Application Note: AN00142

xCORE-XA - xCORE ARM Bridge 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 Bridge (XAB) library to pass data between the xCORE tile and the ARM core within an xCORE-XA 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
- xCORE ARM Bridge library (XAB) - 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 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.
- 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 which is 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 tile and ARM core communicate with each other via the EBI bus connected within the xCORE-XA device. The XAB library provides a software layer to control this communication. On the ARM core, this is accessed via a C-api which manages the EBI hardware and interrupts. It also has capacity to use the DMA unit of the device which is covered in AN00143. On the xCORE tile, the bridge code runs in a separate task that uses a logical core. The application can connect to this task on the xCORE tile to send and receive data.

1.2 Block diagram

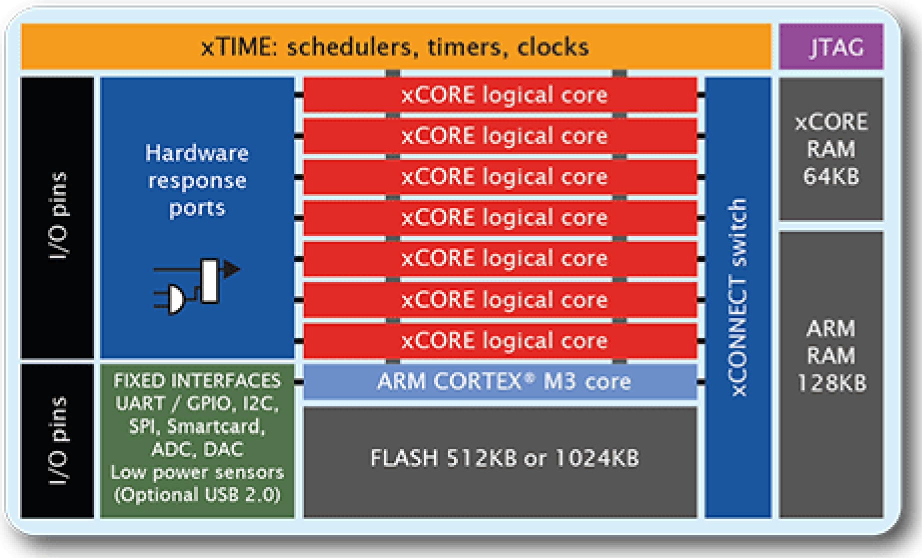

Figure 1: Block diagram of XMOS xCORE-XA microcontroller

2 xCORE-XA - xCORE ARM Bridge 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 XAB library supplied with the XMOS development tools to pass data between the xCORE tile and ARM core.

The example provided has 3 tasks executing, one partitioned to run on the ARM core and two partitioned to run on the xCORE tile, one to run the example application and one to provide the interface between the xCORE tile and the ARM core EBI. The tasks perform the following operations.

- A task executing 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.

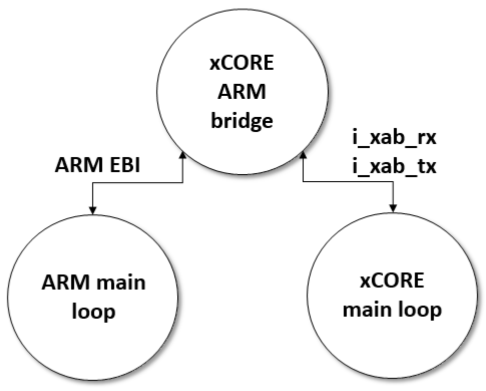

Figure 2: Task diagram of xCORE ARM Bridge example

2.1 Makefile support for using the XAB library

It is a requirement to tell both the xCORE toolchain and ARM toolchain that the XAB library is being used. This is achieved by passing the option -lxab with the associated build flags.

Options passed to xCORE build tools from makefile:

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

Options passed to ARM build tools from makefile:

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

2.2 Declaring resource and setting up the ARM and xCORE

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 demonstrates communication between the ARM core and xCORE tile using the XAB library. As these messages are sent back and forth both the

ARM core and the xCORE tile toggle the status of an LED to indicate these messages are successfully sent and received.

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)
{
  set_core_freq();

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

  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:

- A function is called to configure the ARM core clock speed
- 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 being used
- An XAB library function is called to connect to the xCORE tile
- A function is called to exchange data with 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 XAB library uses the ARM GPIO and EBI and must enable their clocks 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 XAB header file

In order to use the XAB library on the ARM core, the header file `XA/xab.h` must be included.

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

2.6 Using the XAB library to connect the ARM core to the xCORE tile

The XAB library function xab_connect() is used to connect the ARM core to the xCORE tile and synchronize with the xCORE tile so that they can communicate with each other. This function must be called before any other functions within the XAB library can be used.

2.7 Using the XAB library to receive data from the xCORE tile

To receive data from the xCORE tile the XAB library function xab_read is used. This function will block until data is available from the xCORE tile.

```
rx_size = xab_read(buf, MAX_TRANSFER_SIZE);
```

To use xab_read a buffer must be provided for the data to be received into.

```
char buf[MAX_TRANSFER_SIZE];
```

The value for MAX_TRANSFER_SIZE has been defined within the common header file app_common.h. By defining this value here and including it with both the ARM and xCORE source files if ensures both the ARM core and xCORE tile can never send or receive more data than can be handled in a single transaction.

The return value from xab_read will indicate how many bytes it actually received.

2.8 Using the XAB library to send data to the xCORE tile

To send data to the xCORE tile the XAB library function xab_write is used. The function will block until buffer space becomes available on the xCORE tile.

```
xab_write(buf, 1);
```

To use xab_write it is required to store the data to send in a buffer. In this application note the same buffer that was used in the xab_read function is used. The function xab_write must also be told how many bytes are being sent.

2.9 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(){
  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;
}
```

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

- An interface connection is created for reading via the XAB library
- An interface connection is created for writing via the XAB library
- A par statement is used to launch two tasks in parallel
- The XAB library function xcore_arm_bridge() is called in one task
- The function xa_demo_xcore() is called in a second task

2.10 Including the XAB header file

In order to use the XAB library on the xCORE tile, the header file XA/xab.h must be included.

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

2.11 Setting up the bridge on the xCORE tile

The XAB library provides the function xcore_arm_bridge for providing the xCORE tile a connection to the ARM core via the internal EBI bus. The function xcore_arm_bridge executes as a task and must be present in the top level par statement of the program. In this application note the xcore_arm_bridge task executes in parallel with the task xa_demo_xcore which performs the application level communication with the the ARM core.

```
par {
  xcore_arm_bridge(xab_ports, i_xab_rx, i_xab_tx, MAX_TRANSFER_SIZE);
  xa_demo_xcore(i_xab_rx, i_xab_tx);
}
```

To use the XAB library function xcore_arm_bridge the function needs to be passed which xCORE tile GPIO ports are connected to the EBI bus. The XAB library provides a macro XAB_PORTS_INITIALIZER that initializes the ports. The XAB library also provides the type xab_ports_t for convieniently holding the ports. In this application note the ports are declared and initialised at global scope.

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

Two interface connections are also required, one for reading data from the ARM core and one for writing data to the ARM core via the bridge. The XAB library provides the interface xab_read_if that enables the reading of data from the ARM core. The XAB library also provides the interface xab_write_if that enables the xCORE tile to write to the ARM core. In the example provided these interfaces are created outside of the par statement and are then passed to both tasks within the par statement.

```
interface xab_read_if i_xab_rx;
interface xab_write_if i_xab_tx;
```

Finally, the XAB library function xcore_arm_bridge requires to be passed the maximum size of a data transfer that can be communicated with the ARM core. The value for MAX_TRANSFER_SIZE has been defined within the common header file app_common.h. By defining this value here and including it with both the ARM and xCORE source files it ensures both the ARM core and xCORE tile can never send or receive more data than can be handled in a single transaction.

2.12 Using the XAB library to send data to the ARM core

In the application note code the task xa_demo_xcore is being used to perform the sending and receiving of data via the bridge.

```
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;
    }
  }
}
```

The task xa_demo_xcore takes the interface connection xab_write_if as one of its parameters. The application note code will use this interface connection to send data to the ARM core. To send data, create a buffer and fill it with some dummy data. The buffer must not be any bigger than the MAX_TRANSFER_SIZE defined in app_common.h

```
char buf[MAX_TRANSFER_SIZE];
```

The example code then uses the write function on the xab_write_if interface to send this data to the ARM core.

```
xab_tx.write(buf, 1);
```

2.13 Using the XAB library to receive data from the ARM core

In this example the task xa_demo_xcore is being used to perform communication to the ARM core via the bridge.

```
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;
    }
  }
}
```

The task xa_demo_xcore takes the interface connection xab_read_if as one of its parameters. The example code will use this interface connection to receive data from the ARM core. The interface connection will signal an event when data is available to be received from the ARM core. The application note waits on this event by using the select statement and a case statement that calls the function data_ready available on the xab_read_if interface. Once data has been signalled as being available, the function read from the xab_read_if interface is called. The read call returns the number of bytes read from the

ARM core. The data that was read will be stored within the buffer buf that was passed into the read call.

```
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);
```

APPENDIX A - Example Hardware Setup

This example is designed to run on the xCORE-XA core module board which has an LED connected to both the xCORE and the 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.

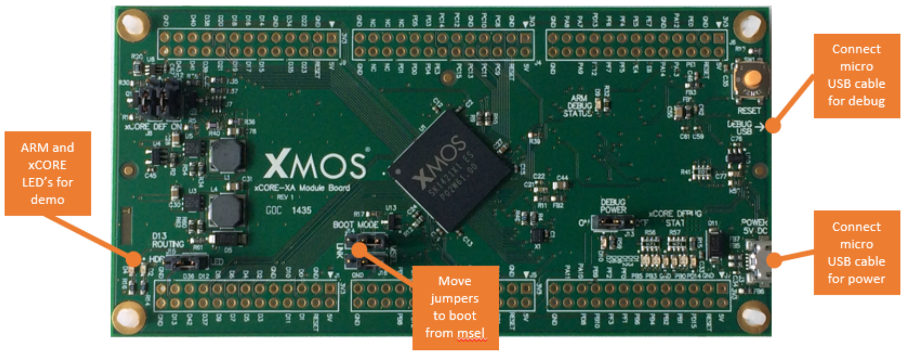
ARM and xCORE LED's for demo
Move jumpers to boot from msel
Connect micro USB cable for debug
Connect micro USB cable for power

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 core LED will flash during when the example example executes in sequence with the xCORE tile LED
- The xCORE tile LED will flash during the when the example executes in sequence with the ARM core LED

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 can be used to download code to both the ARM core and xCORE tile. By changing into the bin directory of the project the example application can be executed on the xCORE-XA microcontroller as follows:

```
> xrun app_xcore_xa_xab              <-- Download and execute the ARM code
> xrun app_xcore_xa_xab.xe           <-- Download and execute the xCORE code
```

Once these two commands have been executed the LED's connected to the ARM core and xCORE tile GPIO should 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 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 "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)
{
  set_core_freq();

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

  setup_gpio_led();
  set_led_off();

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

  return 0;
}
```