Application Note: AN00185

Boot an xCORE-200 device from QuadSPI flash memory

This applcation note shows how to program an application into QuadSPI flash memory and therefore boot the application from QuadSPI flash memory on an XMOS xCORE-200 device.

Required tools and libraries

- xTIMEcomposer Tools Suite version 14.0 or later is required.

Required hardware

This application note is designed to run on an XMOS xCORE-200 series device.

The example code provided with the application has been implemented and tested on the xCORE-200 explorerKIT 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-200 series device.

Prerequisites

- This document assumes familiarity with the XMOS xCORE-200 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 familiarity with QuadSPI flash memory, the xCORE quadflash library and the XMOS tool XFLASH.
- 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 devices[2].

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

1 Introduction

xCORE-200 explorerKIT contains everything you need to start developing applications on the powerful xCORE-200 multicore microcontroller products from XMOS. It's easy to use and provides lots of advanced features on a small, low cost platform.

The xCORE-200 explorerKIT features our XE216-512 xCORE-200 multicore microcontroller. This device has sixteen 32bit logical cores that deliver up to 2000MIPS completely deterministically. The combination of 100/1000 Mbps Ethernet, high speed USB and 53 high performance GPIO make the xCORE-200 explorerKIT an ideal platform for functions ranging from robotics and motion control to networking and digital audio.

1.1 Block diagram

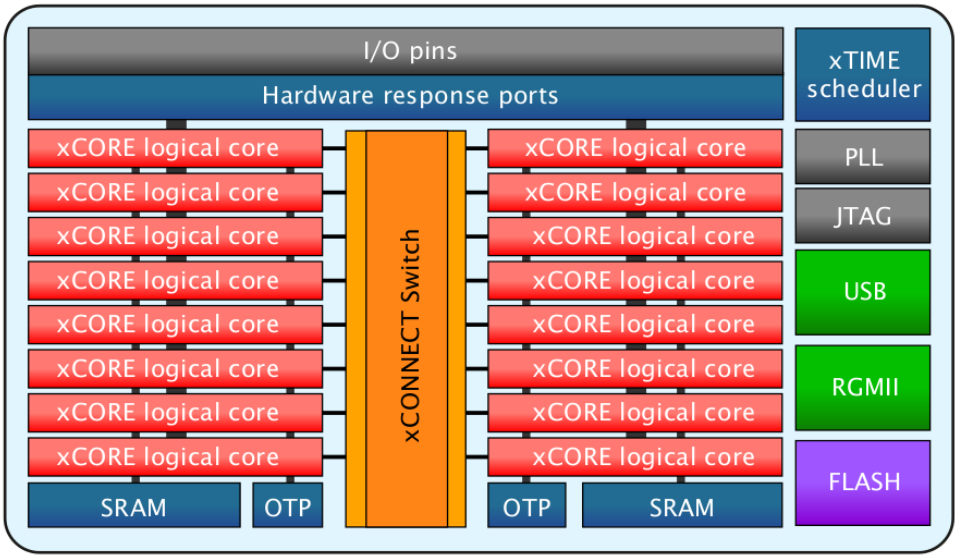

Figure 1: Block diagram of XE216-512 device on xCORE-200 explorerKIT

1.2 QuadSPI flash memory

The xCORE-200 explorerKIT features a QuadSPI flash device providing a 4-bit multiplexed I/O serial interface to boost performance while maintaining the compact form factor of standard serial flash devices. This allows the xCORE-200 device to load data from the data partition of flash memory significantly faster than with traditional SPI flash memory devices.

The QuadSPI flash memory is logically split between a boot and data partition.

The boot partition consists of a flash loader followed by a factory image and zero or more optional upgrade images. Each image starts with a descriptor that contains a unique version number and a header that contains a table of code/data segments for each tile used by the program and a CRC.

This application note demonstrates

- How to configure an XN file so that an xCORE-200 device can boot from QuadSPI flash memory.
- How to set the boot mode of the xCORE-200 device so that the boot ROM will boot from QuadSPI flash memory.
- How to use the XFLASH to program an application to QuadSPI flash memory.

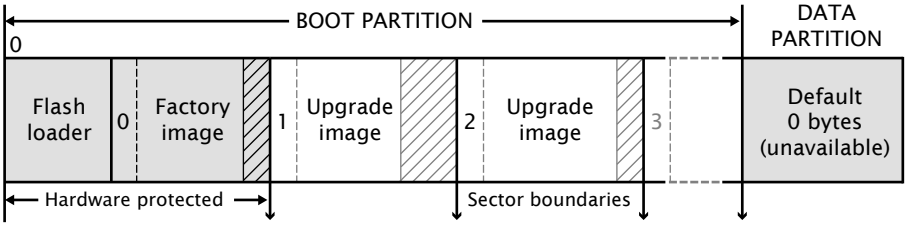

Figure 2: Flash format diagram

2 Booting the xCORE-200 from QuadSPI flash memory

2.1 Source code structure for this application note

This application note example is stored in the following directory structure:

```
src                                  <-- top level project source directory
  boot_xcore_200_qspi.xc             <-- source file for project
  XCORE-200-EXPLORER.xn              <-- xn file for project
Makefile                             <-- make file for project
```

2.2 xCORE-200 Boot Mode Selection

The 4-bit port XS1_PORT_4B is used to decide on how to boot an xCORE-200 series device. This port has pull downs enabled, so its default state is 0b0000. Bits can be pulled high by strapping a 10K resistor to 3V3. The boot modes are encoded as follows:

2	1	0	Core 0	Core 1	Switch
0	0	0	QSPI	Channel End	
0	0	1	SPI Master	Channel End	
0	1	0	SPI Slave	Channel End	
0	1	1	SPI Slave	SPI Slave	
1	0	0	Channel End	Channel End	link 0 enabled in 2W
1	0	1	Channel End	Channel End	links 4..7 enabled in 5W
1	1	0	Channel End	Channel End	links 1,2,5,6 enabled in 5W
1	1	1	Channel End	Channel End	links 0..3 enabled in 5W

Table 1: Boot Modes for xCORE-200 series devices

In this application note the default state 0b0000 on the 4-bit port XS1_PORT_4B is used to boot from QuadSPI.

2.3 XN support for QuadSPI flash memory

The example code in this application note uses the XCORE-200-EXPLORER.xn target XN file.

Looking at this XN file in more detail you can see that the xCORE-200 Node element with attribute `Id` value of “0” has a `Boot->Source` element defined, where the `Location` attribute of `Source` has a value of “bootFlash”.

The `Location` attribute value instructs that this Node will boot from a `Device` element with a `Name` attribute also set to “bootFlash”. Note that only one Node element can boot from each defined `Device` element.

In this XN file it can be seen that a `Device` element has been added inside the element `ExternalDevices` with a `Name` attribute value of “bootFlash”.

The `Device` element uses the attribute `Class` with the value “SQIFlash” to indicate that this is a QuadSPI device. Based on this `Class` attribute value all tools within xTIMEcomposer will now expect this Node to boot from QuadSPI.

It can also be seen in this XN example that three ports are required to communicate with the QuadSPI

device. These three ports are defined in the xCORE-200 Node element with Id attribute value of “0” and Tile element with Number attribute value of “0”. The equivalent names must also be defined within the Device element, tying the xCORE-200 to the QuadSPI device.

The Port element Location attribute with value of “XS1_PORT_1B” has a Name attribute with the value of “PORT_SQI_CS”. This describes that the 1-bit port “XS1_PORT_1B” will be used to control the Chip Select pin of the QuadSPI device. An Attribute element is also defined within the Device element with a matching Name attribute with the valie of “PORT_SQI_CS”.

The Port element Location attribute with value of “XS1_PORT_1C” has a Name attribute with the value of “PORT_SQI_SCLK”. This describes that the 1-bit port “XS1_PORT_1C” will be used to control the clock pin of the QuadSPI device. An Attribute element is also defined within the Device element with a matching Name attribute with the valie of “PORT_SQI_SCLK”.

The Port element Location attribute with value of “XS1_PORT_4B” has a Name attribute with the value of “PORT_SQI_SIO”. This describes that the 4-bit port “XS1_PORT_4B” will be used to send data too and read data from the QuadSPI device. An Attribute element is also defined within the Device element with a matching Name attribute with the valie of “PORT_SQI_SIO”.

2.4 Driving LED patterns on the xCORE-200-EXPLORER board

The example code in this application note example is used to drive LED patterns on the xCORE-200 explorerKIT core module board. Upon reset, the application will be booted from the QuadSPI device.

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

#define DELAY 25000000

on tile[0] : port led_ports = XS1_PORT_4F;

int main(void)
{
  par
  {
    on tile[0]:
    {
      char led_pattern[8];
      int time;
      int i = 0;
      timer t;

      char red   = 8;
      char green = 4;
      char blue  = 2;

      char single = 1;

      led_pattern[0] =          red;
      led_pattern[1] = single | red | green;
      led_pattern[2] =                green;
      led_pattern[3] = single |       green | blue;
      led_pattern[4] =                        blue;
      led_pattern[5] = single | red |         blue;
      led_pattern[6] =          red | green | blue;
      led_pattern[7] = single;
```

```
      t :> time;

      while (1) {
        led_ports <: led_pattern[i];
        i++;
        if (i == 8) {
          i = 0;
        }

        t when timerafter(time + DELAY) :> time;
      }
    }

    on tile[1]:
    {
      while(1);
    }
  }

  return 0;
}
```

APPENDIX A - Example Hardware Setup

This application example is designed to run on the xCORE-200 explorerKIT core module board. The xCORE-200 explorerKIT 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.

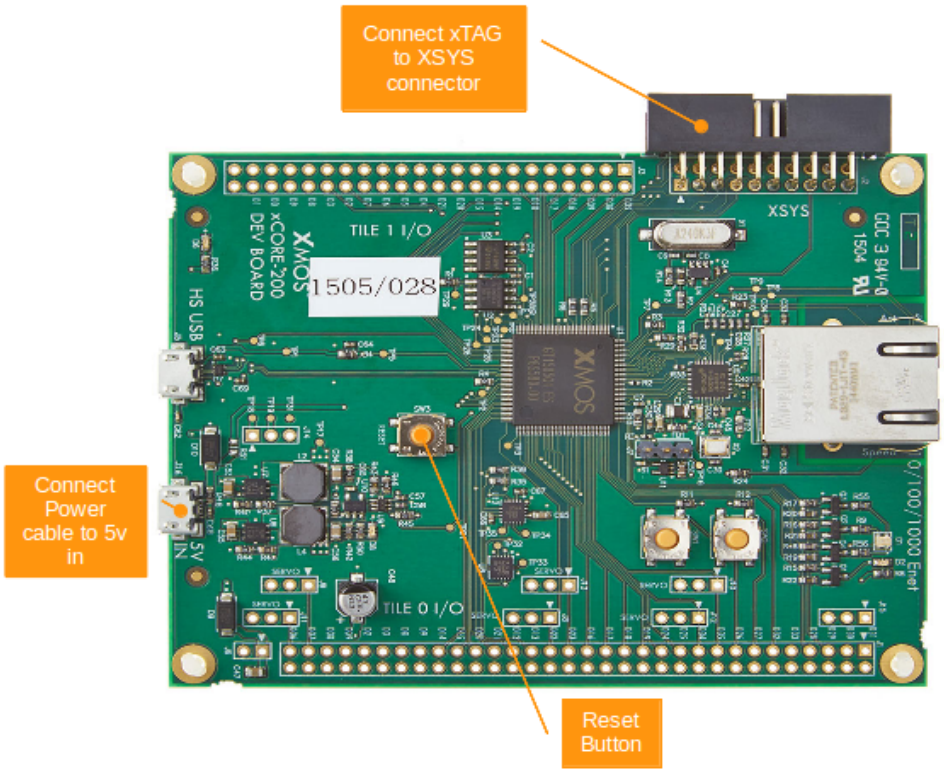

Figure 3: XMOS xCORE-200 explorerKIT

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

- The XTAG debug adapter should be connected to the XSYS connector and the XTAG USB cable should be connected to the host machine
- The xCORE-200 explorerKIT should have the power cable connected
- The RESET button can be used to repeatedly boot the xCORE-200 device from QuadSPI

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 written to the QuadSPI flash memory of the xCORE-200 explorerKIT module board.

Once built there will be a bin directory within the project which contains the binary for the xCORE-200 tile. The xCORE-200 binary has a XMOS standard .xe extension.

B.1 Writing to flash memory from the command line

From the command line the XFLASH tool is used to download the code to the QuadSPI flash device on the xCORE-200 explorerKIT core module board. The complete XFLASH command line is as follows:

```
> xflash --boot-partition-size 0x80000 bin/boot_xcore_200_qspi.xe
```

B.2 Writing to flash memory from xTIMEcomposer Studio

From xTIMEcomposer Studio there is the Flash As mechanism to edit the flash configuration and download the code to the flash device on the xCORE-200 explorerKIT core module board. Within the flash configuration editor the XFLASH options can be explicitly set. In this example, the Boot Partition Size checkbox is ticked and a corresponding value of 0x80000 is inserted.

Once the XFLASH command has been executed successfully, the application note example will have been written to QuadSPI flash memory. XFLASH will automatically reset the xCORE-200 device and subsequently boot the application note example application from QuadSPI.

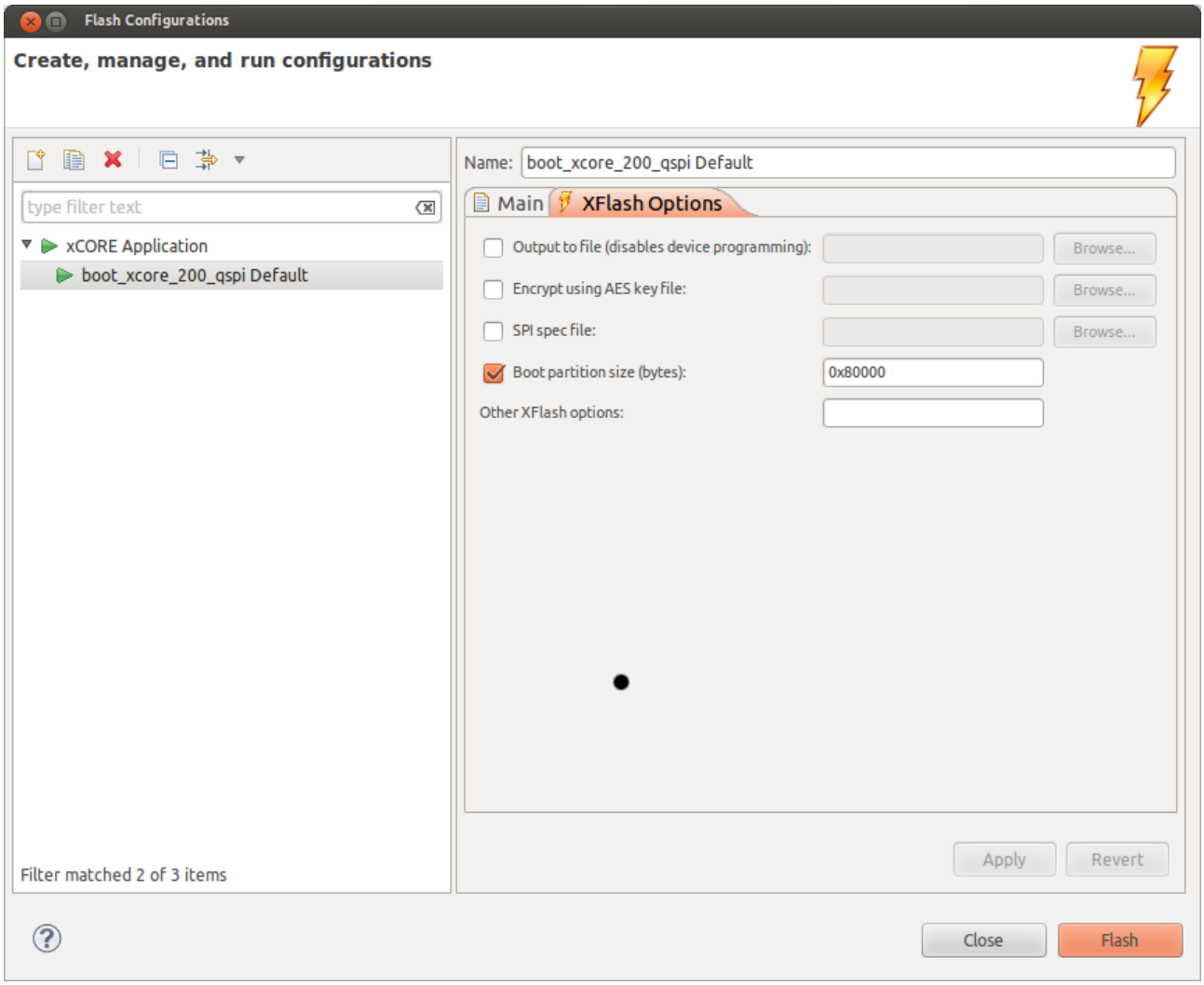

Figure 4: xTIMEcomposer studio flash configuration editor

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 XN for XCORE-200-EXPLORER.xn

D.2 Source code for boot_xcore_200_qspi.xc

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

#define DELAY 25000000

on tile[0] : port led_ports = XS1_PORT_4F;

int main(void)
{
  par
  {
    on tile[0]:
    {
      char led_pattern[8];
      int time;
      int i = 0;
      timer t;

      char red   = 8;
      char green = 4;
      char blue  = 2;

      char single = 1;

      led_pattern[0] =          red;
      led_pattern[1] = single | red | green;
      led_pattern[2] =                green;
      led_pattern[3] = single |       green | blue;
      led_pattern[4] =                        blue;
      led_pattern[5] = single | red |         blue;
      led_pattern[6] =          red | green | blue;
      led_pattern[7] = single;

      t :> time;

      while (1) {
        led_ports <: led_pattern[i];
        i++;
        if (i == 8) {
          i = 0;
        }

        t when timerafter(time + DELAY) :> time;
      }
    }

    on tile[1]:
    {
      while(1);
    }
```

```
  }

  return 0;
}
```