

Use xTIMEcomposer to time a program

IN THIS DOCUMENT

- ▶ Launch the timing analyzer
 - ▶ Time a section of code
 - ▶ Specify timing requirements
 - ▶ Add program execution information
 - ▶ Validate timing requirements during compilation
-

The xCORE Timing Analyzer lets you determine the time taken to execute code on your target platform. Due to the deterministic nature of the xCORE architecture, the tools can measure the shortest and longest time required to execute a section of code. When combined with user-specified requirements, the tools can determine at compile-time whether all timing-critical sections of code are guaranteed to execute within their deadlines.

1 Launch the timing analyzer

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

1. Select a project in the **Project Explorer**.
2. Choose **Run ▶ Time 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. 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.

5. In the **Name** text box, enter a name for the configuration.
6. To save the configuration and launch the timing analyzer, click **Time**.

xTIMEcomposer loads your program in the timing analyzer and opens it in the **Timing** perspective. In this perspective the editor is read-only, to ensure the relationship between the binary and source code remains consistent.

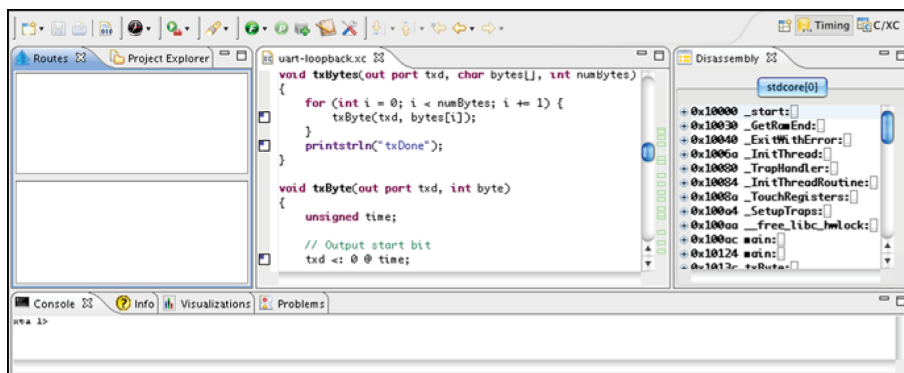


Figure 1:
Timing
perspective



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

2 Time a section of code

A *route* consists of the set of all paths through which control can flow between two points (or *endpoints*) in a program. Each route has a best-case time, in which branches always follow the path that takes the shortest time to execute, and a corresponding worst-case time.

To specify a route and analyze it, follow these steps:



1. Right-click on an endpoint marker in the editor margin and choose **Set from endpoint**. xTIMEcomposer displays a green dot in the top-right quarter of the marker.



2. Right-click on an endpoint marker and choose **Set to endpoint**. xTIMEcomposer displays a red dot in the bottom-right quarter of the marker.

You can specify a start point above an end point. You can also specify a start point at or below an end point, defining a route whose paths flow out and then back into the function. This is typical of functions called multiple times or from within a loop.



3. Click the **Analyze Endpoints** button in the main toolbar. xTIMEcomposer analyzes all the paths in the specified route, displaying a tree-like representation in the lower panel of the **Routes** view and a graph-like representation in the **Structure** tab of the **Visualizations** view.



Alternatively, to analyze the time taken to execute a function, just click the **Analyze Function** button in the main toolbar and select a function from the drop-down list.

xTIMEcomposer provides endpoint markers for all statements whose order is guaranteed to be preserved during compilation. These statements include I/O operations and function calls.

2.1 Visualize a route

The **Routes** view displays a structural representation of the route. Each time you analyze a route, an entry is added to the top panel. Click on a route to view it in the bottom panel. It is represented using the following nodes:

- ★ A source-level function.
- ↓ A list of nodes that are executed in sequence.
- 🔍 A set of nodes that are executed conditionally.
- 🔄 A loop consisting of a sequence of nodes in which the last node can branch back to the first node.
- ☰ A block containing a straight-line sequence of instructions.
- A single machine instruction.

2.2 The Visualizations view

The **Visualizations** view provides graphical representations of the route. The Structure tab represents the route as a line that flows from left to right, as shown in the example below. The route forks into multiple paths whenever the code branches, and all paths join at its end. The best-case timing path is highlighted in green, the worst-case path in red, and all other paths are colored gray.

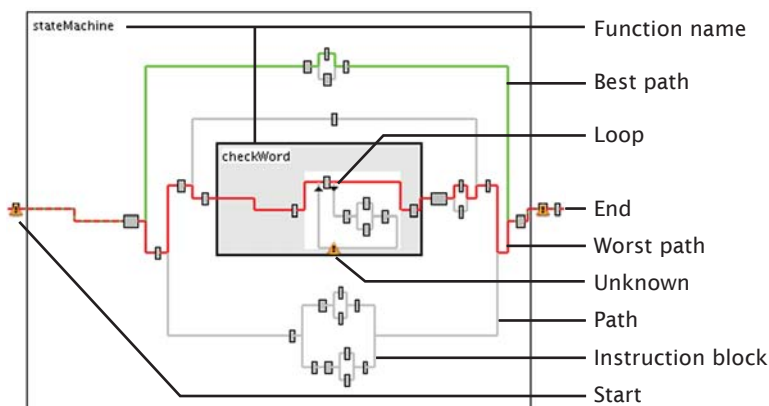






Figure 2:
Visualizations
view

In both the **Route** view and **Structure** view, you can hover over a node to display a summary of its timing properties. Click on a node to highlight its source code in

the editor, or double-click to go to the line at the start of the node. In the **Structure** view, double-click on a function name to expand or collapse it.

3 Specify timing requirements

A *timing requirement* specifies how long the paths in a route may take to execute for the program to behave correctly. In the top panel of the **Routes** view, the status of each route is indicated by an icon to the left of its name:

-  No timing requirement is specified.
-  A timing requirement is specified and met.
-  A timing requirement is specified and met, subject to all I/O instructions being ready to execute.
-  A timing requirement is specified and not met.

To specify a timing requirement, right-click on a route and choose **Set timing requirements**. A dialog box opens. Enter the maximum time in which the paths must execute in either *ns*, *cycles* or *MHz* and click **OK**. xTIMEcomposer updates the status of the route.

4 Add program execution information

Under some conditions the timing analyzer is unable to prove timing without additional information. Examples of common conditions include:

- ▶ The route contains an I/O instruction that can pause for an unknown length of time.
- ▶ The route contains a loop with a data-dependent exit condition.
- ▶ A path fails to meet timing, but the path is only executed as a result of an error condition and is not therefore timing critical.

In these cases you can provide the timing analyzer additional information about the execution of your program. Armed with this additional information, the analyzer may then be able to prove that a route's timing requirement is met. Information you can provide includes:

- ▶ **The number of loop iterations:** Right-click on a loop node and choose **Set loop iterations** to display a dialog box. Enter a maximum loop count and click **OK**.
- ▶ **The maximum pause time for an I/O instruction:** Right-click on an instruction node and choose **Set instruction time** to display a dialog box. Enter a value, select a unit of time/rate (such as nanoseconds or MHz) and click **OK**.
- ▶ **Exclude a path from the route:** Right-click on a node and choose **Exclude**.

4.1 Refine the worst-case analysis

By default, the timing analyzer assumes that a route always follows branches that take the longest time to execute. If you know that this is not the case, for example through inspection during simulation or a formal analysis of your program, you can refine the parameters used by the analyzer. Refinements you can make include:

- ▶ **Specifying an absolute execution time for a function call:** Right-click on a function node and choose **Set function time** to open a dialog box. Enter a time and click **OK**.
- ▶ **Specifying an absolute time for a path:** Select a path by holding down Ctrl (Windows, Linux) or ⌘ (Mac) and clicking on two instruction nodes, then right-click and choose **Set path time** to open a dialog box. Enter a time and click **OK**.
- ▶ **Specifying the number of times a node is executed:** By default, the analyzer assumes that the number of times a node is executed is the multiplication of each loop count in its scope. To change the iteration count to be an absolute value, right-click on a node and choose **Set loop scope** to open a dialog box. Select **Make scope absolute** and click **OK**.
- ▶ **Specifying the number of times a conditional is executed in a loop:** By default, the analyzer assumes that a conditional node always follows the path that takes the longest time to execute. To specify the number of times a conditional target is executed, right-click on the target node and choose **Set loop path iterations** to open a dialog box. Enter the number of iterations and click **OK**.

5 Validate timing requirements during compilation

Once you've specified the timing requirements for your program, including any refinements about its execution, you can generate a script that checks these requirements at compile-time.

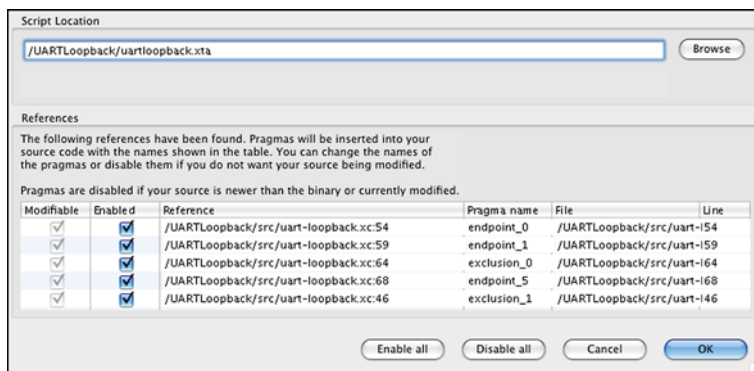
To create a script that checks all timing requirements specified in the **Routes** view, follow these steps:



1. Click the **Generate Script** button.
2. In the **Script location** text box, enter a filename for the script. The filename must have a **.xta** extension.
3. To change the names of the pragmas added to the source file, modify their values in the **Pragma name** fields.
4. Click **OK** to save the script and update your source code. xTIMEcomposer adds the script to your project and opens it in the editor. It also updates your source files with any pragmas required by the script.

The next time you compile your program, the timing requirements are checked and any failures are reported as compilation errors. Double-click on a timing error to view the failing requirement in the script.

Figure 3:
Script
Options
dialog box



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.