Application Note: AN00141

xCORE-XA - Application Development

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 driving a LED connected to both the ARM and xCORE GPIO.

This simple 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 tools support and project structure which allows xCORE-XA development.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2.0 or later

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 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.

1.2 Block diagram

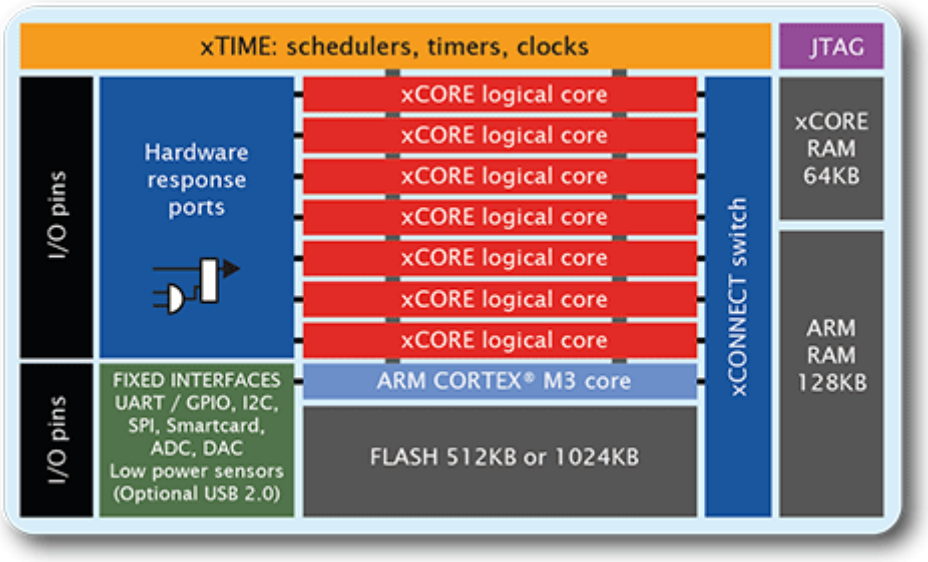

Figure 1: Block diagram of XMOS xCORE-XA microcontroller

2 xCORE-XA Application Development

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 access an LED via the GPIO of both the xCORE tile and the ARM core within the xCORE-XA device. This simple example shows the additions that are required to XMOS makefiles to build an xCORE-XA project and also how projects of this type are structured.

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 toggling a GPIO on the ARM core
- A task executing a loop and toggling a GPIO on the xCORE tile

This can be seen in the following task diagram.

Figure 2: Task diagram of xCORE-XA LED example

2.1 Source code structure for xCORE-XA projects

The XMOS development tools provide support for the compilation of code for both xCORE and ARM targets within a common environment. This allows a single development environment to be used when developing xCORE-XA application code. In order to achieve this, projects which are intended for deployment on an xCORE-XA are required to have a specific layout. This involves partitioning the code into a directory structure that seperates the code which is ARM and xCORE specific and allowing code which is common between both to be used within the project.

In the example for this application note there are 2 source files, one for the main() function of the xCORE tile `main_xcore.xc` and one for the main() function of the ARM core `main_arm.c`.

These are stored in the following directory structure:

```
Makefile                  <-- shared makefile for ARM and xCORE targets
src                       <-- top level project source directory
  arm                     <-- ARM specific application code
    main_arm.c
    ...
  xcore                   <-- xCORE specific application code
    main_xcore.xc
    ...
```

As you can see from this structure, it is clear that the code specific to either the xCORE or the ARM is stored in the target specfic directory. This allows the XMOS build system to use the correct compiler when building the application. Any code that is not within either of these directories or within another directory within the project will be compiled and used with both the ARM and xCORE targets of the xCORE-XA.

2.2 Makefile support for xCORE-XA projects

As there is only one makefile for xCORE-XA based projects there is the requirement to tell the XMOS build system that a particular project is targeting a device of this type. This is done by setting a variable in the makefile to inform the build system that this is a multi-architecture target.

Within the makefile the following variable is set to mark this project as this type:

```
XCORE_ARM_PROJECT = 1
```

It is also a requirement to be able to pass compiler and build options to the specific tool chain used to build the application code for the ARM and the xCORE. These are also makefile variables and are specified as follow in the example code for this application note.

Options passed to xCORE build tools from makefile:

```
XCC_FLAGS = -O2 -g -report
```

Options passed to ARM build tools from makefile:

```
ARM_GCC_FLAGS = -std=c99 -O0 -g
```

These variables allow a user to specify options such as the level of optimization and the linking of additional libraries when building an application.

2.3 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 simply executes a loop which flashes an LED on the development board from both the xCORE tile and ARM core in parallel. There is no communication within this application so the LED flashing code is free to execute without any form of synchronization.

2.4 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_gpio_led();

  set_led_off();

  while (1) {
      for (unsigned int i = 0; i < 250000; i++) {}
          set_led_on();
      for (unsigned int i = 0; i < 250000; i++) {}
          set_led_off();
  }

  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 GPIO ports being used
- There is a while loop which executes forever
- Inside the while loop an LED is toggled on and off at a specific rate
- A for loop is used to add a delay between toggling the LED on and off

2.5 Setting up ARM GPIO for the application

In order to use a GPIO on the ARM core for toggling an LED it needs to be configured into the mode that is required. This is done from the function setup_gpio_led() called from main() in the file `main_arm.c`

```
void setup_gpio_led(){
  // Green LED
  GPIO_PinModeSet(gpioPortB, 13,  gpioModePushPull, 1);
}
```

This function configures GPIO port B13 on the ARM core to be used to toggle the LED in the example application. The GPIO configuration function is provided within the XMOS development tools and is accessed by including the following header file:

```
#include <em_gpio.h>
```

2.6 Toggling the LED connected to the ARM GPIO

In order to toggle the LED connected to the ARM core it is required to change the value being driven to the GPIO port. This is done in the application within the following functions,

```
void set_led_on() {
    GPIO_PinOutClear(gpioPortB,  13);
}

void set_led_off() {
    GPIO_PinOutSet(gpioPortB,  13);
}
```

On the development board being used for this example the LED connected to the ARM core is active low so driving a 0 onto the GPIO pin to turn the LED on and a 1 onto the GPIO pin to turn it off. These functions are called from main() with a delay inbetween to control the speed of the LED flash.

2.7 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() {
  while (1) {
      set_led_on();
      delay_seconds(1);
      set_led_off();
      delay_seconds(1);
  }
  return 0;
}
```

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

- There is a while loop which executes forever
- Inside the while loop an LED is toggled on and off at a specific rate

- A library function is used to add a delay between toggling the LED on and off

2.8 Setting up the xCORE GPIO for the application

In order to use a GPIO port on the xCORE tile for toggling an LED it needs to be configured into the mode required. This is done in the file `main_xcore.xc` with the following code,

```
out port green_led = XS1_PORT_1F;
```

This configures the GPIO port XS1_PORT_1F on the xCORE tile as an output port with the symbolic name green_led. This port can then be used from the application to toggle the LED that is connected.

2.9 Toggling the LED connected to the xCORE GPIO

In order to toggle the LED connected to the xCORE GPIO it is required to change the value being driven to the GPIO port. This is done in the application within the following functions,

```
void set_led_on() {
    green_led <: 0;
}

void set_led_off() {
    green_led <: 1;
}
```

On the development board being used for this example the LED connected to the xCORE tile is active low so driving a 0 onto the GPIO pin will turn the LED on and a 1 onto the GPIO pin will turn it off. These functions are called from main() with a delay inbetween to control the speed of the LED flash.

APPENDIX A - Example Hardware Setup

This application example is designed to run on the xCORE-XA core module board which has an LED connected to both the xCORE and ARM GPIO ports. 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. This can be seen in the following image.

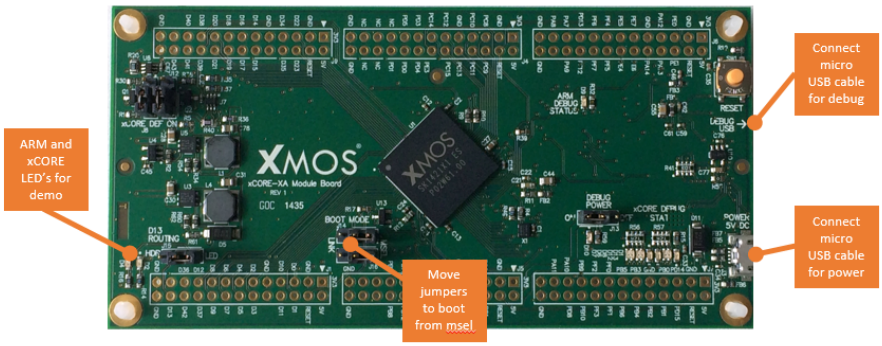

Figure 3: XMOS xCORE-XA core module board setup

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
- The ARM led will flash when the example is executing
- The xCORE led will flash when the example is executing

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.

B.1 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. 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_leds              <-- Download and execute the ARM code
> xrun app_xcore_xa_leds.xe           <-- Download and execute the xCORE code
```

Once these two commands have been executed the LED's connected to ARM core and xCORE tile will be flashing.

B.2 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. Selecting the xCORE binary, right click and then run as xCORE application will perform the same operation for the xCORE tile.

Once these two commands have been executed the LED's connected to ARM core and xCORE tile will be flashing.

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
#include <platform.h>
#include <xs1.h>

out port green_led = XS1_PORT_1F;

void set_led_on() {
    green_led <: 0;
}

void set_led_off() {
    green_led <: 1;
}

int main() {
  while (1) {
      set_led_on();
      delay_seconds(1);
      set_led_off();
      delay_seconds(1);
  }
  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"

void setup_gpio_led(){
  // Green LED
  GPIO_PinModeSet(gpioPortB, 13,  gpioModePushPull, 1);
}

void set_led_on() {
    GPIO_PinOutClear(gpioPortB,  13);
}

void set_led_off() {
    GPIO_PinOutSet(gpioPortB,  13);
}

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

  set_led_off();

  while (1) {
      for (unsigned int i = 0; i < 250000; i++) {}
```

```
            set_led_on();
        for (unsigned int i = 0; i < 250000; i++) {}
            set_led_off();
    }

    return 0;
}
```