Application Note: AN00214

# Using memory for debug message logging

This application shows how to implement a simple debug logging system to memory which when used in conjuction with the XMOS debugger will allow you to provide a printing mechanism which can be used in systems where it is not possible to have a debugger connected permenantly, for example when debugging boot from flash or issues around a device reset.

The example provided uses the command line xgdb debugger to provide a way of attaching to a running system and dumping the contents of the debug log to the terminal. In order to capture print messages from the application the code demonstrates how to override the functionality provided in the XMOS standard libraries for printing to provide a custom application specific implementation.

## Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0

## Required hardware

This application note is designed to run on any XMOS multicore microcontroller or the XMOS simulator.

## 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 demonstrates how to to override the standard library function used by the XMOS tools for debug printing in order to redirect output to a circular memory buffer on the device. This allows a debug logging system to be created which persists within the device to allow debugging of issues such as flash booting and device reset handling.

# 2 Getting Started

This application is designed to run from the command line due to using advanced features of the XGDB GUI which are not exposed through the xTIMEcomposer Studio debugger interface.

The example can be downloaded through the example browser in xTIMEcomposer studio but after that you will need a command prompt set up with the XMOS development tools.

This application contains a very simple main function which loops round and outputs a debug message using the standard library printing routings every 100ms. From the code in main you can see that there is no change to any of the standard printing routines required at the user code level.

The code is as follows,

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <print.h>

int main() {
    unsigned int i = 0;
    timer tmr;
    unsigned int tmrvalue = 0;
    while (1) {
      delay_milliseconds(100);
      i++;
      tmr :> tmrvalue;
      printstr("Value of i = "); printuint(i);
      printstr(" at time "); printuint(tmrvalue);
      printstr("\n");
    }
    return 0;
}
```

The code contained in the *main()* function is very simple, it has a while loop which executes a simple routine. A library function is used to cause a delay of 100ms then a message is output using the standard print functions. The message contains a timer value read from the device and a variable which is incremented in the code.

The file *src/debug_memory_log_buffer.c* contains the code to override the mechanism in the standard XMOS libraries used for printing.

The code looks as follows,

```
#define DEBUG_MEMORY_LOG_ENABLED 1
#ifdef DEBUG_MEMORY_LOG_ENABLED

#include <stdlib.h>

unsigned int debug_memory_log_buffer_index = 0;
#define DEBUG_MEMORY_LOG_BUFFER_SIZE 2048
unsigned char debug_memory_log_buffer[DEBUG_MEMORY_LOG_BUFFER_SIZE];

// Override the weak symbol used for print messages
int _write(int fd, const unsigned char *data, size_t len) {

  // Check for wrap of the circular buffer
  if ((debug_memory_log_buffer_index + len + 1) > DEBUG_MEMORY_LOG_BUFFER_SIZE)
    debug_memory_log_buffer_index = 0;

  // Copy write message into log buffer
  for(unsigned int i = 0; i < len; i++) {
    debug_memory_log_buffer[debug_memory_log_buffer_index] = data[i];
    debug_memory_log_buffer_index++;
  }

  // Terminate the whole buffer after the current message
  debug_memory_log_buffer[debug_memory_log_buffer_index] = '\0';

  return len;
}

#endif
```

The code to override the *_write* library call is contained within this file, this symbol is declared as a weak symbol within user application code so a custom implementation can be provided.

The code declares an array which is used to store the debug messages passed from the user application to the *_write* function. There is also a define to declare the size of the array and a variable used for keeping track of the state of the circular buffer.

Inside the *_write* function there is an implementation of a simple circular buffer which is written to with the data passed in as arguments to the *_write* function.

## 2.1 Build the application

To build the application, in the console set up to use the XMOS development tools change to the top level of the application note directory and run the command *xmake all*

The defauly target is set to the XMOS startKIT, if you require this to run on another board then change the target and rebuild.

This will build the application and generate the binary for the application.

## 2.2 Launching the example in XGDB

The XGDB debugger has a very simple but powerful command line interface that we can use to print out the messages that are stored by our application in memory. Firstly we need to connect to the target via the XGDB command line:

```
> xgdb bin/app_using_memory_for_debug.xe
```

Connect to the development board connected to the development machine, in this case there is an XMOS startKIT attached:

```
(gdb) connect
0x00010000 in _start ()
```

Now run the application using the run command in XGDB:

```
(gdb) run
Loading setup image to XCore 0
Loading section .text, size 0xfc lma 0x10000
Loading section .cp.rodata, size 0x18 lma 0x100fc
Loading section .dp.data, size 0x8 lma 0x10114
Start address 0x10000, load size 284
Transfer rate: 69 KB/sec, 94 bytes/write.
First stage multi-node boot started
First stage multi-node boot completed
Loading application image to XCore 0
Loading section .text, size 0x5d2 lma 0x10000
Loading section .init, size 0x1a lma 0x105d2
Loading section .fini, size 0x2e lma 0x105ec
Loading section .cp.rodata, size 0x20 lma 0x1061c
Loading section .cp.const4, size 0x24 lma 0x1063c
Loading section .cp.rodata.cst4, size 0x18 lma 0x10660
Loading section .cp.rodata.string, size 0x11 lma 0x10678
Loading section .dp.data, size 0x20 lma 0x1068c
Start address 0x10000, load size 1703
Transfer rate: 87 KB/sec, 212 bytes/write.
```

Let the application run for a few seconds and then enter a ctrl-c command into the console to interrupt the application:

```
Program received signal SIGINT, Interrupt.
0x00010342 in delay_ticks_longlong ()
```

The *tile apply all* command can now be used to pass a command to every tile in the system, in this case we are using the command *printf "%s", debug_memory_log_buffer* which will display the contents of our target memory debug log:

```
(gdb) tile apply all printf "%s", debug_memory_log_buffer

Thread 1 (tile[0] core[0]):
Value of i = 1 at time 22889918
Value of i = 2 at time 32890814
Value of i = 3 at time 42891710
Value of i = 4 at time 52892606
Value of i = 5 at time 62893502
Value of i = 6 at time 72894398
Value of i = 7 at time 82895294
Value of i = 8 at time 92896190
Value of i = 9 at time 102897086
Value of i = 10 at time 112898015
Value of i = 11 at time 122898988
Value of i = 12 at time 132899961
Value of i = 13 at time 142900934
Value of i = 14 at time 152901907
Value of i = 15 at time 162902880
Value of i = 16 at time 172903853
Value of i = 17 at time 182904826
Value of i = 18 at time 192905799
Value of i = 19 at time 202906772
Value of i = 20 at time 212907745
Value of i = 21 at time 222908718
Value of i = 22 at time 232909691
Value of i = 23 at time 242910664
Value of i = 24 at time 252911637
Value of i = 25 at time 262912610
Value of i = 26 at time 272913583
Value of i = 27 at time 282914556
(gdb)
```

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

int main() {
    unsigned int i = 0;
    timer tmr;
    unsigned int tmrvalue = 0;
    while (1) {
      delay_milliseconds(100);
      i++;
      tmr :> tmrvalue;
      printstr("Value of i = "); printuint(i);
      printstr(" at time "); printuint(tmrvalue);
      printstr("\n");
    }
    return 0;
}
```

## 4.2 Source code for debug_memory_log_buffer.c

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

#define DEBUG_MEMORY_LOG_ENABLED 1
#ifdef DEBUG_MEMORY_LOG_ENABLED

#include <stdlib.h>

unsigned int debug_memory_log_buffer_index = 0;
#define DEBUG_MEMORY_LOG_BUFFER_SIZE 2048
unsigned char debug_memory_log_buffer[DEBUG_MEMORY_LOG_BUFFER_SIZE];

// Override the weak symbol used for print messages
int _write(int fd, const unsigned char *data, size_t len) {

  // Check for wrap of the circular buffer
  if ((debug_memory_log_buffer_index + len + 1) > DEBUG_MEMORY_LOG_BUFFER_SIZE)
    debug_memory_log_buffer_index = 0;

  // Copy write message into log buffer
  for(unsigned int i = 0; i < len; i++) {
    debug_memory_log_buffer[debug_memory_log_buffer_index] = data[i];
    debug_memory_log_buffer_index++;
  }

  // Terminate the whole buffer after the current message
  debug_memory_log_buffer[debug_memory_log_buffer_index] = '\0';

  return len;
}

#endif
```