Application Note: AN00144

xCORE-XA - xCORE ARM Boot Library

This application note shows how to create a simple application 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 using the xCORE ARM Boot (XABoot) library so that the xCORE tile can be booted by the ARM core from the internal flash on the xCORE-XA. In doing so, the application note also demonstrates how the xCORE tile binary can be stored within the ARM flash instead of having a seperate SPI device.

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.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2.0 or later
- 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 assuming the xCORE-XA bootloader has been written to the OTP and the mode pins are set to boot the xCORE tile from xCONNECT Link.

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.
- This document extends Application Note AN00142 Using the XAB Library.
- 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.

The xCORE ARM XABoot library is used to boot the xCORE tile inside an xCORE-XA device from the ARM core. The xCORE factory and upgrade images can be stored within the ARM core internal flash memory rather than an external SPI device.

1.2 Block diagram

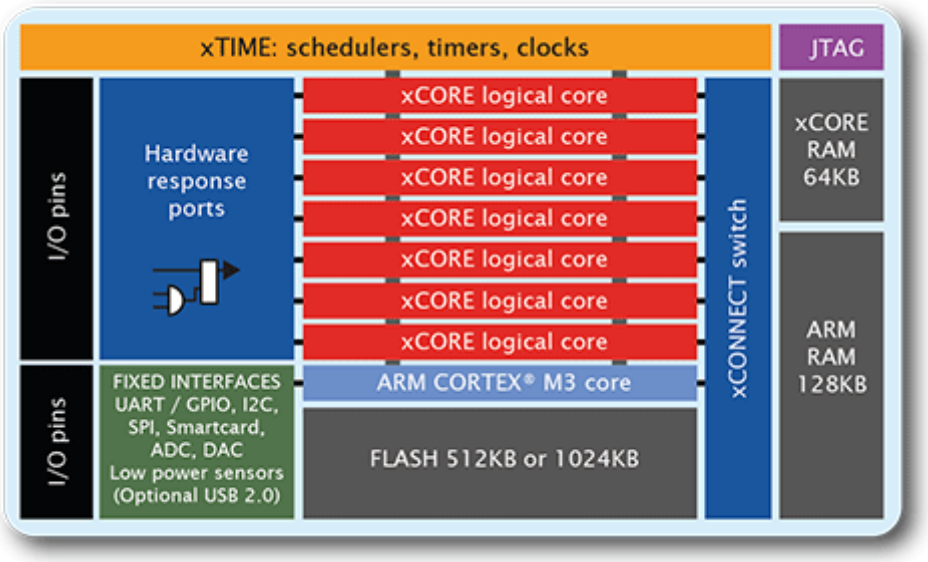

Figure 1: Block diagram of XMOS xCORE-XA microcontroller

2 xCORE-XA - xCORE ARM Boot Library

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 use the XABoot library supplied with the XMOS development tools to boot the xCORE tile from the ARM core.

There are 3 tasks operating in this example with one partitioned to run on the ARM core, one partitioned to run on the xCORE tile for the user code and a second task on the xCORE tile to interface with the ARM core.

The tasks perform the following operations.

- A task that boots the xCORE tile and then executes a loop to send and receive messages using the XAB library and toggling an LED on the ARM core
- A task executing a loop to send and receive messages using the XAB library and toggling an LED on the xCORE tile
- A task executing the hardware interface between the xCORE tile and the ARM core EBI

This can be seen in the following task diagram.

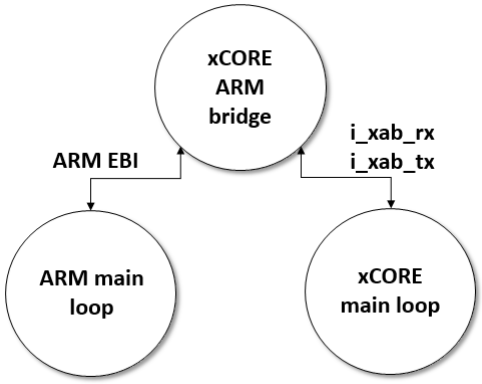

Figure 2: Task diagram of xCORE-XA xCORE ARM Boot application example

2.1 Makefile support for using the XABoot library

The makefile variable EMBED_XCORE_IMAGE_IN_ARM_IMAGE is used to select if the build system will automatically generate a binary containing both the ARM core and the xCORE tile application code. The binary generated will have the extension -with-xcore-image applied to the end of the project name. In this application note this variable is set to 1 indicating that we want to store the xCORE tile image within ARM core flash memory.

Variable to embed the xCORE image in the ARM binary:

```
EMBED_XCORE_IMAGE_IN_ARM_IMAGE = 1
```

It is a requirement to tell only the ARM toolchain that the XABoot library is being used. This is achieved by passing the option -lxaboot with the associated build flags. It is also a requirement to tell the ARM toolchain how much flash space should be reserved for the xCORE images. This is achieved by passing the options -Wl and–defsym=XCORE_FLASH_SIZE=0x40000. In this application note example 0x40000 bytes of ARM flash memory are being reserved.

Options passed to ARM build tools from makefile:

```
ARM_GCC_FLAGS = -std=c99 -O0 -g -lxab -Wl,--defsym=XCORE_FLASH_SIZE=0x40000 -lxaboot
```

2.2 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 main() function for both the ARM core and xCORE tile. This application sees the ARM core use the XABoot library to boot the xCORE tile. Once booted, both the ARM core and xCORE tile use the XAB library to exchange messages and toggle the status of an LED as described in application note AN00142.

2.3 The ARM core application main() function

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

```
int main(void) {

  // The XABoot library requires GPIO and EBI clocks to be enabled
  CMU_ClockEnable(cmuClock_GPIO, true);
  CMU_ClockEnable(cmuClock_EBI, true);

  //Boot the xCORE tile
  xaboot_boot_xcore();

  //Increase the ARM core frequency to 48MHz
  set_core_freq();

  setup_gpio_led();
  set_led_off();

  xab_connect();
  xa_demo_arm();
  xab_disconnect();

  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
- An XABoot library function is called to boot the xCORE tile
- A function is called to configure the ARM core clock speed
- A function is called to configure the GPIO ports used by the example code
- An XAB library function is called to connect to the xCORE tile
- A function is called to send and receive data to the xCORE tile and to toggle the LED on the ARM core
- An XAB library function is called to disconnect from the xCORE tile

2.4 Setting up the ARM GPIO and EBI

The xCORE tile and ARM core communicate with each other via the EBI bus connected within the xCORE-XA device. The XABoot library uses the ARM GPIO and EBI and the associated clocks must be enabled using the ARM function `CMU_ClockEnable`.

```
CMU_ClockEnable(cmuClock_GPIO, true);
CMU_ClockEnable(cmuClock_EBI, true);
```

This function is contained within the ARM header file em_cmu.h provided within the XMOS development tools.

```
#include "em_cmu.h"
```

2.5 Including the XABoot header file

In order to use the XABoot library on the ARM, the header file XA/xaboot.h must be included.

```
#include "XA/xaboot.h"
```

2.6 Using the XABoot library to boot the xCORE tile

The XABoot library function xaboot_boot_xcore() is used by the ARM core to boot the xCORE tile.

```
xaboot_boot_xcore();
```

Note how the set_core_freq() function is not called until after the xaboot_boot_xcore() function has returned. It is advised that the ARM core is running at its default frequency of 13MHz when booting the xCORE tile.

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 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. The boot_mode pins should also be set to boot from ARM.

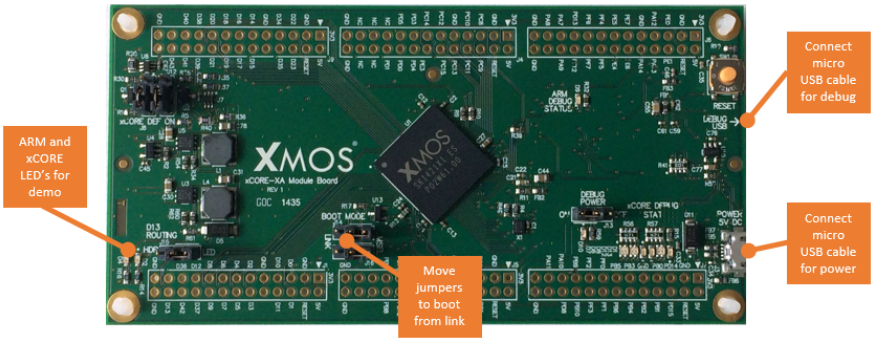

Figure 3: XMOS xCORE-XA core module board setup

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 link
- The ARM core led will flash during the example, the xCORE tile LED will light when it is powered on by the ARM core

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 it 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. There will also be a binary with the extension `-with-xcore-image` applied to the end of the project name. This binary contains both the ARM core and xCORE tile images in one file.

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 to the ARM core. If we change into the bin directory of the project the example can be executed on the xCORE-XA microcontroller as follows:

```
> xrun app_xcore_xa_boot-with-xcore-image        <-- Download and execute the ARM code
```

Once the command has been executed and the xCORE-XA device has been reset the LED's connected to ARM core and xCORE tile GPIO should be flashing.

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio the run mechanism can be used to download code to the ARM core. By Selecting the ARM binary with the `-with-xcore-image` extension contained within the project bin directory, right click and then run as ARM application, the application will be downloaded and executed on the ARM.

Once the command has been executed and the device has been reset the LED's connected to ARM core and xCORE tile GPIO should 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>
#include "XA/xab.h"
#include "app_common.h"

out port green_led = XS1_PORT_1F;

void set_led_on() {
    green_led <: 0;
}

void set_led_off() {
    green_led <: 1;
}

void xa_demo_xcore(client xab_read_if xab_rx, client xab_write_if xab_tx)
{
  char buf[MAX_TRANSFER_SIZE];
  size_t rx_size;

  delay_milliseconds(100);

  // Fill the buffer with dummy data
  buf[0] = 0x2;

  // Signal start
  xab_tx.write(buf, 1);

  while (1) {
    // wait for event from ARM bridge
    select {

    // Event when a packet comes back from the ARM
    case xab_rx.data_ready():

      // Read the data from the ARM
      rx_size = xab_rx.read(buf, 1);

      // Turn on led
      set_led_on();

      // Wait a bit...
      delay_milliseconds(250);

      // Fill the buffer with dummy data
      buf[0] = 0x2;

      // Send packet to the ARM
      xab_tx.write(buf, 1);

      // Turn off led
      set_led_off();
```

```
      break;
    }
  }
}

xab_ports_t xab_ports = on tile[0]: XAB_PORTS_INITIALIZER(XS1_CLKBLK_1);

int main(){
  interface xab_read_if i_xab_rx;
  interface xab_write_if i_xab_tx;
  par {
    xcore_arm_bridge(xab_ports, i_xab_rx, i_xab_tx, MAX_TRANSFER_SIZE);
    xa_demo_xcore(i_xab_rx, i_xab_tx);
  }
  return 0;
}
```

D.2 Source code for main_arm.c

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include "XA/xab.h"
#include "XA/xaboot.h"
#include "em_gpio.h"
#include "em_cmu.h"
#include "app_common.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);
}

void xa_demo_arm(void)
{
  char buf[MAX_TRANSFER_SIZE];
  size_t rx_size;

  while (1) {

    // Read the data from the xCORE
    rx_size = xab_read(buf, MAX_TRANSFER_SIZE);

    // Turn LED on
    for (unsigned int i = 0; i < 250000; i++) {
      set_led_on();
    }

    // Fill the buffer with dummy data
    buf[0] = 0x1;
```

```
    // Send packet to the xCORE
    xab_write(buf, 1);

    // Turn off LED
    set_led_off();
  }
}

void set_core_freq()
{
  CMU_ClockDivSet(cmuClock_HF, cmuClkDiv_1);
  CMU_OscillatorEnable(cmuOsc_HFXO, true, true);
  CMU_ClockSelectSet(cmuClock_HF, cmuSelect_HFXO);
}

int main(void) {

  // The XABoot library requires GPIO and EBI clocks to be enabled
  CMU_ClockEnable(cmuClock_GPIO, true);
  CMU_ClockEnable(cmuClock_EBI, true);

  //Boot the xCORE tile
  xaboot_boot_xcore();

  //Increase the ARM core frequency to 48MHz
  set_core_freq();

  setup_gpio_led();
  set_led_off();

  xab_connect();
  xa_demo_arm();
  xab_disconnect();

  return 0;
}
```