Application Note: AN00145

# xCORE-XA - Power Management

This application note shows how to create a simple application which targets the XMOS xCORE-XA device and demonstrates how to implement power management of the xCORE component.

The code associated with this application note provides an example of using GPIO on the ARM core to enable and disable the 3v3 and 1v0 power supplies of the xCORE tile contained within the xCORE-XA device. The xCORE tile application code in this application note boots from the ARM core internal flash.

This simple example uses the XMOS tools features and tools libraries for xCORE-XA to develop this application and runtime tools to download and deploy code into the ARM flash.

In the application note we generate code that does not communicate inside the application between the ARM core and the xCORE tile in order to simply demonstrate the ability to manage the power supplies of the xCORE tile. There is however communication within the boot process to allow the xCORE tile to boot from the ARM core internal flash and this is handled by the libraries provided with the XMOS tool chain.

## Required tools and libraries

- xTIMEcomposer Tools - Version 13.2.0
- xCORE ARM Bridge library (XAB) - Provided with tools version 13.2.0
- xCORE ARM Boot library (XABoot) - Provided with 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 the application 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.
- This document assumes the reader has read and understood Application Note AN00141 xCORE-XA Application Development.
- 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 and also the API and description of the xCORE-XA bridge library and the xCORE-XA boot library, both of which are used in this example[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.

It is possible to control the power 1v0 and 3v3 power supplies of the xCORE tile from the ARM core GPIO given suitable support on the development board containing the xCORE-XA device. This allows for finer grained power control where the xCORE tile can be woken up when required to allow more intensive real time processing to take place. This software interface to this is provided within the XMOS development tools via the XABoot library.

## 1.2 Block diagram

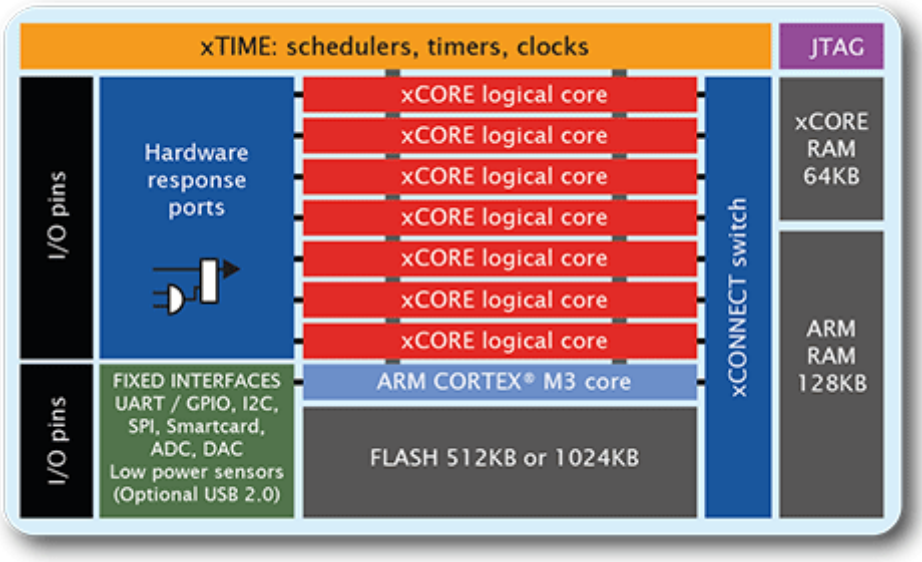


Figure 1: Block diagram of XMOS xCORE-XA microcontroller

# 2 xCORE-XA Power Management

The example in this application note does not have any external dependancies on application libraries other than those supplied with the XMOS development tools. It demonstrates how to manage control of the xCORE tile 1v0 and 3v3 power supplies using a GPIO on the ARM core. This application note uses the xCORE-ARM bridge library to allow the xCORE tile to communicate with the ARM core and the xCORE-ARM boot library to allow booting of the xCORE tile from the ARM core internal flash and also power control of the xCORE tile.

This example uses 2 tasks operating in parallel with one partitioned to run on the ARM core and one partitioned to run on the xCORE tile.

The tasks perform the following operations.

- A task executing a loop, flashing an LED and controlling the xCORE tile power supplies on the ARM core
- A task executing a loop and lighting a LED on the xCORE tile

This can be seen in the following task diagram.


Figure 2: Task diagram of xCORE-XA power management example

## 2.1 Makefile additions for booting the xCORE from the ARM flash

Further information about setting up a project to boot from the ARM flash can be found in the referenced XMOS tools user guide.

Within the makefile the following variable is set to mark that we want to to allow the xCORE to boot from the ARM flash. This will reserve space in the ARM image to embed the xCORE application image.:

```
EMBED_XCORE_IMAGE_ARM_IMAGE = 1
```

When using this flag it is also required to specify the size of the partition required within the ARM image to store the xCORE program code. This is specified by adding the following flag to the ARM_GCC_FLAGS variable in the makefile.:

```
-Wl,--defsym=XCORE_FLASH_SIZE=0x40000
```

In this example the flash specifies we want to reserve a space of 0x40000 hex bytes within the ARM program image.

Once these flags are set the tools will automatically embed the xCORE tile image into the ARM core image.

## 2.2 Declaring resource and setting up the ARM and xCORE

The demo code in this application note is split into 2 files containing the implementation of a simple main() function for both the ARM and xCORE. This application simply executes a loop which flashes an

LED on the ARM core while it turns on and off the xCORE tile. The xCORE application in this example is very simple and will light an LED to signal that it has powered on and booted correctly.

## 2.3 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);
  // Enabled EBI clock
  CMU_ClockEnable(cmuClock_EBI, true);
  setup_gpio();

  int count = 0;
  int xcore_on = 1;

  xaboot_power_off_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
  delay();
  xaboot_power_on_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
  delay();
  xaboot_boot_xcore();

  set_led_off();

  while (1) {
    if (count == 5) {
       xcore_on = !xcore_on;
       if (xcore_on) {
         xaboot_power_on_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
         delay();
         xaboot_boot_xcore();
       } else {
         xaboot_power_off_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
       }
       count = 0;
    }
    set_led_on();
    delay();
    set_led_off();
    delay();
    count++;
  }

  return 0;
}
```

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

- The GPIO clock for the ARM core is explicitly enabled
- The EBI clock for the ARM core is explicitly enabled
- A function is called to configure the GPIO ports we are using
- We turn the xCORE tile power state off and on to begin using the xaboot library power control functions
- We boot the xCORE tile from the ARM core flash using the xaboot library boot function
- There is a while loop which executes forever
- Inside the while loop we toggle an LED on and off at a specific rate

- After a number of iterations of the loop we toggle the xCORE tile power
- When we power the xCORE tile on we also boot the xCORE tile again

## 2.4 Setting up ARM GPIO for the application

In order to use a GPIO on the ARM core for toggling an LED and controlling the xCORE tile power they need to be configured into the mode we require. This is done from the function setup_gpio() called from main() in the file `main_arm.c`

```
void setup_gpio(){
  // Green LED
  GPIO_PinModeSet(gpioPortB, 13,  gpioModePushPull, 1);
  // xCORE 3v3
  GPIO_PinModeSet(gpioPortD, 2,  gpioModePushPull, 1);
  // xCORE 1v0
  GPIO_PinModeSet(gpioPortD, 10,  gpioModePushPull, 1);
}
```

This function configures the following ARM GPIO ports,

- GPIO port B13 on the ARM core to be used to toggle the LED in the demo application.
- GPIO port D2 is used to control the xCORE tile 3v3 power supply
- GPIO port D10 is used to control the xCORE tile 1v0 power supply

The GPIO configuration functions are provided within the XMOS development tools and are accessed by including the following header file:

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

## 2.5 Controlling the ARM GPIO

In order to toggle the LED connected to the ARM core we need to change the value we are driving 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 we are using for this example the LED connected to the ARM core is active low so we drive 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.

In order to control the GPIO ports which control the xCORE tile power supplies we use the functions provided within the xCORE ARM boot library. These take configured GPIO ports as arguments and manipulates the ports inside the library code. It also needs to be informed if either of the power supplies are active high or low, this is also specified as an argument to the power control functions. In this example the 3v3 power supply is active low and the 1v0 power supply is active high.

In order to use these power control functions the following header file needs to be included:

```
#include <XA/xaboot.h>
```

To turn the xCORE tile power on we use the following function

```
xaboot_power_on_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
```

To turn the xCORE tile power off we use the following function

```
xaboot_power_off_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
```

We pass the GPIO ports we have configured earlier in the code to these functions and they are responsible for setting the state correctly to control the xCORE tile power supplies.

## 2.6 The xCORE application main() function

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

```
int main() {
  set_led_on();
  while (1);
  return 0;
}
```

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

- An LED is set on to signal the start of main()
- There is a while loop which executes forever

## 2.7 Setting up the xCORE GPIO for the application

In order to use a GPIO on the xCORE tile for toggling an LED it needs to be configured into the mode we require. 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 which it is connected to it.

## 2.8 Turning on the xCORE connected LED

In order to toggle the LED connected to the xCORE tile GPIO we need to change the value we are driving to the GPIO port. This is done in the application within the following functions,

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

On the development board we are using for this example the LED connected to the xCORE tile is active low so we drive a 0 onto the GPIO pin to turn the LED on and a 1 onto the GPIO pin to turn it off. This function is called from main().

# 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 tile and the ARM core. 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.

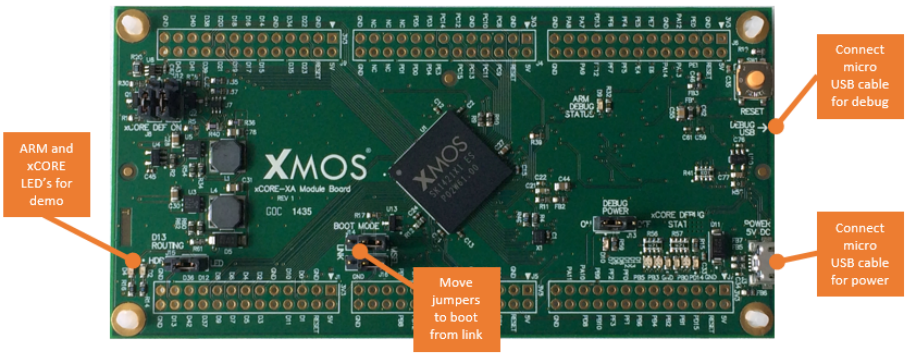


Figure 3: XMOS xCORE-XA core module board setup

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

- Both the power and debug adapter USB cables should be connected
- The boot mode of the board should have the jumpers selecting boot from link
- The ARM core led will flash during the demo, the xCORE tile LED will light when it is powered on by the ARM core

# APPENDIX B - Launching the demo application

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio we can execute the application 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 and xCORE devices plus the combined image where the xCORE application is embedded into the ARM application. 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 can be used to download code only to the ARM core as the xCORE tile will be getting its application code from the ARM core flash. If we change into the bin directory of the project we can execute the code on the xCORE-XA microcontroller as follows:

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

Once this command has finished the ARM flash will be programmed with both the ARM application code and the xCORE application code and the application will be running.

## B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio the run mechanism is used to download code to the ARM core within the xCORE-XA. If we select the ARM binary contained within the project bin directory which contains the xcore image `app_xa_xcore_power_with_xcore_image`, right click and then run as ARM application, the application will be downloaded and executed on the ARM core.

Once this command has finished the ARM flash will be programmed with both the ARM application code and the xCORE application code and the application will be running.

# 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 <xs1.h>

out port green_led = XS1_PORT_1F;

void set_led_on() {
    green_led <: 0;
}

int main() {
  set_led_on();
  while (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"
#include "XA/xaboot.h"

void setup_gpio(){
  // Green LED
  GPIO_PinModeSet(gpioPortB, 13,  gpioModePushPull, 1);
  // xCORE 3v3
  GPIO_PinModeSet(gpioPortD, 2,  gpioModePushPull, 1);
  // xCORE 1v0
  GPIO_PinModeSet(gpioPortD, 10,  gpioModePushPull, 1);
}

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

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

void delay() {
  for (unsigned int i = 0; i < 250000; i++) {}
}

int main(void) {
  // Enable GPIO clock
  CMU_ClockEnable(cmuClock_GPIO, true);
  // Enabled EBI clock
  CMU_ClockEnable(cmuClock_EBI, true);
  setup_gpio();

  int count = 0;
  int xcore_on = 1;
```

```
  xaboot_power_off_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
  delay();
  xaboot_power_on_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
  delay();
  xaboot_boot_xcore();

  set_led_off();

  while (1) {
    if (count == 5) {
       xcore_on = !xcore_on;
       if (xcore_on) {
         xaboot_power_on_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
         delay();
         xaboot_boot_xcore();
       } else {
         xaboot_power_off_xcore(gpioPortD, 10, 1, gpioPortD, 2, 0);
       }
       count = 0;
    }
    set_led_on();
    delay();
    set_led_off();
    delay();
    count++;
  }

  return 0;
}
```