Application Note: AN00147

xCORE-XA - Debug printing via SWO

This application note shows how to create a simple example which targets the XMOS xCORE-XA device and demonstrates how to build and run this application using the XMOS development tools.

The code associated with this application note provides an example of setting up and using debug printing via the ARM core serial wire output (SWO) interface and connecting this to the debug tools running on the host PC.

This example shows how to develop code targeting the xCORE-XA, how to use the XMOS development tools to compile and build applications and how to deploy code onto an xCORE-XA device using the supported development adapters.

The example code in this application note does not communicate between the xCORE tile and the ARM code in order to introduce the mechanism used to provide debug print messages from applications running on the ARM core.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2.0

Required hardware

This application note is designed to run on an XMOS xCORE-XA series device.

The example code provided with this application note has been implemented and tested on the xCORE-XA core module board but there is no dependancy on this i board and it can be modified to run on any development board which uses an xCORE-XA series device.

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 descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- The XMOS tools manual contains information regarding the use of xCORE-XA devices[2].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xtimecomposer-user-guide

1 Overview

1.1 Introduction

xCORE-eXtended Architecture (the XA Family) combines multicore microcontroller technology with an ultra-low-power ARM Cortex-M3 processor, to create the next wave in programmable system-on-chip (SoC) products.

The xCORE-XA architecture allows embedded system designers to use high-level software to configure a device with the exact set of interfaces and peripherals needed for their design, while re-using existing ARM binary code and standard library functions, and taking advantage of ultra-low energy fixed-function peripherals. Designers can also add real-time data-plane plus control processing and DSP blocks, using multiple xCORE processor cores, with the ARM available to run control plane processing software such as communication protocol stacks, standard graphics libraries, or complex monitoring systems.

SWO is a dedicated pin of ARM's Cortex-M debug interface. While it can be used to output various information in real time via the CPU, its main use is to handle terminal output in real-time with very low intrusion.

This permits most programs to perform debug outputs via a terminal without losing the real-time behavior of an application.

1.2 Block diagram

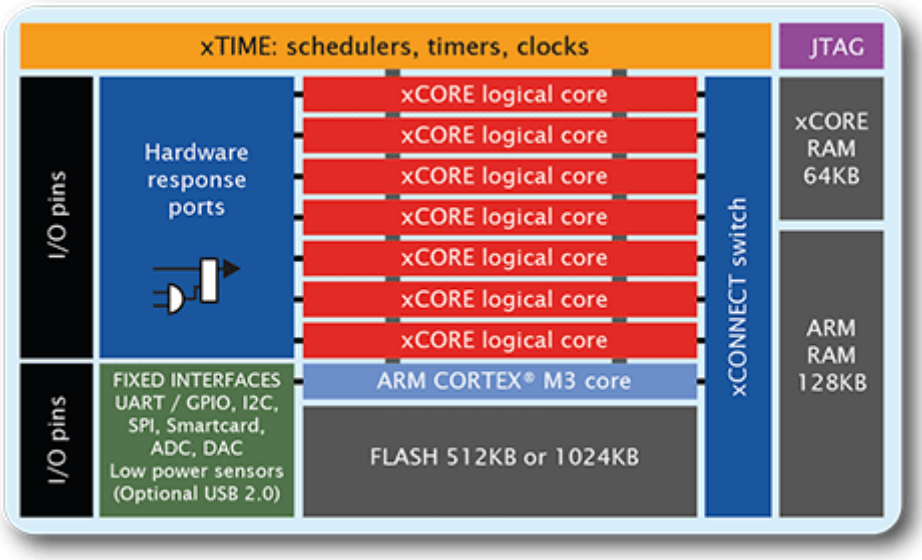

Figure 1: Block diagram of XMOS xCORE-XA microcontroller

2 xCORE-XA - Debug printing via SWO

The example in this document does not have any external dependancies on application libraries other than those supplied with the XMOS development tools. It demonstrates how to use the SWO output pin on the ARM core of an xCORE-XA device to implement debug printing from a running application and capture the output on the host machine.

This example is implemented using 2 tasks which are partitioned so that one executes on the ARM core and one on the xCORE tile.

The tasks perform the following operations.

- A task executing a loop and printing out a series of values via SWO
- A task executing an empty main function on the xCORE tile

This can be seen in the following task diagram.

Figure 2: Task diagram of xCORE-XA SWO debug printing example

2.1 Declaring resource and setting up the ARM core and xCORE tile

The example code in this application note is split into 2 files containing the implementation of a simple main() function for both the ARM core and xCORE tile . This application executes code on the ARM core to output data via the SWO mechanism, on the xCORE tile the application simply exits from main(). There is no communication within this application so the code is free to execute without any form of synchronization.

2.2 The ARM core application main() function

The main() function for the ARM core is contained within the file `main_arm.c` and is as follows,

```
int main(void) {
  // Enable GPIO clock
  CMU_ClockEnable(cmuClock_GPIO, true);
  setup_swo_for_print();

  for (int i = 0; i < 20; i++) {
    printf("Value of i is ..... %d\r\n", i);
  }

  return 0;
}
```

Looking at this function in more detail you can see the following:

- The GPIO clock for the ARM core is explicitly enabled
- A function is called to configure the SWO mechanism
- There is a for loop which executes 20 iterations
- Inside the for loop the printf() function is used to output debug messages

2.3 Setting up ARM SWO for the application

In order to use SWO on the ARM core for debug messages it needs to be configured into the mode that is required. This is done from the function setup_swo_for_print() called from main() in the file `main_arm.c`

```
void setup_swo_for_print(void) {
  /* Enable GPIO clock. */
  CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

  /* Enable Serial wire output pin */
  GPIO->ROUTE |= GPIO_ROUTE_SWOPEN;

  /* Set location 0 */
  GPIO->ROUTE = (GPIO->ROUTE & ~(_GPIO_ROUTE_SWLOCATION_MASK)) | GPIO_ROUTE_SWLOCATION_LOC0;

  /* Enable output on pin - GPIO Port F, Pin 2 */
  GPIO->P[5].MODEL &= ~(_GPIO_P_MODEL_MODE2_MASK);
  GPIO->P[5].MODEL |= GPIO_P_MODEL_MODE2_PUSHPULL;

  /* Enable debug clock AUXHFRCO */
  CMU->OSCENCMD = CMU_OSCENCMD_AUXHFRCOEN;

  /* Wait until clock is ready */
  while (!(CMU->STATUS & CMU_STATUS_AUXHFRCORDY));

  /* Enable trace in core debug */
  CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
  ITM->LAR  = 0xC5ACCE55;
  ITM->TER  = 0x0;
  ITM->TCR  = 0x0;
  TPI->SPPR = 2;
  TPI->ACPR = 0xf;
  ITM->TPR  = 0x0;
  DWT->CTRL = 0x400003FE;
  ITM->TCR  = 0x0001000D;
  TPI->FFCR = 0x00000100;
  ITM->TER  = 0x1;
}
```

This function sets up and configures the SWO hardware for use by the application and connects it to the appropriate ARM core GPIO pins. Once this function has executed the SWO debug hardware can be used to output messages to the host machine via the debug adapter.

2.4 Sending character output to the SWO mechanism

In order to send characters to the SWO debug hardware from the application the function `ITM_SendChar()` is used. This will output a single character via SWO to the host machine. To use this function with `printf()` the C runtime library function `_write()` is provided as part of the application, this overloads the standard symbol which exists within the C runtime with the version provided by the application.

This can be seen in the following code.

```
int _write(int fd, unsigned char *buffer, unsigned int count) {
  for (int i = 0; i < count; i++) {
    ITM_SendChar(buffer[i]);
  }
  return 0;
}
```

The function _write() recieves the buffer and a size to output from printf(). In this code there is a simple loop which outputs a character at a time via SWO to the host machine. This data will be captured and reported by the development tools running on the host.

2.5 The xCORE tile application main() function

The main() function for the xCORE tile is contained within the file main_xcore.xc and is as follows,

```
int main() {
  return 0;
}
```

This function is basically a placeholder and simply returns from main().

APPENDIX A - Example Hardware Setup

This application example is designed to run on the xCORE-XA core module board. The xCORE-XA core module board should be connected to both power and have the development adapters connected to a host machine to allow program download and also SWO debug messages.

This can be seen in the following image.

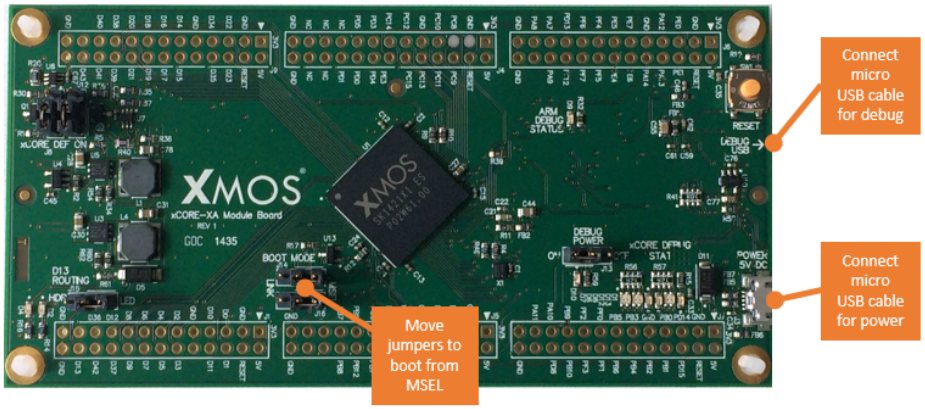

Figure 3: XMOS xCORE-XA core module board setup

The hardware should be configured as displayed above for this example:

- Both the power and debug adapter USB cables should be connected
- The boot mode of the board should have the jumpers selecting boot from msel

APPENDIX B - Launching the example application

Once the example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer Studio the application can be executed on the xCORE-XA core module board.

Once built there will be a `bin` directory within the project which contains the binary for both the ARM core and xCORE tile. The xCORE binary has a XMOS standard .xe extension.

In order to allow access to the debug interface of the ARM core the SEGGER gdb server application needs to be running. The documentation for setting up and executing this can be found in the XMOS development tools user guide and the xCORE-XA development chapter.

In addition to the standard GDB server setup required for the ARM core, in order to view the SWO output as it is generated there are additional options required to enable this. For this example the command line SEGGER gdb server is used. The script file used to enable SWO on the debug adapter is provided in the host directory of this application note.

With then SEGGER jlink software on your path and from the top level of the application note directory execute the following command to start gdb server with SWO enabled.

Windows:

```
> JLinkGDBServerCL.exe -if SWD -x host\enableswo.jlink
```

Linux:

```
> JLinkGDBServer -if SWD -x host\enableswo.jlink
```

OSX:

```
> JLinkGDBServer -if SWD -x host\enableswo.jlink
```

B.1 Connecting to the SWO server

The SWO server is exposed via a socket port on the local machine to which a connection can be made and SWO terminal output will appear. In this example we use telnet to connect to this socket port. The default gdb server socket port for terminal output is on localhost port 2333.

From another command prompt start a telnet session and connect to port 2333 on the local machine:

```
> telnet localhost 2333
```

This will produce the following output when connected correctly:

```
SEGGER J-Link GDB Server V4.80f - Terminal output channel
```

This terminal will now recieve out SWO messages from the ARM core.

B.2 Launching from the command line

From the command line the `xrun` tool is used to download code to both the ARM core and xCORE tile. In this example code is only being downloaded to the ARM core. Changing into the bin directory of the project will allow the application to be executed on the xCORE-XA microcontroller as follows:

```
> xrun app_xcore_xa_swo_print             <-- Download and execute the ARM code
```

Once this command has been executed the printf messages from the application running on the ARM core will appear on the telnet console.

B.3 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio there is the run mechanism to download code to both the ARM core and xCORE tile. By selecting the ARM binary contained within the project bin directory, right click and then run as ARM application, the application will be downloaded and executed on the ARM core. In this example only the binary for the ARM core is being downloaded.

Once this command has been executed the printf messages from the application running on the ARM core will appear on the telnet console.

B.4 Application output via SWO

After running the code on the ARM core via either of the above methods the following output will be visible on the telnet console:

```
Value of i is ..... 0
Value of i is ..... 1
Value of i is ..... 2
Value of i is ..... 3
Value of i is ..... 4
Value of i is ..... 5
Value of i is ..... 6
Value of i is ..... 7
Value of i is ..... 8
Value of i is ..... 9
Value of i is ..... 10
Value of i is ..... 11
Value of i is ..... 12
Value of i is ..... 13
Value of i is ..... 14
Value of i is ..... 15
Value of i is ..... 16
Value of i is ..... 17
Value of i is ..... 18
Value of i is ..... 19
```

APPENDIX C - 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

APPENDIX D - Full source code listing

D.1 Source code for main_xcore.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
int main() {
  return 0;
}
```

D.2 Source code for main_arm.c

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include "em_gpio.h"
#include "em_cmu.h"
#include <stdio.h>


void setup_swo_for_print(void) {
  /* Enable GPIO clock. */
  CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

  /* Enable Serial wire output pin */
  GPIO->ROUTE |= GPIO_ROUTE_SWOPEN;

  /* Set location 0 */
  GPIO->ROUTE = (GPIO->ROUTE & ~(_GPIO_ROUTE_SWLOCATION_MASK)) | GPIO_ROUTE_SWLOCATION_LOC0;

  /* Enable output on pin - GPIO Port F, Pin 2 */
  GPIO->P[5].MODEL &= ~(_GPIO_P_MODEL_MODE2_MASK);
  GPIO->P[5].MODEL |= GPIO_P_MODEL_MODE2_PUSHPULL;

  /* Enable debug clock AUXHFRCO */
  CMU->OSCENCMD = CMU_OSCENCMD_AUXHFRCOEN;

  /* Wait until clock is ready */
  while (!(CMU->STATUS & CMU_STATUS_AUXHFRCORDY));

  /* Enable trace in core debug */
  CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
  ITM->LAR  = 0xC5ACCE55;
  ITM->TER  = 0x0;
  ITM->TCR  = 0x0;
  TPI->SPPR = 2;
  TPI->ACPR = 0xf;
  ITM->TPR  = 0x0;
  DWT->CTRL = 0x400003FE;
  ITM->TCR  = 0x0001000D;
  TPI->FFCR = 0x00000100;
  ITM->TER  = 0x1;
}

int _write(int fd, unsigned char *buffer, unsigned int count) {
  for (int i = 0; i < count; i++) {
    ITM_SendChar(buffer[i]);
  }
  return 0;
}

int main(void) {
  // Enable GPIO clock
  CMU_ClockEnable(cmuClock_GPIO, true);
  setup_swo_for_print();

  for (int i = 0; i < 20; i++) {
    printf("Value of i is ..... %d\r\n", i);
  }

  return 0;
}
```