Application Note: AN00200

Getting Started with the Task Viewer in xTIMEcomposer Studio

This application note shows how to get started with the task viewer in the xTIMEcomposer studio. It shows you how to view the task graph for a given source file, modify the source code and see the change reflected in the updated task graph.

To get started, simply double click on *Getting Started with the task viewer 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

This application note shows how to get started with the task viewer in the xTIMEcomposer studio. It shows you how to view the task graph for a given source file, modify the source code and see the change reflected in the updated task graph.

2 Getting Started

Ensure you are in the edit perspective by clicking on the ‘Edit’ perspective button on the left hand side toolbar.

2.1 View the source

In the *Project Explorer* view you will see the *app_getting_started_with_the_task_viewer* project. Open *src/main.xc* by double clicking.

This application contains 2 tasks, a server task and a client task. They are connected via an interface which defines a single function *func1*, which accepts a single integer parameter. The client task calls this function passing ‘5’ as the parameter, and the server tasks implementation simply prints this to the console.

2.2 View the task graph

Select the *Task Viewer* view which by default is located in the bottom pane. This view shows a pictorial representation of the tasks in the source file currently being edited, and the connections between them. As you can see, in this case, both tasks run on the same tile, each running on its own logical core.

2.3 Task Viewer Naviagtion

Zooming can be achieved via the mouse wheel or the zoom buttons in the views toolbar, and the portion of the task graph being viewed can be altered via left-click and dragging. Tile/core placement information can be toggled on/off via the *Show/Hide Task Placement* button in the views toolbar.

2.4 Update the application code

Now uncomment the ‘[[distributable]]’ attribute above the *serverTask* definition. Save the file via the *Save* button on the main toolbar, or via the Ctrl-S shortcut. As you can see, the task graph has now been updated to show the fact that *serverTask* has been distributed into the client and is no longer placed on it’s own logical core.

3 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

4 Full source code listing

4.1 Source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved

#include <print.h>

interface TestInterface {
  void func1(int param);
};

void clientTask(client interface TestInterface iface) {
  iface.func1(5);
}

void serverTask(server interface TestInterface iface){
  unsigned int complete = 0;
  while (!complete) {
    select {
      case iface.func1(int param):
        printintln(param);
        complete = 1;
        break;
    }
  }
}

int main() {
  interface TestInterface iface;
  par {
    clientTask(iface);
    serverTask(iface);
  }
  return 0;
}
```