debug** view. If the next line of code is a function call, the debugger suspends at the first statement in the called function. All other cores are resumed.



► **Step over:** Executes a single line of source code on the core selected in the **Debug** view. All other cores are resumed.



► **Step return:** Steps the core selected in the **Debug** view until the current function returns. If the next line of code is a function call, the debugger executes the entire function. All other cores are resumed.



► **Step through:** Switches the debugger context to the corresponding input core of a channel output statement. This is useful for following the path of data as it flows between cores. No cores are resumed.



When debugging optimized code, a step operation is not guaranteed to advance to the next line in the source code, since the compiler may have reordered instruction execution to improve performance.

3 Examine a suspended program

Once a program is suspended, you can query the state of each core and can inspect the values held in registers and memory.

► **Examine a core's call stack:** The **Debug** view displays a list of software tasks, each of which can be expanded to show its call stack, as shown in Figure 2.

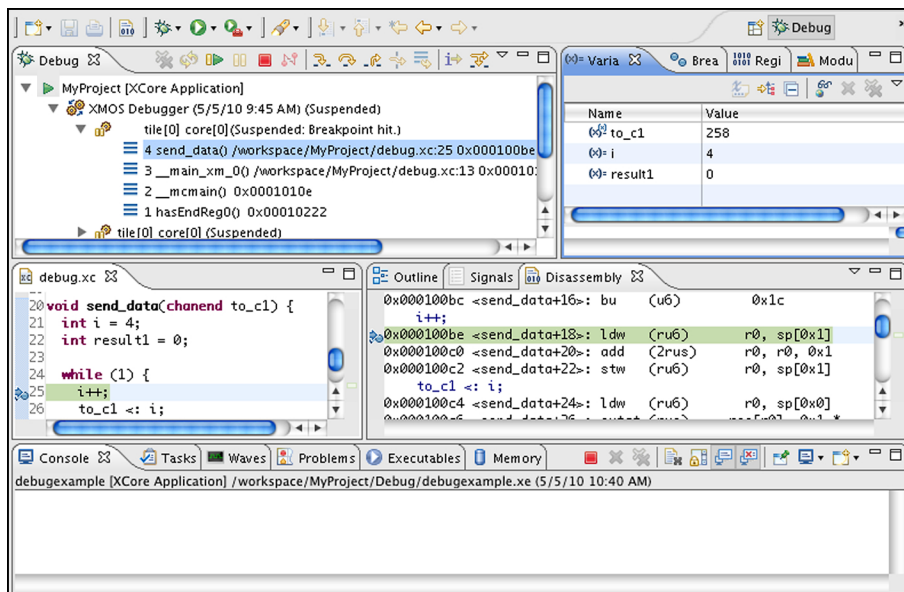
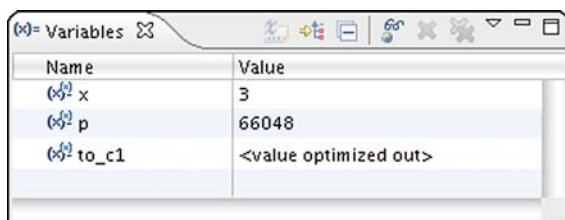


Figure 2:
Debug view

In the example above, the tile `tile[0]` is suspended at a breakpoint in the function `send_data` on line 25 of the file `debug.xc`.

- **Examine Variables:** The **Variables** view displays variables and their values. In the **Debug** view click on any function in a core's call stack to view its variables, as shown in Figure 3.



Name	Value
x	3
p	66048
to_c1	<value optimized out>

Figure 3:
Variables
view

To view a global variable, right-click in the **Variables** view, select **Add Global Variables** from the pop-up menu to open a dialog box and select the global variable to add to the view.



Compiling a program without optimizations guarantees that every variable is held in memory for the duration of its scope so that its value can always be displayed. If optimizations are enabled, a variable may not be available to be examined, resulting in the message `<value optimized out>`.

You can do the following with variables:

- **Display a variable's value in hexadecimal format:** Right-click on a variable to bring up a menu and choose **Format ► Hexadecimal**. You can also choose binary, decimal or normal. The normal format is determined by the type of the variable.
- **Change a variable's value:** Click on a value to highlight it, enter a new value and press **Enter**. The table entry is highlighted yellow to indicate its value has changed. This allows you to test what happens under what-if scenarios.
- **Prevent the debugger from reading a variable:** Right-click on a variable and choose **Disable** from the contextual menu. This is useful if the variable's type is qualified with `volatile`. To apply settings to multiple variables at once, press **Ctrl** (Windows, Linux) or **⌘** (Mac) while you click on multiple variables, then right-click and select an option from the contextual menu.
- **Examine Memory:** The **Memory** view provides a list of memory monitors, each representing a section of memory. To open the Memory view, choose **Window ► Show View ► Memory**. In the **Debug** view click on any core to view the contents of its memory, as shown in Figure 3.



To specify a memory location to view, click the **Add** button to open the **Memory Monitor** dialog box, enter a memory location and click **OK**. You can enter either an absolute address or a C/XC expression. To view the contents of an array just enter its name.



Figure 4:
Memory view

To display the memory contents in a different format such as Hex or ASCII, click the **New Renderings** tab, select a format and click **Add Renderings**. xTIMEcomposer adds new tabs to the right of the **Memory** view, each showing a different interpretation of the values in memory.

4 Set a breakpoint

A *breakpoint* is a marker in the program that instructs the debugger to interrupt execution so that you can investigate the state of the program. You can add a breakpoint to any executable line of code, causing execution to suspend before that line of code executes.



To add a breakpoint, double-click the marker bar in the left margin of the code editor next to the line at which you wish to suspend execution. A blue dot is displayed to indicate the presence of the breakpoint. Note that the breakpoint applies to every core that executes the function.

Breakpoints are also displayed in the **Breakpoints** view. To open the Breakpoints view, choose **Window ► Show ► View ► Breakpoints**. Double-click on a breakpoint to locate the corresponding line in the source code editor.

Here are some other things can do with breakpoints:

- **Set a conditional breakpoint:** Right-click on a breakpoint marker to bring up a contextual menu, and choose **Breakpoint Properties** to display a properties dialog box. Click the **Common** option in the left panel and enter a C/XC conditional expression in the **Condition** text box in the right panel. The expression can contain any variables in the scope of the breakpoint.
- **Set a conditional breakpoint:** Right-click on a breakpoint marker to bring up a contextual menu, and choose **Breakpoint Properties** to display a properties dialog box. Click the **Common** option in the left panel and enter a C/XC conditional expression in the **Condition** text box in the right panel. The expression can contain any variables in the scope of the breakpoint.
- **Set a watchpoint on a global variable:** A watchpoint is a special breakpoint that suspends execution whenever the value of an expression changes (without specifying where it might happen). Right-click anywhere in the **Breakpoints** view and choose **Add Watchpoint C/XC** from the contextual menu. Enter a C/XC

expression in the dialog box, for example `a[MAX]`. Select **Write** to break when the expression is written, and **Read** to break when the expression is read.

- **Disable a breakpoint:** In the **Breakpoints** view, clear the checkbox next to a breakpoint. Enable the checkbox to re-enable the breakpoint.
- **Remove a breakpoint:** Double-click on a breakpoint marker in the code editor to remove it. Alternatively, right-click a breakpoint in the **Breakpoints** view and select **Remove** from the contextual menu; to remove all breakpoints, select **Remove All**.

5 View disassembled code

The **Disassembly** view displays the assembly instructions that are executed on the target platform. To open the **Disassembly** view, choose **Window ► Show View ► Disassembly**.

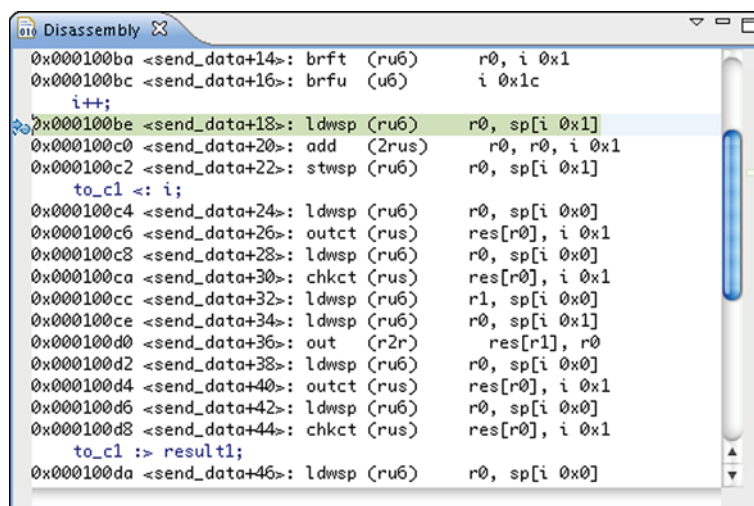


Figure 5:
Disassembly
view



xTIMEcomposer automatically enables instruction stepping mode whenever the **Disassembly** view has focus. Alternatively, click the **Instruction Stepping Mode** button to enable. Once enabled, click the **Step** button to advance the program by a single assembly instruction.



Copyright © 2013, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.