Application Note: AN00192

Getting Started with Timing Analysis in xTIMEcomposer Studio

The XMOS architecture has predictable timing, which allows many interfaces to be performed in software. This application note shows how to get started with timing analysis using the xTIMEcomposer studio. It shows you how to add timing constraints to the parts of your program that must run within strict time limits, enabling correct real-time behavior to be validated for your target device at compile-time.

To get started, simply double click on *Getting Started with timing analysis in xTIMEcomposer Studio* in the Examples view, and click finish in the resulting import dialog. The sample project will then be imported and you will be switched to the XMOS edit perspective. The getting started pdf is then accessable from the *doc/pdf* folder at the top level of the imported project.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0

Required hardware

None

Prerequisites

None

1 Overview

1.1 Introduction

Our comprehensive development tools suite provides everything you need to write, debug and test applications based on xCORE multicore microcontrollers. The full xTIMEcomposer tool set includes unique capabilities such as the xSCOPE logic analyzer and XMOS Timing Analyzer, that let you get the best performance from the deterministic xCORE architecture. With our collection of libraries and examples, it's easy to create and deliver xCORE applications.

xTIMEcomposer features:

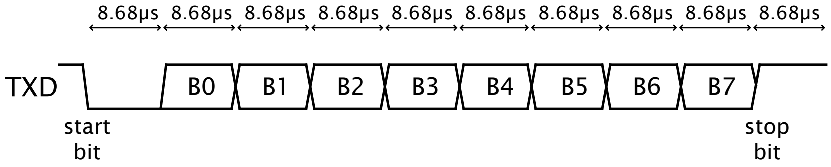
- Eclipse graphical environment + plus command line tools
- LLVM C, C++ and xC compilers
- xDEBUG: GDB multicore debugger
- xSIM: Cycle accurate simulator
- xSCOPE: In-circuit instrumentation + real-time logic analyzer
- XTA: Static timing analysis
- Multiple platform support: Windows, OS X, Linux
- Enterprise/Community editions: Tools support for everyone

The XMOS architecture has predictable timing, which allows many interfaces to be performed in software. This application note shows how to get started with timing analysis using the xTIMEcomposer studio. It shows you how to add timing constraints to the parts of your program that must run within strict time limits, enabling correct real-time behavior to be validated for your target device at compile-time.

2 Examine the application code

Ensure you are in the edit perspective by clicking on the *Edit* perspective button on the left hand side toolbar. Open the file *src/main.xc* in the editor.

This application implements a UART interface. 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, and the transmitter outputs bytes of data as shown in the diagram below.



The quiescent state of the wire is high.

A byte is sent by first driving a **start bit** (0), followed by the eight **data bits** and finally a **stop bit** (1). A rate of 115200 bits/s means that each bit must be driven for a period of 1/115200 = 8.68us.

The transmitter source code has the following structure:

```
TX ->
TX-BYTE ->
  Output start bit     (Endpoint A)
  Loop-8
    Output data bit    (Endpoint B)
  Output stop bit      (Endpoint C)
  Wait for bit period  (Endpoint D)
```

When compiled and run on a target device, the time taken to execute the code between the following pairs of endpoints must always be less than 8.68us:

- Route 1: From Endpoint A to Endpoint B (from the start bit to data bit)
- Route 2: From Endpoint B to Endpoint B (between consecutive data bits)
- Route 3: From Endpoint B to Endpoint C (from the last data bit to the stop bit)
- Route 4: From Endpoint C to Endpoint D (holding the stop bit for the bit period)
- Route 5: Between successive calls to the function txByte

If the constraint on the fifth route is not met, individual bytes will be transmitted correctly, but the UART will operate at a lower data rate.

More generally, a route consists of the set of all paths through which control can flow between two endpoints. Each route has a worst-case time, in which branches always follow the path that takes the longest time to execute. This time must satisfy the constraint for the program to be guaranteed correct under all possible executions.

3 Validate timing constraints interactively

3.1 Build the application

To build the application, select 'Project -> Build Project' in the menu, or click the *Build* button on the toolbar. The output from the compilation process will be visible on the console.

3.2 Load the binary into the timing analysis perspective

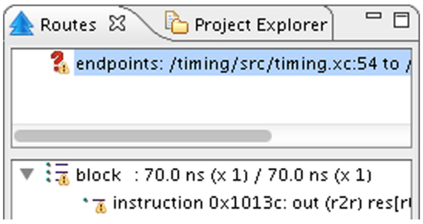
In the *project explorer* view, expand the *Binaries* node and double click on the binary (*.xe) file. This will load the binary into the analysis tools and switch perspective to the most recently used tool. For the purposes of this tutorial, ensure that the *Analyze Timing* button in the toolbar is selected.

Alternatively, click on the *Analyze* button in the left-hand toolbar, then ensure that the *Analyze Timing* button in the toolbar is selected. The *Load binary into XTA* toolbar button can then be used to load the required binary.

3.3 Validate Route 1

To check that the first route meets its timing constraint, follow these steps:

1. In the editor, locate the first output statement (Endpoint A), right-click on the endpoint marker in the left margin to bring up a menu, and choose **Set from endpoint.** A green dot 🟩 is displayed in the top-right quarter of the marker.
2. Locate the second output statement (Endpoint B), right-click on its endpoint marker and choose **Set to endpoint.** A red dot is 🟥 displayed in the bottom-right quarter of the marker.
3. Click the Analyze Endpoints button in the main toolbar. The timing analyzer identifies a single path between Endpoints A and B, which it times. The analyzed route is added to the *Routes list* in the top panel of the *Routes* view. The bottom panel of the *Routes* view shows a textual representation of the structure and timing for the selected route. The *Structure* panel in the *Visualizations view* shows a graphical visualization of the structure and timing for the selected route. In the *Routes list*, a red question mark icon ? is displayed next to the route name, indicating that no timing constraint is set.



endpoints: /timing/src/timing.xc:54 to /

block : 70.0 ns (x 1) / 70.0 ns (x 1)

instruction 0x1013c: out (r2r) res[r

4. Right-click on the route in the *Routes list* to bring up a menu and choose **Set timing requirement** to open the **Requirement box.**
5. Enter a value of 8.68, select the unit *us* and click **OK.** The red question mark is replaced by a green tick ✓, indicating that the route meets the specified timing constraint.
6. Hover over the route to view information such as the number of paths and worst-case timing.

3.4 Validate Route 2

To check that the second route meets its timing constraint, follow these steps:

1. Right-click on the marker for Endpoint B to bring up a menu, and choose **Set from endpoint.** The warning triangle generated by the previous Analyze endpoints action obscures the endpoint marker, but you can still set a new endpoint by right-clicking on the marker. This statement is now set as both the *from* and *to* endpoints.

2. Click Analyze Endpoints E. The timing analyzer identifies the path around the loop. However, in this case, as optimisations are enabled, the compiler has unrolled the loop. For this reason, 8 routes are created, each corresponding to an iteration of the loop. However, as the final route corresponds to the path that attempts to exit the loop and then later re-enter it, this can be ignored. Simply right click on this route and select *delete*.
3. In the *Routes view*, select all the routes and set a timing constraint of 8.68us on each. The status of the routes are updated, indicating that the routes now all meet thier timing constraints.

3.5 Validate Routes 3 and 4

Use the techniques introduced in the previous two sections to check that Route 3 (Endpoint B to Endpoint C) and Route 4 (Endpoint C to Endpoint D) meet their timing constraints. Once completed, a total of four routes should be displayed in the **Routes list**.

3.6 Validate Route 5

To check that route 5 meets its timing constraint (between consecutive byte transmissions), follow these steps:

1. Set the endpoints for a route from Endpoint D to A.
2. In the function `txBytes`, right-click on the endpoint marker following the loop and choose **Add to exclusion list**. The timing analyzer will not attempt to analyze paths outside of the loop.
3. Click Analyze Endpoints E in the main toolbar to create the route.
4. In the *Routes view*, set a timing constraint of 8.68us. The status of the route is updated, indicating that the route meets its timing constraint. You should have a total of five routes in the *Routes list*.

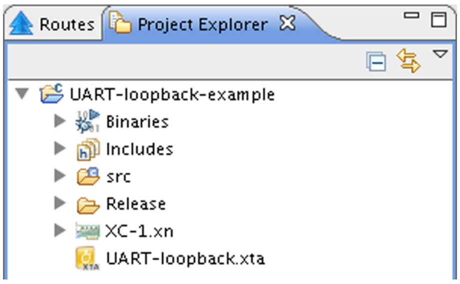
4 Validate the timing constraints during compilation

Having checked that all of the UART timing constraints are met, you can create a script that performs the same checks every time the program is compiled.

4.1 Generate a timing script

To create a timing script and update the source file with the pragmas required to make the script portable, follow these steps:

1. Click **Generate Script** button in the main toolbar. The *Script Options* dialog opens.
2. Enter a name for the script (with a .xta extension) in the **Script location** text box.
3. To change the names of the pragmas added to the source file, click the values in the **Pragma name** fields and edit.
4. Click **OK** to save the script and update your source code. xTIMEcomposer Studio adds the script to your project and opens it in the editor.



The next time you compile your program, the timing constraints are checked and any failures are reported as compilation errors.

5 References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

6 Full source code listing

6.1 Source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

#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);
}
```

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.