Application Note: AN00239

Using the logging library

The logging library provides the debug_printf() function which is a lightweight implementation of printf(). It also provides a framework for compile-time control of which debug messages are enabled.

Required tools and libraries

The code in this application note is known to work on version 14.2.3 of the xTIMEcomposer tools suite, it may work on other versions.

The application depends on the following libraries:

- lib_logging (>=2.1.0)

Required hardware

The example code provided with the application has been implemented and tested on the xCORE-200 explorerKIT.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For a description of XMOS related terms found in this document please see the XMOS Glossary[1].

[1] http://www.xmos.com/published/glossary

1 Overview

1.1 Introduction

This application note shows how to use the logging library. It covers the difference between the logging library (debug_printf()) and the system library printing function (printf()).

It also covers the difference between JTAG and xSCOPE to perform the I/O to the host, including approximate values for resource usage and performance of each approach.

2 Example logging library usage

2.1 The Makefile

The logging library needs to be added to the USED_MODULES line of your Makefile:

```
USED_MODULES = .. lib_logging ..
```

Also, debug_printf() calls are only active if you enable them in your Makefile. This is done by by setting DEBUG_PRINT_ENABLE to 1 in the XCC_FLAGS:

```
XCC_FLAGS += -DDEBUG_PRINT_ENABLE=1
```

2.2 Include

The function prototypes are declared in a single header file which must be included from your source file:

```
#include "debug_print.h"
```

2.3 Using the logging library

The logging library provides the debug_printf() function. The function has the same form as a standard printf(), but only supports a subset of formats - namely the %d, %x, %s, %u, %p and %c format specifiers with no conversions.

The example application outputs “Hello world”.

```

debug_printf("Hello world\n");
```

2.4 Building and Running the application using xTIMEcomposer

Once the application source code is imported into the tools it can be built by pressing the **Build** button in xTIMEcomposer. This will create the AN00239.xe binary in the bin folder of the project.

A *Run Configuration* then needs to be set up. This can be done by selecting the **Run** ▶ **Run Configurations...** menu. You can create a new run configuration by right clicking on the **xCORE application** group in the left hand pane and **New**.

Select your XMOS XTAG in the Target box and click **Apply**. Ensure that the **Run Configurations** ▶ **Target I/O** ▶ **xSCOPE via xCONNECT...** option is set. You can now click the **Run** button to launch the application.

2.5 Building and Running the application using the command line

First open an command prompt/terminal window with the tools environment setup. A setup batch/script file is provided in the xTIMEcomposer package to do this for you.

Building is done by changing to the folder containing the Makefile and src sub-folder and doing a call to xmake. Running the application is then done using the command: `xrun --xscope bin/AN00239.xe`.

2.6 Debug units

Applications can be created with different *units* whose debug output is independently controlled. The example application also calls a function in another unit:

```

unit_function();
```

That file has put its debug messages as a separate debug unit by doing:

```
#define DEBUG_UNIT unit
#include "debug_print.h"
```

And by default these debug messages are not enabled, so running the program will only produce the following output:

```
$ xrun --xscope bin/AN00239.xe
Hello world
```

In order to enable the debug_print messages in `unit.xc` it is necessary to add the following to the Makefile (uncomment the line):

```
XCC_FLAGS += -DDEBUG_PRINT_ENABLE_unit=1
```

After rebuilding the application it will now produce:

```
$ xrun --xscope bin/AN00239.xe
Hello world
Unit print
```

2.7 xSCOPE printing

On the xCORE platform it is possible to have I/O messages sent to the console using either JTAG or xSCOPE. The default is for JTAG to be used. However, when doing I/O over JTAG all cores on the xCORE are stopped and hence real-time functionality is no longer maintained. As a result xSCOPE I/O should be preferred.

xSCOPE I/O is enabled by creating a `config.xscope` file. This file can be created in the same folder as the Makefile or with the source files. When `config.xscope` exists, it controls whether I/O messages are enabled, and whether they use xSCOPE or JTAG. A basic file which enables I/O over xSCOPE contains:

```
<xSCOPEconfig ioMode="basic" enabled="true">
</xSCOPEconfig>
```

3 Notes

3.1 Resource usage

The following table shows the memory and cycle requirements for doing a simple print of "Hello world %d\n", using either `printf()` or debug_`printf()`, and using either JTAG or xSCOPE as the transport to the host.

Function	Transport	Program Memory (kB)	Time (us)	Channel Ends
None	N/A	0.9	0.0	0
debug_printf()	JTAG	1.86	72000	0
debug_printf()	xSCOPE	2.88	8.5	1 per tile
printf()	JTAG	9.02	72000	0
printf()	xSCOPE	9.99	18.6	1 per tile

Table 1: Resource usage

The JTAG timings are approximate and will depend on a number of factors including the host machine being used.

3.2 Lossless vs lossy

The advantage of using xSCOPE instead of JTAG should be clear from the performance figures above. However, it is important to understand that xSCOPE is a lossy transport and as such if too much I/O is created then messages can be lost.

3.3 Ordering

It is also essential to understand that messages produced by different logical cores are interleaved on the host console. There is no guarantee of the order in which they will be printed.

3.4 print.h

The tools provide extremely lightweight printing functions in `print.h`. For example it is possible to do:

```
#include <print.h>
...
printstr("The number ");
printint(10);
printstrln(" should be 10");
```

But each of these print fragments can arrive mixed with messages from other logical cores. Whereas:

```
debug_printf("The number %d should be 10\n", 10);
```

will be printed as a single line.

4 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

5 Full Source code listing

5.1 Source code for main.xc

```
// Copyright (c) 2014-2016, XMOS Ltd, All rights reserved

#include "debug_print.h"

void unit_function();

int main() {

    debug_printf("Hello world\n");

    unit_function();

    return 0;
}
```

5.2 Source code for unit.xc

```
// Copyright (c) 2014-2016, XMOS Ltd, All rights reserved

#define DEBUG_UNIT unit
#include "debug_print.h"

void unit_function()
{
    debug_printf("Unit print\n");
}
```