

xTIMEcomposer Simulator Tutorial

IN THIS DOCUMENT

- ▶ Introduction
 - ▶ Open the simulator example
 - ▶ Output a VCD trace file
 - ▶ Relate I/O signals to source code
 - ▶ Connect ports in a loopback
 - ▶ What to read next
-

1 Introduction

xTIMEcomposer includes a cycle-based simulator that you can use to verify the behaviour of programs without the need for target hardware. xTIMEcomposer also includes a waveform viewer for visualising I/O data signals generated by the simulator, and relating the signals back to the source code.

This tutorial shows you how to:


- ▶ Run a program on the simulator and view the processor instruction trace
- ▶ Save the I/O data signal generated by the simulator and view it in the waveform viewer
- ▶ Relate the trace data to the source code
- ▶ Connect two ports together to create a loopback

2 Open the simulator example

This part of the tutorial shows you how to create a project in xTIMEcomposer and output a simulator trace to the *Console*.

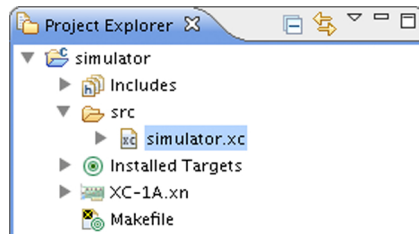
2.1 Create a project

Before writing any code, you need a project to store your files in. To create a new project follow these steps:

1. Choose **File > New > XDE Project**  to open the **New XDE Project** dialog.
2. In **Project Name**, enter a name such as **simulator**.
3. In **Target Hardware**, select the option **XC-1A Board**. You don't need this board to complete the tutorial.

4. In **Application Software**, select the option **Empty XC File**.

5. Click **Finish**.



2.2 Add the code

The program below implements a UART interface:

```
/*
 * =====
 * Name      : uart-loopback.xc
 * Description : UART loopback example
 * =====
 */

#include <xs1.h>
#include <print.h>
#include <platform.h>

#define NUM_BYTES 3
#define BIT_RATE 115200
#define BIT_TIME XS1_TIMER_HZ / BIT_RATE

void txBytes(out port txd, char bytes[], int numBytes);
void txByte(out port txd, int byte);
void rxBytes(in port rxd, char bytes[], int numBytes);
char rxByte(in port rxd);

out port txd = XS1_PORT_1H;
in port rxd = XS1_PORT_1I;

int main()
{
    char transmit[] = { 0b00110101, 0b10101100, 0b11110001 };
    char receive[] = { 0, 0, 0 };

    // Drive port high (inactive) to begin
    txd <: 1;

    par {
        txBytes(txd, transmit, NUM_BYTES);
        rxBytes(rxd, receive, NUM_BYTES);
    }
}
```

```
    return 0;
}

void txBytes(out port txd, char bytes[], int numBytes)
{
    for (int i = 0; i < numBytes; i += 1) {
        txByte(txd, bytes[i]);
    }
    printstrln("txDone"); // Transmit_Done
}

void txByte(out port txd, int byte)
{
    unsigned time;

    // Output start bit
    txd <: 0 @ time; // Endpoint A

    // Output data bits
    for (int i = 0; i < 8; i++) {
        time += BIT_TIME;
        txd @ time <: >> byte; // Endpoint B
    }

    // Output stop bit
    time += BIT_TIME;
    txd @ time <: 1; // Endpoint C

    // Hold stop bit
    time += BIT_TIME;
    txd @ time <: 1; // Endpoint D
}

void rxBytes(in port rxd, char bytes[], int numBytes)
{
    for (int i = 0; i < numBytes; i += 1) {
        bytes[i] = rxByte(rxd);
    }


    printstrln("rxDone");
    for (int i = 0; i < NUM_BYTES; i++) {
        printheXln(bytes[i]);
    }
}

char rxByte(in port rxd)
{
    unsigned byte, time;

    // Wait for start bit
    rxd when pinseq (0) :> void @ time;
    time += BIT_TIME / 2;

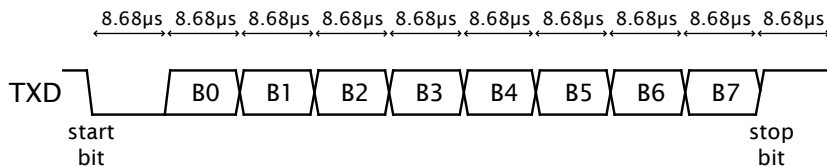
    // Input data bits
```

```
for (int i = 0; i < 8; i++) {  
    time += BIT_TIME;  
    rxd @ time :> >> byte;  
}  
  
// Input stop bit  
time += BIT_TIME;  
rxd @ time :> void;  
  
return (byte >> 24);  
}
```

Copy and paste the code into your project, and then choose **File > Save**  to save your changes to file.


2.3 Examine the code

The source code contains a transmitter thread and receiver thread running concurrently. The UART is configured to operate at a data rate of 115200 bits/s. The transmitter outputs a byte (0b00110101) to a 1-bit port and inputs a byte from another 1-bit port.

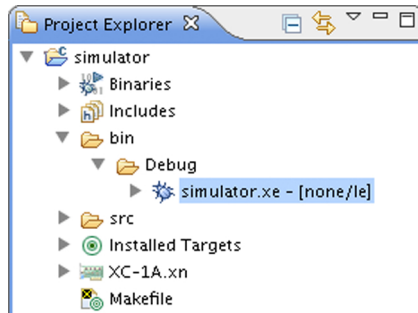


2.4 Build and run your project

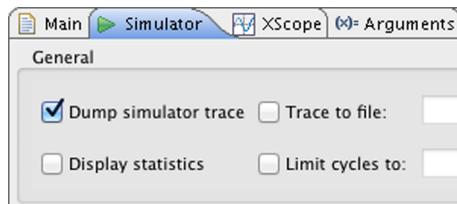
To build and run your project, follow these steps:

1. In the **Project Explorer**, click your project to select it, and then choose the menu option **Project > Build Project** .

xTIMEcomposer displays its progress in the **Console**. When the build is complete, xTIMEcomposer adds the compiled binary file to the application subfolder bin/Debug.



2. Choose **Run > Run Configurations**.
3. In the **Run Configurations** dialog, in the left panel, double-click **XCore Application**.
4. In the right panel, in **Name**, enter the name `simulate`.
5. In **Project**, ensure that your project is displayed. If not, click **Browse** to open the **Project Selection** dialog, select your project, and then click **OK**.
6. In **Device options**, in **Run on**, select the option **simulator**.
7. Select the *Simulator* tab.
8. Select the **Dump simulator trace** option.



9. Click **Run** to save your configuration and run it.
xTIMEcomposer loads the application binary into the simulator, displaying its progress in the **Console**.
For details of the output format see the xTIMEcomposer User Guide¹.

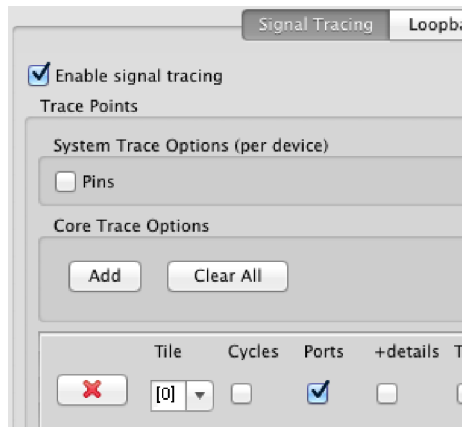
3 Output a VCD trace file

The Simulator can output a VCD file that you can then view using the integrated waveform viewer.

¹<http://www.xmos.com/published/tools-user-guide?version=latest>

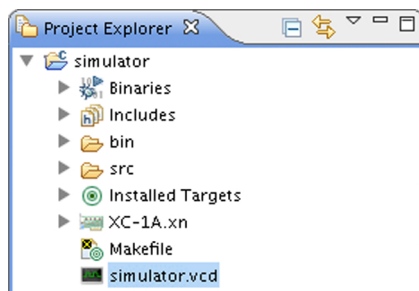
3.1 Output a VCD file

1. Select **Run > Run Configurations** and select the configuration you created previously.
2. Select the *Simulator* tab.
3. Click **Enable signal tracing**.



4. Click **Add**.
5. Select the **Ports** checkbox. Leave the **Tile** option set to **tile[0]**.
6. Click **Run**.


A VCD file is created by the simulator and appears in the *Project Explorer*.




3.2 Display the I/O signals generated by the program

1. Double-click the **VCD file** in the Project Explorer.

Two additional views are opened in the C/XC perspective: *Waves* and *Signals*. The *Signals* view contains a list of all the ports and signals for the target device.

This defaults to a hierarchical view of the traced signals, but can be changed to a flat view by clicking on the **Flat/Hierarchical** button  on the *Signals* view toolbar.

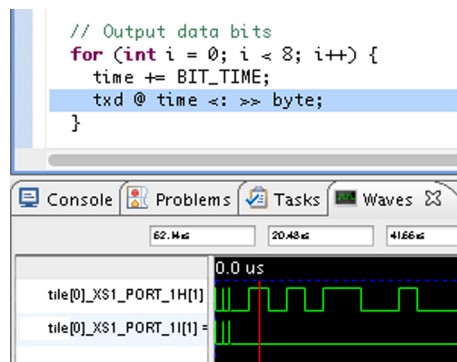
2. Expand the **Ports** node in the *Signals* tree.
3. Expand the **XS1_PORT_1H** node and drag the **tile[0]_XS1_PORT_1H (PORT_UART_TX)** signal into the *Waves* view.
4. Do the same for the **tile[0]_XS1_PORT_1I(PORT_UART_RX)** signal.
5. Click the **Zoom Fit** icon  in the *Waves* view toolbar to view the full waveforms.
The trace for PORT_1H shows that its pin starts high initially and is driven low when the program is first executed. It waits for the bit time to elapse (calculated by dividing the XS1 timer by the UART bit rate) then outputs the least significant bit of data, waits for the bit time to elapse again and then outputs the next bit. When the last bit has been output the signal is driven high to indicate the stop bit.
6. Move the mouse over the *Waves* view. The current position of the cursor is displayed (in nanoseconds) in the left numerical control at the top of the window.
7. Click in the *Waves* view. The red marker is drawn at the position the mouse is clicked. Its position is displayed (in nanoseconds) in the center numerical control at the top of the window.


4 Relate I/O signals to source code

This part of the tutorial shows you how to relate signals in the *Waves* view to the source code.

4.1 Identify the output statements in the source code

1. Move the cursor over the *PORT_1H* wave. As you move the cursor along the signal it changes to a pointing hand, indicating that the signal corresponds to an output statement in the source code.
2. Double-click the **PORT_1H** signal after it is first driven low.
The output statement `txd <: 0 @ time;` in the function `txByte` is highlighted in the editor indicating that it is responsible for the initial transition from high to low.
3. Double-click the **PORT_1H** signal after it has been driven high. The output statement `txd @ time <: >> byte;` is now highlighted, indicating that it is responsible for the value driven on the pin at this time.

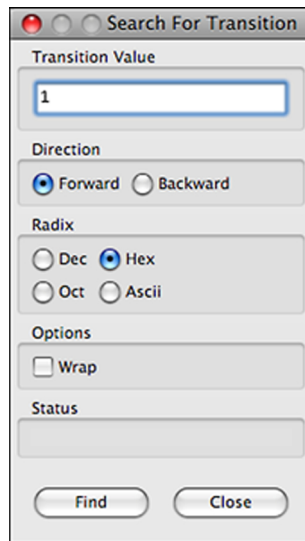


4. Select the *PORT_1H* signal in the *Waves* signal list and click the waveform at the start of the signal to set the marker.
5. Click the **Next** icon  in the *Waves* toolbar to move to the next low/high transition.
The code responsible for the transition is highlighted in the *Editing* view.
6. Click the **Next** icon several times to move along the signal highlighting the code responsible for each transition as you move.

4.2 Identify a specific signal transition

The *Wave* view lets you identify specific transitions along the signal.

1. Right-click on the **PORT_1H** signal in the *Waves* signal list and select *Search for transition*.
2. Enter 1 in the *Transition Value* and click **Find**.



The cursor jumps to the first low/high transition.

3. Click **Find** again. The cursor goes to the next low/high transition.

4. Change the *Transition Value* to 0 and click **Find**.

The cursor moves to the next high/low transition.

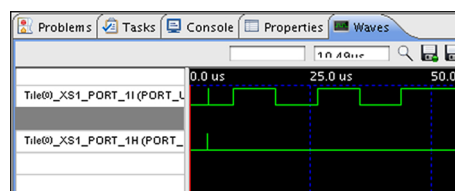
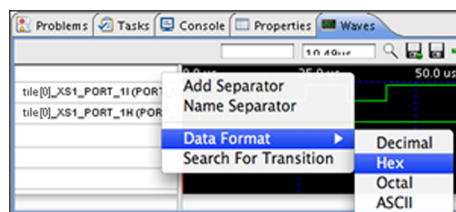
5. Change the *Transition Value* back to 1 and click **Find**.

The cursor continues to search forward from the current position until it gets to the final low/high transition.

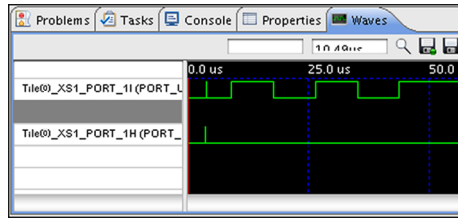
4.3 Organise the signals and display format

You can organise the signals in the Waves view and change the display format, to fit your requirements.

1. Right-click on the **PORT_1H** signal in the Waves window and select **Data Format** from the pop-up list.
2. Select the **Ascii** display value.




3. Right-click on the **PORT_1I** signal in the list and select **Add Separator** from the pop-up list.



4. Enter a name for the separator (for example *UART Transmit*) and click **OK**.
A blank grey separator is added above the signal.
5. Click on the separator and drag it to the top of the signal list.
6. Click-and-drag the **PORT_1H** signal so that it is below **PORT_1I**, which is below the separator.

4.4 Save the Wave window configuration

Once you have organised the signals and separator, you can save the configuration to a Settings file.

1. Click the **Write Session File** icon  in the *Wave* window toolbar.
2. Enter a name and location for the file and click **Save**.
Your settings are saved to the workspace.

You can reload the configuration at a future time using the **Load Settings** icon



5 Connect ports in a loopback

To verify that a value is passed correctly from one pin to another (in this example you can check that the byte 0b00110101 output from 1H is input to 1I) you can connect the pins together in a loopback. The link is specified as a pair of pins, one for each end of the connection.

5.1 Set up a loopback


1. Open the Run Configuration.
2. Select the *Simulator* tab, and the *Loopback* tab in the *Plugins* box.
3. Select **Enable pin connections**.
4. Click **Add**.

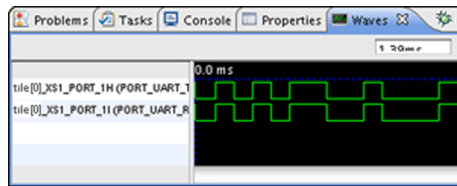
5. Select the properties for the start and end nodes.

	Node	Tile	Port	Offset	Width
from		Tile[0]	XS1_PORT_1H	0	1
to		Tile[0]	XS1_PORT_1I	0	1

6. Click **Apply** and **Run**.

5.2 Reload the wave signals

1. Select the *Signals* view in the *C/XC perspective*.
2. Click the **Reload** icon  in the top right corner of the window to reload the current VCD file.
3. The *Waves* view now shows that as the byte data is output from **PORT_1H**, it is input to **PORT_1I** at the same time.



6 What to read next

Congratulations you have now finished this introduction to the XMOS Simulator. For more information on using the simulator please see the xTIMEcomposer User Guide².

We also recommend that you try the xTIMEcomposer Studio tutorial, which shows how to create a project using xSOFTip and use it in the Simulator. See Help > Tutorials > xTIME Composer Tutorial³.



Copyright © 2012, 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.

²<http://www.xmos.com/published/tools-user-guide?version=latest>

³<http://www.xmos.com/published/xtimecomposer-studio-tutorial?automate=OpenInNewTab&tabname=xTutorial>